lwIP

COLLABORATORS

	TITLE : lwIP		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 16, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Overview **1**

1.1 Upgrading . . . 3
1.2 Changelog . . . 9
1.3 How to contribute to lwIP . . . 94
1.4 CMake build system . . . 96
1.5 Common pitfalls . . . 98
1.6 Debugging memory pool sizes . . . 99
1.7 Reporting bugs . . . 99
1.8 Zero-copy RX . . . 99
1.9 System initialization . . . 100
1.10 Multithreading . . . 101
1.11 Optimization hints . . . 102
1.12 Deprecated List . . . 102

2 Topic Documentation **103**

2.1 lwIP . . . 103
2.1.1 Modules . . . 103
2.1.2 Detailed Description . . . 103
2.1.3 Mainloop mode ("NO_SYS") . . . 103
2.1.3.1 Functions . . . 103
2.1.3.2 Detailed Description . . . 103
2.1.3.3 Function Documentation . . . 106
2.1.3.3.1 ethernet_input() . . . 106
2.1.3.3.2 ip_input() . . . 106
2.1.3.3.3 lwip_init() . . . 106
2.1.3.3.4 netif_input() . . . 106
2.1.3.3.5 sys_check_timeouts() . . . 107
2.1.4 OS mode (TCPIP thread) . . . 107
2.1.4.1 Macros . . . 107
2.1.4.2 Functions . . . 107

2.1.4.3 Detailed Description 107
2.1.4.4 Macro Definition Documentation 107
2.1.4.4.1 tcpip_callback_with_block 107
2.1.4.5 Function Documentation 107
2.1.4.5.1 tcpip_callback() 107
2.1.4.5.2 tcpip_callbackmsg_delete() 108
2.1.4.5.3 tcpip_callbackmsg_new() 108
2.1.4.5.4 tcpip_callbackmsg_trycallback() 108
2.1.4.5.5 tcpip_callbackmsg_trycallback_fromisr() 108
2.1.4.5.6 tcpip_init() 109
2.1.4.5.7 tcpip_input() 109
2.1.4.5.8 tcpip_try_callback() 109
2.1.5 Porting (system abstraction layer) 109
2.1.5.1 Modules 109
2.1.5.2 Detailed Description 110
2.1.5.3 Non-standard functions 110
2.1.5.3.1 Functions 110
2.1.5.3.2 Detailed Description 110
2.1.5.3.3 Function Documentation 110
2.1.5.3.3.1 lwip_itoa() 110
2.1.5.3.3.2 lwip_memcmp_consttime() 110
2.1.5.3.3.3 lwip_stricmp() 110
2.1.5.3.3.4 lwip_strnicmp() 111
2.1.5.3.3.5 lwip_strnistr() 111
2.1.5.3.3.6 lwip_strnstr() 111
2.1.5.4 OS abstraction layer 111
2.1.5.4.1 Modules 111
2.1.5.4.2 Detailed Description 111
2.1.5.4.3 Semaphores 112
2.1.5.4.3.1 Functions 112
2.1.5.4.3.2 Detailed Description 112
2.1.5.4.3.3 Function Documentation 112
2.1.5.4.3.4 sys_arch_sem_wait() 112
2.1.5.4.3.5 sys_sem_free() 112
2.1.5.4.3.6 sys_sem_new() 113
2.1.5.4.3.7 sys_sem_set_invalid() 113
2.1.5.4.3.8 sys_sem_signal() 113
2.1.5.4.3.9 sys_sem_valid() 113
2.1.5.4.4 Mutexes 113

2.1.5.4.4.1 Functions . . . 113
2.1.5.4.4.2 Detailed Description . . . 114
2.1.5.4.4.3 Function Documentation . . . 114
2.1.5.4.4.4 sys_mutex_free() . . . 114
2.1.5.4.4.5 sys_mutex_lock() . . . 114
2.1.5.4.4.6 sys_mutex_new() . . . 114
2.1.5.4.4.7 sys_mutex_set_invalid() . . . 114
2.1.5.4.4.8 sys_mutex_unlock() . . . 114
2.1.5.4.4.9 sys_mutex_valid() . . . 115
2.1.5.4.5 Mailboxes . . . 115
2.1.5.4.5.1 Functions . . . 115
2.1.5.4.5.2 Detailed Description . . . 115
2.1.5.4.5.3 Function Documentation . . . 115
2.1.5.4.5.4 sys_arch_mbox_fetch() . . . 115
2.1.5.4.5.5 sys_arch_mbox_tryfetch() . . . 116
2.1.5.4.5.6 sys_mbox_free() . . . 116
2.1.5.4.5.7 sys_mbox_new() . . . 116
2.1.5.4.5.8 sys_mbox_post() . . . 116
2.1.5.4.5.9 sys_mbox_set_invalid() . . . 116
2.1.5.4.5.10 sys_mbox_trypost() . . . 117
2.1.5.4.5.11 sys_mbox_trypost_fromisr() . . . 117
2.1.5.4.5.12 sys_mbox_valid() . . . 117
2.1.5.4.6 Misc . . . 117
2.1.5.4.6.1 Functions . . . 117
2.1.5.4.6.2 Detailed Description . . . 117
2.1.5.4.6.3 Function Documentation . . . 117
2.1.5.4.6.4 sys_init() . . . 117
2.1.5.4.6.5 sys_msleep() . . . 118
2.1.5.4.6.6 sys_thread_new() . . . 118
2.1.5.5 Time . . . 118
2.1.5.5.1 Functions . . . 118
2.1.5.5.2 Detailed Description . . . 118
2.1.5.5.3 Function Documentation . . . 118
2.1.5.5.3.1 sys_now() . . . 118
2.1.5.6 Critical sections . . . 118
2.1.5.6.1 Macros . . . 118
2.1.5.6.2 Detailed Description . . . 119
2.1.5.6.3 Macro Definition Documentation . . . 119
2.1.5.6.3.1 SYS_ARCH_DECL_PROTECT . . . 119

2.1.5.6.3.2 SYS_ARCH_PROTECT 119
2.1.5.6.3.3 SYS_ARCH_UNPROTECT 119
2.1.5.7 Compiler/platform abstraction 119
2.1.5.7.1 Macros 119
2.1.5.7.2 Detailed Description 120
2.1.5.7.3 Macro Definition Documentation 120
2.1.5.7.3.1 BYTE_ORDER 120
2.1.5.7.3.2 LWIP_ALIGNMENT_CAST 120
2.1.5.7.3.3 LWIP_CONST_CAST 121
2.1.5.7.3.4 LWIP_DECLARE_MEMORY_ALIGNED 121
2.1.5.7.3.5 LWIP_MEM_ALIGN 121
2.1.5.7.3.6 LWIP_MEM_ALIGN_BUFFER 121
2.1.5.7.3.7 LWIP_MEM_ALIGN_SIZE 121
2.1.5.7.3.8 LWIP_NO_CTYPE_H 121
2.1.5.7.3.9 LWIP_NO_INTTYPES_H 121
2.1.5.7.3.10 LWIP_NO_LIMITS_H 121
2.1.5.7.3.11 LWIP_NO_STDDEF_H 122
2.1.5.7.3.12 LWIP_NO_STDINT_H 122
2.1.5.7.3.13 LWIP_PACKED_CAST 122
2.1.5.7.3.14 LWIP_PLATFORM_ASSERT 122
2.1.5.7.3.15 LWIP_PLATFORM_DIAG 122
2.1.5.7.3.16 LWIP_PROVIDE_ERRNO 122
2.1.5.7.3.17 LWIP_PTR_NUMERIC_CAST 122
2.1.5.7.3.18 LWIP_RAND 122
2.1.5.7.3.19 LWIP_UNUSED_ARG 123
2.1.5.7.3.20 PACK_STRUCT_BEGIN 123
2.1.5.7.3.21 PACK_STRUCT_END 123
2.1.5.7.3.22 PACK_STRUCT_FIELD 123
2.1.5.7.3.23 PACK_STRUCT_FLD_8 123
2.1.5.7.3.24 PACK_STRUCT_FLD_S 123
2.1.5.7.3.25 PACK_STRUCT_STRUCT 123
2.1.5.7.3.26 PACK_STRUCT_USE_INCLUDES 123
2.1.5.8 Performance measurement 124
2.1.6 Version 124
2.1.6.1 Macros 124
2.1.6.2 Detailed Description 124
2.1.6.3 Macro Definition Documentation 124
2.1.6.3.1 LWIP_RC_DEVELOPMENT 124
2.1.6.3.2 LWIP_RC_RELEASE 124

2.1.6.3.3 LWIP_VERSION 124
2.1.6.3.4 LWIP_VERSION_MAJOR 124
2.1.6.3.5 LWIP_VERSION_MINOR 125
2.1.6.3.6 LWIP_VERSION_RC 125
2.1.6.3.7 LWIP_VERSION_REVISION 125
2.1.6.3.8 LWIP_VERSION_STRING 125
2.1.7 Options (lwipopts.h) 125
2.1.7.1 Modules 125
2.1.7.2 Detailed Description 125
2.1.7.3 Debugging 125
2.1.7.3.1 Modules 125
2.1.7.3.2 Detailed Description 126
2.1.7.3.3 Assertion handling 126
2.1.7.3.3.1 Macros 126
2.1.7.3.3.2 Detailed Description 126
2.1.7.3.3.3 Macro Definition Documentation 126
2.1.7.3.3.4 LWIP_NOASSERT 126
2.1.7.3.4 Statistics 126
2.1.7.3.4.1 Macros 126
2.1.7.3.4.2 Detailed Description 127
2.1.7.3.4.3 Macro Definition Documentation 127
2.1.7.3.4.4 ETHARP_STATS 127
2.1.7.3.4.5 ICMP6_STATS 127
2.1.7.3.4.6 ICMP_STATS 127
2.1.7.3.4.7 IGMP_STATS 127
2.1.7.3.4.8 IP6_FRAG_STATS 127
2.1.7.3.4.9 IP6_STATS 127
2.1.7.3.4.10 IP_STATS 127
2.1.7.3.4.11 IPFRAG_STATS 127
2.1.7.3.4.12 LINK_STATS 127
2.1.7.3.4.13 LWIP_STATS 128
2.1.7.3.4.14 LWIP_STATS_DISPLAY 128
2.1.7.3.4.15 MEM_STATS 128
2.1.7.3.4.16 MEMP_STATS 128
2.1.7.3.4.17 MIB2_STATS 128
2.1.7.3.4.18 MLD6_STATS 128
2.1.7.3.4.19 ND6_STATS 128
2.1.7.3.4.20 SYS_STATS 128
2.1.7.3.4.21 TCP_STATS 128

2.1.7.3.4.22 UDP_STATS . . . 128
2.1.7.3.5 Debug messages . . . 129
2.1.7.3.5.1 Modules . . . 129
2.1.7.3.5.2 Macros . . . 129
2.1.7.3.5.3 Detailed Description . . . 130
2.1.7.3.5.4 Macro Definition Documentation . . . 130
2.1.7.3.5.5 ACD_DEBUG . . . 130
2.1.7.3.5.6 API_LIB_DEBUG . . . 130
2.1.7.3.5.7 API_MSG_DEBUG . . . 130
2.1.7.3.5.8 AUTOIP_DEBUG . . . 130
2.1.7.3.5.9 DHCP6_DEBUG . . . 130
2.1.7.3.5.10 DHCP_DEBUG . . . 130
2.1.7.3.5.11 DNS_DEBUG . . . 130
2.1.7.3.5.12 ETHARP_DEBUG . . . 131
2.1.7.3.5.13 ICMP_DEBUG . . . 131
2.1.7.3.5.14 IGMP_DEBUG . . . 131
2.1.7.3.5.15 INET_DEBUG . . . 131
2.1.7.3.5.16 IP6_DEBUG . . . 131
2.1.7.3.5.17 IP_DEBUG . . . 131
2.1.7.3.5.18 IP_REASS_DEBUG . . . 131
2.1.7.3.5.19 LWIP_DBG_MIN_LEVEL . . . 131
2.1.7.3.5.20 LWIP_DBG_TYPES_ON . . . 131
2.1.7.3.5.21 MEM_DEBUG . . . 132
2.1.7.3.5.22 MEMP_DEBUG . . . 132
2.1.7.3.5.23 NETIF_DEBUG . . . 132
2.1.7.3.5.24 PBUF_DEBUG . . . 132
2.1.7.3.5.25 RAW_DEBUG . . . 132
2.1.7.3.5.26 SLIP_DEBUG . . . 132
2.1.7.3.5.27 SOCKETS_DEBUG . . . 132
2.1.7.3.5.28 SYS_DEBUG . . . 132
2.1.7.3.5.29 TCP_CWND_DEBUG . . . 132
2.1.7.3.5.30 TCP_DEBUG . . . 132
2.1.7.3.5.31 TCP_FR_DEBUG . . . 133
2.1.7.3.5.32 TCP_INPUT_DEBUG . . . 133
2.1.7.3.5.33 TCP_OUTPUT_DEBUG . . . 133
2.1.7.3.5.34 TCP_QLEN_DEBUG . . . 133
2.1.7.3.5.35 TCP_RST_DEBUG . . . 133
2.1.7.3.5.36 TCP_RTO_DEBUG . . . 133
2.1.7.3.5.37 TCP_WND_DEBUG . . . 133

2.1.7.3.5.38 TCPIP_DEBUG . . . 133
2.1.7.3.5.39 TIMERS_DEBUG . . . 133
2.1.7.3.5.40 UDP_DEBUG . . . 133
2.1.7.3.5.41 LWIP_DBG_MIN_LEVEL and LWIP_DBG_TYPES_ON values . . . 134
2.1.7.3.5.42 Debug level (LWIP_DBG_MIN_LEVEL) . . . 134
2.1.7.3.5.43 Enable/disable debug messages completely (LWIP_DBG_TYPES_ON) . . . 134
2.1.7.3.5.44 Debug message types (LWIP_DBG_TYPES_ON) . . . 134
2.1.7.3.5.45 Detailed Description . . . 134
2.1.7.3.5.46 Macro Definition Documentation . . . 134
2.1.7.3.5.47 LWIP_DBG_FRESH . . . 134
2.1.7.3.5.48 LWIP_DBG_HALT . . . 134
2.1.7.3.5.49 LWIP_DBG_LEVEL_ALL . . . 134
2.1.7.3.5.50 LWIP_DBG_LEVEL_SERIOUS . . . 134
2.1.7.3.5.51 LWIP_DBG_LEVEL_SEVERE . . . 134
2.1.7.3.5.52 LWIP_DBG_LEVEL_WARNING . . . 135
2.1.7.3.5.53 LWIP_DBG_OFF . . . 135
2.1.7.3.5.54 LWIP_DBG_ON . . . 135
2.1.7.3.5.55 LWIP_DBG_STATE . . . 135
2.1.7.3.5.56 LWIP_DBG_TRACE . . . 135
2.1.7.3.6 Performance . . . 135
2.1.7.3.6.1 Macros . . . 135
2.1.7.3.6.2 Detailed Description . . . 135
2.1.7.3.6.3 Macro Definition Documentation . . . 135
2.1.7.3.6.4 LWIP_PERF . . . 135
2.1.7.4 Infrastructure . . . 135
2.1.7.4.1 Modules . . . 135
2.1.7.4.2 Detailed Description . . . 136
2.1.7.4.3 NO_SYS . . . 136
2.1.7.4.3.1 Macros . . . 136
2.1.7.4.3.2 Detailed Description . . . 136
2.1.7.4.3.3 Macro Definition Documentation . . . 136
2.1.7.4.3.4 NO_SYS . . . 136
2.1.7.4.4 Timers . . . 136
2.1.7.4.4.1 Macros . . . 136
2.1.7.4.4.2 Detailed Description . . . 136
2.1.7.4.4.3 Macro Definition Documentation . . . 136
2.1.7.4.4.4 LWIP_TIMERS . . . 136
2.1.7.4.4.5 LWIP_TIMERS_CUSTOM . . . 136
2.1.7.4.5 memcpy . . . 137

2.1.7.4.5.1 Macros 137
2.1.7.4.5.2 Detailed Description 137
2.1.7.4.5.3 Macro Definition Documentation 137
2.1.7.4.5.4 MEMCPY 137
2.1.7.4.5.5 MEMMOVE 137
2.1.7.4.5.6 SMEMCPY 137
2.1.7.4.6 Core locking and MPU 137
2.1.7.4.6.1 Macros 137
2.1.7.4.6.2 Detailed Description 137
2.1.7.4.6.3 Macro Definition Documentation 137
2.1.7.4.6.4 LWIP_ASSERT_CORE_LOCKED 137
2.1.7.4.6.5 LWIP_MARK_TCPIP_THREAD 138
2.1.7.4.6.6 LWIP_MPU_COMPATIBLE 138
2.1.7.4.6.7 LWIP_TCPIP_CORE_LOCKING 138
2.1.7.4.6.8 LWIP_TCPIP_CORE_LOCKING_INPUT 138
2.1.7.4.6.9 SYS_LIGHTWEIGHT_PROT 138
2.1.7.4.7 Heap and memory pools 138
2.1.7.4.7.1 Macros 138
2.1.7.4.7.2 Detailed Description 139
2.1.7.4.7.3 Macro Definition Documentation 139
2.1.7.4.7.4 LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT 139
2.1.7.4.7.5 MEM_ALIGNMENT 139
2.1.7.4.7.6 MEM_CUSTOM_ALLOCATOR 139
2.1.7.4.7.7 MEM_LIBC_MALLOC 139
2.1.7.4.7.8 MEM_OVERFLOW_CHECK 140
2.1.7.4.7.9 MEM_SANITY_CHECK 140
2.1.7.4.7.10 MEM_SIZE 140
2.1.7.4.7.11 MEM_USE_POOLS 140
2.1.7.4.7.12 MEM_USE_POOLS_TRY_BIGGER_POOL 140
2.1.7.4.7.13 MEMP_MEM_INIT 140
2.1.7.4.7.14 MEMP_MEM_MALLOC 140
2.1.7.4.7.15 MEMP_OVERFLOW_CHECK 141
2.1.7.4.7.16 MEMP_SANITY_CHECK 141
2.1.7.4.7.17 MEMP_USE_CUSTOM_POOLS 141
2.1.7.4.8 Internal memory pools 141
2.1.7.4.8.1 Macros 141
2.1.7.4.8.2 Detailed Description 142
2.1.7.4.8.3 Macro Definition Documentation 142
2.1.7.4.8.4 LWIP_NUM_SYS_TIMEOUT_INTERNAL 142

2.1.7.4.8.5 MEMP_NUM_ALTCP_PCB . . . 142
2.1.7.4.8.6 MEMP_NUM_API_MSG . . . 142
2.1.7.4.8.7 MEMP_NUM_ARP_QUEUE . . . 142
2.1.7.4.8.8 MEMP_NUM_DNS_API_MSG . . . 142
2.1.7.4.8.9 MEMP_NUM_FRAG_PBUF . . . 143
2.1.7.4.8.10 MEMP_NUM_IGMP_GROUP . . . 143
2.1.7.4.8.11 MEMP_NUM_LOCALHOSTLIST . . . 143
2.1.7.4.8.12 MEMP_NUM_NETBUF . . . 143
2.1.7.4.8.13 MEMP_NUM_NETCONN . . . 143
2.1.7.4.8.14 MEMP_NUM_NETDB . . . 143
2.1.7.4.8.15 MEMP_NUM_NETIFAPI_MSG . . . 143
2.1.7.4.8.16 MEMP_NUM_PBUF . . . 143
2.1.7.4.8.17 MEMP_NUM_RAW_PCB . . . 143
2.1.7.4.8.18 MEMP_NUM_REASSDATA . . . 144
2.1.7.4.8.19 MEMP_NUM_SELECT_CB . . . 144
2.1.7.4.8.20 MEMP_NUM_SOCKET_SETGETSOCKOPT_DATA . . . 144
2.1.7.4.8.21 MEMP_NUM_SYS_TIMEOUT . . . 144
2.1.7.4.8.22 MEMP_NUM_TCP_PCB . . . 144
2.1.7.4.8.23 MEMP_NUM_TCP_PCB_LISTEN . . . 144
2.1.7.4.8.24 MEMP_NUM_TCP_SEG . . . 144
2.1.7.4.8.25 MEMP_NUM_TCPIP_MSG_API . . . 144
2.1.7.4.8.26 MEMP_NUM_TCPIP_MSG_INPKT . . . 144
2.1.7.4.8.27 MEMP_NUM_UDP_PCB . . . 145
2.1.7.4.8.28 PBUF_POOL_SIZE . . . 145
2.1.7.4.9 SNMP MIB2 callbacks . . . 145
2.1.7.4.9.1 Macros . . . 145
2.1.7.4.9.2 Detailed Description . . . 145
2.1.7.4.9.3 Macro Definition Documentation . . . 145
2.1.7.4.9.4 LWIP_MIB2_CALLBACKS . . . 145
2.1.7.4.10 Multicast . . . 145
2.1.7.4.10.1 Macros . . . 145
2.1.7.4.10.2 Detailed Description . . . 145
2.1.7.4.10.3 Macro Definition Documentation . . . 145
2.1.7.4.10.4 LWIP_MULTICAST_TX_OPTIONS . . . 145
2.1.7.4.11 Threading . . . 145
2.1.7.4.11.1 Macros . . . 145
2.1.7.4.11.2 Detailed Description . . . 146
2.1.7.4.11.3 Macro Definition Documentation . . . 146
2.1.7.4.11.4 DEFAULT_ACCEPTMBOX_SIZE . . . 146

2.1.7.4.11.5 DEFAULT_RAW_RECVMBOX_SIZE 146
2.1.7.4.11.6 DEFAULT_TCP_RECVMBOX_SIZE 146
2.1.7.4.11.7 DEFAULT_THREAD_NAME 146
2.1.7.4.11.8 DEFAULT_THREAD_PRIO 146
2.1.7.4.11.9 DEFAULT_THREAD_STACKSIZE 147
2.1.7.4.11.10 DEFAULT_UDP_RECVMBOX_SIZE 147
2.1.7.4.11.11 LWIP_TCPIP_THREAD_ALIVE 147
2.1.7.4.11.12 SLIPIF_THREAD_NAME 147
2.1.7.4.11.13 SLIPIF_THREAD_PRIO 147
2.1.7.4.11.14 SLIPIF_THREAD_STACKSIZE 147
2.1.7.4.11.15 TCPIP_MBOX_SIZE 147
2.1.7.4.11.16 TCPIP_THREAD_NAME 147
2.1.7.4.11.17 TCPIP_THREAD_PRIO 147
2.1.7.4.11.18 TCPIP_THREAD_STACKSIZE 148
2.1.7.4.12 Checksum 148
2.1.7.4.12.1 Macros 148
2.1.7.4.12.2 Detailed Description 148
2.1.7.4.12.3 Macro Definition Documentation 148
2.1.7.4.12.4 CHECKSUM_CHECK_ICMP 148
2.1.7.4.12.5 CHECKSUM_CHECK_ICMP6 148
2.1.7.4.12.6 CHECKSUM_CHECK_IP 148
2.1.7.4.12.7 CHECKSUM_CHECK_TCP 148
2.1.7.4.12.8 CHECKSUM_CHECK_UDP 149
2.1.7.4.12.9 CHECKSUM_GEN_ICMP 149
2.1.7.4.12.10 CHECKSUM_GEN_ICMP6 149
2.1.7.4.12.11 CHECKSUM_GEN_IP 149
2.1.7.4.12.12 CHECKSUM_GEN_TCP 149
2.1.7.4.12.13 CHECKSUM_GEN_UDP 149
2.1.7.4.12.14 LWIP_CHECKSUM_CTRL_PER_NETIF 149
2.1.7.4.12.15 LWIP_CHECKSUM_ON_COPY 149
2.1.7.4.13 Hooks 149
2.1.7.4.13.1 Macros 149
2.1.7.4.13.2 Detailed Description 150
2.1.7.4.13.3 Macro Definition Documentation 150
2.1.7.4.13.4 LWIP_HOOK_DHCP6_APPEND_OPTIONS 150
2.1.7.4.13.5 LWIP_HOOK_DHCP_APPEND_OPTIONS 151
2.1.7.4.13.6 LWIP_HOOK_DHCP_PARSE_OPTION 151
2.1.7.4.13.7 LWIP_HOOK_ETHARP_GET_GW 152
2.1.7.4.13.8 LWIP_HOOK_FILENAME 152

2.1.7.4.13.9 LWIP_HOOK_IP4_CANFORWARD 152
2.1.7.4.13.10 LWIP_HOOK_IP4_INPUT 153
2.1.7.4.13.11 LWIP_HOOK_IP4_ROUTE 153
2.1.7.4.13.12 LWIP_HOOK_IP4_ROUTE_SRC 153
2.1.7.4.13.13 LWIP_HOOK_IP6_INPUT 154
2.1.7.4.13.14 LWIP_HOOK_IP6_ROUTE 154
2.1.7.4.13.15 LWIP_HOOK_MEMP_AVAILABLE 154
2.1.7.4.13.16 LWIP_HOOK_ND6_GET_GW 154
2.1.7.4.13.17 LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE 155
2.1.7.4.13.18 LWIP_HOOK_SOCKETS_GETSOCKOPT 155
2.1.7.4.13.19 LWIP_HOOK_SOCKETS_SETSOCKOPT 156
2.1.7.4.13.20 LWIP_HOOK_TCP_INPACKET_PCB 156
2.1.7.4.13.21 LWIP_HOOK_TCP_ISN 157
2.1.7.4.13.22 LWIP_HOOK_TCP_OUT_ADD_TCPOPTS 157
2.1.7.4.13.23 LWIP_HOOK_TCP_OUT_TCPOPT_LENGTH 157
2.1.7.4.13.24 LWIP_HOOK_UNKNOWN_ETH_PROTOCOL 158
2.1.7.4.13.25 LWIP_HOOK_VLAN_CHECK 158
2.1.7.4.13.26 LWIP_HOOK_VLAN_SET 159
2.1.7.5 Callback-style APIs 159
2.1.7.5.1 Modules 159
2.1.7.5.2 Detailed Description 159
2.1.7.5.3 RAW 159
2.1.7.5.3.1 Macros 159
2.1.7.5.3.2 Detailed Description 159
2.1.7.5.3.3 Macro Definition Documentation 159
2.1.7.5.3.4 LWIP_RAW 159
2.1.7.5.3.5 RAW_TTL 160
2.1.7.5.4 DNS 160
2.1.7.5.4.1 Macros 160
2.1.7.5.4.2 Detailed Description 160
2.1.7.5.4.3 Macro Definition Documentation 160
2.1.7.5.4.4 DNS_DOES_NAME_CHECK 160
2.1.7.5.4.5 DNS_LOCAL_HOSTLIST 160
2.1.7.5.4.6 DNS_LOCAL_HOSTLIST_IS_DYNAMIC 160
2.1.7.5.4.7 DNS_MAX_NAME_LENGTH 161
2.1.7.5.4.8 DNS_MAX_RETRIES 161
2.1.7.5.4.9 DNS_MAX_SERVERS 161
2.1.7.5.4.10 DNS_TABLE_SIZE 161
2.1.7.5.4.11 LWIP_DNS 161

2.1.7.5.4.12 LWIP_DNS_SECURE . . . 161
2.1.7.5.4.13 LWIP_DNS_SUPPORT_MDNS_QUERIES . . . 161
2.1.7.5.5 UDP . . . 161
2.1.7.5.5.1 Macros . . . 161
2.1.7.5.5.2 Detailed Description . . . 162
2.1.7.5.5.3 Macro Definition Documentation . . . 162
2.1.7.5.5.4 LWIP_NETBUF_RECVINFO . . . 162
2.1.7.5.5.5 LWIP_UDP . . . 162
2.1.7.5.5.6 LWIP_UDPLITE . . . 162
2.1.7.5.5.7 UDP_TTL . . . 162
2.1.7.5.6 TCP . . . 162
2.1.7.5.6.1 Macros . . . 162
2.1.7.5.6.2 Detailed Description . . . 163
2.1.7.5.6.3 Macro Definition Documentation . . . 163
2.1.7.5.6.4 LWIP_ALTCP . . . 163
2.1.7.5.6.5 LWIP_ALTCP_TLS . . . 163
2.1.7.5.6.6 LWIP_EVENT_API . . . 163
2.1.7.5.6.7 LWIP_TCP . . . 163
2.1.7.5.6.8 LWIP_TCP_MAX_SACK_NUM . . . 164
2.1.7.5.6.9 LWIP_TCP_PCB_NUM_EXT_ARGS . . . 164
2.1.7.5.6.10 LWIP_TCP_RTO_TIME . . . 164
2.1.7.5.6.11 LWIP_TCP_SACK_OUT . . . 164
2.1.7.5.6.12 LWIP_TCP_TIMESTAMPS . . . 164
2.1.7.5.6.13 LWIP_WND_SCALE . . . 164
2.1.7.5.6.14 TCP_CALCULATE_EFF_SEND_MSS . . . 164
2.1.7.5.6.15 TCP_DEFAULT_LISTEN_BACKLOG . . . 165
2.1.7.5.6.16 TCP_LISTEN_BACKLOG . . . 165
2.1.7.5.6.17 TCP_MAXRTX . . . 165
2.1.7.5.6.18 TCP_MSS . . . 165
2.1.7.5.6.19 TCP_OOSEQ_MAX_BYTES . . . 165
2.1.7.5.6.20 TCP_OOSEQ_MAX_PBUFS . . . 165
2.1.7.5.6.21 TCP_OVERSIZE . . . 165
2.1.7.5.6.22 TCP_QUEUE_OOSEQ . . . 166
2.1.7.5.6.23 TCP_SND_BUF . . . 166
2.1.7.5.6.24 TCP_SND_QUEUELEN . . . 166
2.1.7.5.6.25 TCP_SNDLOWAT . . . 166
2.1.7.5.6.26 TCP_SNDQUEUELOWAT . . . 166
2.1.7.5.6.27 TCP_SYNMAXRTX . . . 166
2.1.7.5.6.28 TCP_TTL . . . 166

2.1.7.5.6.29 TCP_WND 166
2.1.7.5.6.30 TCP_WND_UPDATE_THRESHOLD 166
2.1.7.6 Thread-safe APIs 167
2.1.7.6.1 Modules 167
2.1.7.6.2 Detailed Description 167
2.1.7.6.3 Netconn 167
2.1.7.6.3.1 Macros 167
2.1.7.6.3.2 Detailed Description 167
2.1.7.6.3.3 Macro Definition Documentation 167
2.1.7.6.3.4 LWIP_NETCONN 167
2.1.7.6.3.5 LWIP_NETCONN_FULLDUPLEX 167
2.1.7.6.3.6 LWIP_NETCONN_SEM_PER_THREAD 167
2.1.7.6.3.7 LWIP_TCPIP_TIMEOUT 167
2.1.7.6.4 Sockets 168
2.1.7.6.4.1 Macros 168
2.1.7.6.4.2 Detailed Description 168
2.1.7.6.4.3 Macro Definition Documentation 168
2.1.7.6.4.4 LWIP_COMPAT_SOCKETS 168
2.1.7.6.4.5 LWIP_FIONREAD_LINUXMODE 168
2.1.7.6.4.6 LWIP_POSIX_SOCKETS_IO_NAMES 169
2.1.7.6.4.7 LWIP_SO_LINGER 169
2.1.7.6.4.8 LWIP_SO_RCVBUF 169
2.1.7.6.4.9 LWIP_SO_RCVTIMEO 169
2.1.7.6.4.10 LWIP_SO_SNDRCVTIMEO_NONSTANDARD 169
2.1.7.6.4.11 LWIP_SO_SNDTIMEO 169
2.1.7.6.4.12 LWIP_SOCKET 169
2.1.7.6.4.13 LWIP_SOCKET_EXTERNAL_HEADERS 169
2.1.7.6.4.14 LWIP_SOCKET_OFFSET 169
2.1.7.6.4.15 LWIP_SOCKET_POLL 170
2.1.7.6.4.16 LWIP_SOCKET_SELECT 170
2.1.7.6.4.17 LWIP_TCP_CLOSE_TIMEOUT_MS_DEFAULT 170
2.1.7.6.4.18 LWIP_TCP_KEEPALIVE 170
2.1.7.6.4.19 RECV_BUFSIZE_DEFAULT 170
2.1.7.6.4.20 SO_REUSE 170
2.1.7.6.4.21 SO_REUSE_RXTOALL 170
2.1.7.7 IPv4 170
2.1.7.7.1 Modules 170
2.1.7.7.2 Macros 171
2.1.7.7.3 Detailed Description 171

2.1.7.7.4 Macro Definition Documentation 171
2.1.7.7.4.1 IP_DEFAULT_TTL 171
2.1.7.7.4.2 IP_FORWARD 171
2.1.7.7.4.3 IP_FORWARD_ALLOW_TX_ON_RX_NETIF 171
2.1.7.7.4.4 IP_FRAG 171
2.1.7.7.4.5 IP_OPTIONS_ALLOWED 171
2.1.7.7.4.6 IP_REASS_MAX_PBUFS 172
2.1.7.7.4.7 IP_REASS_MAXAGE 172
2.1.7.7.4.8 IP_REASSEMBLY 172
2.1.7.7.4.9 IP_SOF_BROADCAST 172
2.1.7.7.4.10 IP_SOF_BROADCAST_RECV 172
2.1.7.7.4.11 LWIP_IPV4 172
2.1.7.7.5 ARP 172
2.1.7.7.5.1 Macros 172
2.1.7.7.5.2 Detailed Description 173
2.1.7.7.5.3 Macro Definition Documentation 173
2.1.7.7.5.4 ARP_MAXAGE 173
2.1.7.7.5.5 ARP_QUEUE_LEN 173
2.1.7.7.5.6 ARP_QUEUEING 173
2.1.7.7.5.7 ARP_TABLE_SIZE 173
2.1.7.7.5.8 ETH_PAD_SIZE 173
2.1.7.7.5.9 ETHARP_SUPPORT_STATIC_ENTRIES 173
2.1.7.7.5.10 ETHARP_SUPPORT_VLAN 173
2.1.7.7.5.11 ETHARP_TABLE_MATCH_NETIF 174
2.1.7.7.5.12 LWIP_ARP 174
2.1.7.7.5.13 LWIP_ETHERNET 174
2.1.7.7.6 ICMP 174
2.1.7.7.6.1 Macros 174
2.1.7.7.6.2 Detailed Description 174
2.1.7.7.6.3 Macro Definition Documentation 174
2.1.7.7.6.4 ICMP_TTL 174
2.1.7.7.6.5 LWIP_BROADCAST_PING 174
2.1.7.7.6.6 LWIP_ICMP 175
2.1.7.7.6.7 LWIP_MULTICAST_PING 175
2.1.7.7.7 DHCP 175
2.1.7.7.7.1 Macros 175
2.1.7.7.7.2 Detailed Description 175
2.1.7.7.7.3 Macro Definition Documentation 175
2.1.7.7.7.4 LWIP_DHCP 175

2.1.7.7.7.5 LWIP_DHCP_BOOTP_FILE 175
2.1.7.7.7.6 LWIP_DHCP_DISCOVER_ADD_HOSTNAME 175
2.1.7.7.7.7 LWIP_DHCP_DOES_ACD_CHECK 175
2.1.7.7.7.8 LWIP_DHCP_GET_NTP_SRV 176
2.1.7.7.7.9 LWIP_DHCP_MAX_DNS_SERVERS 176
2.1.7.7.7.10 LWIP_DHCP_MAX_NTP_SERVERS 176
2.1.7.7.8 AUTOIP 176
2.1.7.7.8.1 Macros 176
2.1.7.7.8.2 Detailed Description 176
2.1.7.7.8.3 Macro Definition Documentation 176
2.1.7.7.8.4 LWIP_AUTOIP 176
2.1.7.7.8.5 LWIP_DHCP_AUTOIP_COOP 176
2.1.7.7.8.6 LWIP_DHCP_AUTOIP_COOP_TRIES 176
2.1.7.7.9 ACD 176
2.1.7.7.9.1 Macros 176
2.1.7.7.9.2 Detailed Description 177
2.1.7.7.9.3 Macro Definition Documentation 177
2.1.7.7.9.4 LWIP_ACD 177
2.1.7.7.10 IGMP 177
2.1.7.7.10.1 Macros 177
2.1.7.7.10.2 Detailed Description 177
2.1.7.7.10.3 Macro Definition Documentation 177
2.1.7.7.10.4 LWIP_IGMP 177
2.1.7.8 PBUF 177
2.1.7.8.1 Macros 177
2.1.7.8.2 Detailed Description 177
2.1.7.8.3 Macro Definition Documentation 177
2.1.7.8.3.1 LWIP_PBUF_CUSTOM_DATA 177
2.1.7.8.3.2 LWIP_PBUF_CUSTOM_DATA_INIT 177
2.1.7.8.3.3 LWIP_PBUF_REF_T 178
2.1.7.8.3.4 PBUF_LINK_ENCAPSULATION_HLEN 178
2.1.7.8.3.5 PBUF_LINK_HLEN 178
2.1.7.8.3.6 PBUF_POOL_BUFSIZE 178
2.1.7.9 NETIF 178
2.1.7.9.1 Modules 178
2.1.7.9.2 Macros 178
2.1.7.9.3 Detailed Description 179
2.1.7.9.4 Macro Definition Documentation 179
2.1.7.9.4.1 LWIP_NETIF_API 179

2.1.7.9.4.2 LWIP_NETIF_EXT_STATUS_CALLBACK . . . 179
2.1.7.9.4.3 LWIP_NETIF_HOSTNAME . . . 179
2.1.7.9.4.4 LWIP_NETIF_HWADDRHINT . . . 179
2.1.7.9.4.5 LWIP_NETIF_LINK_CALLBACK . . . 179
2.1.7.9.4.6 LWIP_NETIF_REMOVE_CALLBACK . . . 179
2.1.7.9.4.7 LWIP_NETIF_STATUS_CALLBACK . . . 179
2.1.7.9.4.8 LWIP_NETIF_TX_SINGLE_PBUF . . . 179
2.1.7.9.4.9 LWIP_NUM_NETIF_CLIENT_DATA . . . 180
2.1.7.9.4.10 LWIP_SINGLE_NETIF . . . 180
2.1.7.9.5 Loopback interface . . . 180
2.1.7.9.5.1 Macros . . . 180
2.1.7.9.5.2 Detailed Description . . . 180
2.1.7.9.5.3 Macro Definition Documentation . . . 180
2.1.7.9.5.4 LWIP_HAVE_LOOPIF . . . 180
2.1.7.9.5.5 LWIP_LOOPBACK_MAX_PBUFS . . . 180
2.1.7.9.5.6 LWIP_LOOPIF_MULTICAST . . . 180
2.1.7.9.5.7 LWIP_NETIF_LOOPBACK . . . 180
2.1.7.9.5.8 LWIP_NETIF_LOOPBACK_MULTITHREADING . . . 181
2.1.7.10 IPv6 . . . 181
2.1.7.10.1 Modules . . . 181
2.1.7.10.2 Macros . . . 181
2.1.7.10.3 Detailed Description . . . 181
2.1.7.10.4 Macro Definition Documentation . . . 181
2.1.7.10.4.1 IPV6_REASS_MAXAGE . . . 181
2.1.7.10.4.2 LWIP_IPV6 . . . 182
2.1.7.10.4.3 LWIP_IPV6_ADDRESS_LIFETIMES . . . 182
2.1.7.10.4.4 LWIP_IPV6_AUTOCONFIG . . . 182
2.1.7.10.4.5 LWIP_IPV6_DUP_DETECT_ATTEMPTS . . . 182
2.1.7.10.4.6 LWIP_IPV6_FORWARD . . . 182
2.1.7.10.4.7 LWIP_IPV6_FRAG . . . 182
2.1.7.10.4.8 LWIP_IPV6_NUM_ADDRESSES . . . 182
2.1.7.10.4.9 LWIP_IPV6_REASS . . . 182
2.1.7.10.4.10 LWIP_IPV6_SCOPES . . . 183
2.1.7.10.4.11 LWIP_IPV6_SCOPES_DEBUG . . . 183
2.1.7.10.4.12 LWIP_IPV6_SEND_ROUTER_SOLICIT . . . 183
2.1.7.10.5 ICMP6 . . . 183
2.1.7.10.5.1 Macros . . . 183
2.1.7.10.5.2 Detailed Description . . . 183
2.1.7.10.5.3 Macro Definition Documentation . . . 183

2.1.7.10.5.4 LWIP_ICMP6 . . . 183
2.1.7.10.5.5 LWIP_ICMP6_DATASIZE . . . 183
2.1.7.10.5.6 LWIP_ICMP6_HL . . . 184
2.1.7.10.6 Multicast listener discovery . . . 184
2.1.7.10.6.1 Macros . . . 184
2.1.7.10.6.2 Detailed Description . . . 184
2.1.7.10.6.3 Macro Definition Documentation . . . 184
2.1.7.10.6.4 LWIP_IPV6_MLD . . . 184
2.1.7.10.6.5 MEMP_NUM_MLD6_GROUP . . . 184
2.1.7.10.7 Neighbor discovery . . . 184
2.1.7.10.7.1 Macros . . . 184
2.1.7.10.7.2 Detailed Description . . . 185
2.1.7.10.7.3 Macro Definition Documentation . . . 185
2.1.7.10.7.4 LWIP_ND6_ALLOW_RA_UPDATES . . . 185
2.1.7.10.7.5 LWIP_ND6_DELAY_FIRST_PROBE_TIME . . . 185
2.1.7.10.7.6 LWIP_ND6_MAX_ANYCAST_DELAY_TIME . . . 185
2.1.7.10.7.7 LWIP_ND6_MAX_MULTICAST_SOLICIT . . . 185
2.1.7.10.7.8 LWIP_ND6_MAX_NEIGHBOR_ADVERTISEMENT . . . 185
2.1.7.10.7.9 LWIP_ND6_MAX_UNICAST_SOLICIT . . . 185
2.1.7.10.7.10 LWIP_ND6_NUM_DESTINATIONS . . . 185
2.1.7.10.7.11 LWIP_ND6_NUM_NEIGHBORS . . . 185
2.1.7.10.7.12 LWIP_ND6_NUM_PREFIXES . . . 186
2.1.7.10.7.13 LWIP_ND6_NUM_ROUTERS . . . 186
2.1.7.10.7.14 LWIP_ND6_QUEUEING . . . 186
2.1.7.10.7.15 LWIP_ND6_RDNSS_MAX_DNS_SERVERS . . . 186
2.1.7.10.7.16 LWIP_ND6_REACHABLE_TIME . . . 186
2.1.7.10.7.17 LWIP_ND6_RETRANS_TIMER . . . 186
2.1.7.10.7.18 LWIP_ND6_TCP_REACHABILITY_HINTS . . . 186
2.1.7.10.7.19 MEMP_NUM_ND6_QUEUE . . . 186
2.1.7.10.8 DHCPv6 . . . 187
2.1.7.10.8.1 Macros . . . 187
2.1.7.10.8.2 Detailed Description . . . 187
2.1.7.10.8.3 Macro Definition Documentation . . . 187
2.1.7.10.8.4 LWIP_DHCP6_GET_NTP_SRV . . . 187
2.1.7.10.8.5 LWIP_DHCP6_MAX_DNS_SERVERS . . . 187
2.1.7.10.8.6 LWIP_DHCP6_MAX_NTP_SERVERS . . . 187
2.1.7.10.8.7 LWIP_IPV6_DHCP6 . . . 187
2.1.7.10.8.8 LWIP_IPV6_DHCP6_STATEFUL . . . 187
2.1.7.10.8.9 LWIP_IPV6_DHCP6_STATELESS . . . 187

2.2 Infrastructure . . . 188
2.2.1 Modules . . . 188
2.2.2 Detailed Description . . . 188
2.2.3 IP address handling . . . 188
2.2.3.1 Modules . . . 188
2.2.3.2 Data Structures . . . 188
2.2.3.3 Macros . . . 188
2.2.3.4 Typedefs . . . 189
2.2.3.5 Enumerations . . . 189
2.2.3.6 Functions . . . 189
2.2.3.7 Detailed Description . . . 189
2.2.3.8 Macro Definition Documentation . . . 189
2.2.3.8.1 ip_addr_cmp . . . 189
2.2.3.8.2 ip_addr_cmp_zoneless . . . 189
2.2.3.8.3 ip_addr_eq . . . 189
2.2.3.8.4 ip_addr_isany . . . 189
2.2.3.8.5 ip_addr_isany_val . . . 190
2.2.3.8.6 ip_addr_isbroadcast . . . 190
2.2.3.8.7 ip_addr_islinklocal . . . 190
2.2.3.8.8 ip_addr_isloopback . . . 190
2.2.3.8.9 ip_addr_ismulticast . . . 190
2.2.3.8.10 ip_addr_net_eq . . . 190
2.2.3.8.11 ip_addr_netcmp . . . 191
2.2.3.8.12 ip_addr_zoneless_eq . . . 191
2.2.3.8.13 IP_ANY_TYPE . . . 191
2.2.3.9 Typedef Documentation . . . 191
2.2.3.9.1 ip_addr_t . . . 191
2.2.3.10 Enumeration Type Documentation . . . 191
2.2.3.10.1 lwip_ip_addr_type . . . 191
2.2.3.11 Function Documentation . . . 191
2.2.3.11.1 ipaddr_aton() . . . 191
2.2.3.11.2 ipaddr_ntoa() . . . 192
2.2.3.11.3 ipaddr_ntoa_r() . . . 192
2.2.3.12 IPv4 only . . . 192
2.2.3.12.1 Macros . . . 192
2.2.3.12.2 Detailed Description . . . 192
2.2.3.12.3 Macro Definition Documentation . . . 192
2.2.3.12.3.1 IP4_ADDR_ANY . . . 192
2.2.3.12.3.2 IP4_ADDR_ANY4 . . . 192

2.2.3.12.3.3 ip_2_ip4 . . . 192
2.2.3.12.3.4 IP_ADDR_ANY . . . 193
2.2.3.13 IPv6 only . . . 193
2.2.3.13.1 Macros . . . 193
2.2.3.13.2 Detailed Description . . . 193
2.2.3.13.3 Macro Definition Documentation . . . 193
2.2.3.13.3.1 IP6_ADDR_ANY . . . 193
2.2.3.13.3.2 IP6_ADDR_ANY6 . . . 193
2.2.3.13.3.3 ip_2_ip6 . . . 193
2.2.4 Memory pools . . . 193
2.2.4.1 Macros . . . 193
2.2.4.2 Detailed Description . . . 193
2.2.4.3 Macro Definition Documentation . . . 194
2.2.4.3.1 LWIP_MEMPOOL_ALLOC . . . 194
2.2.4.3.2 LWIP_MEMPOOL_DECLARE . . . 194
2.2.4.3.3 LWIP_MEMPOOL_FREE . . . 194
2.2.4.3.4 LWIP_MEMPOOL_INIT . . . 194
2.2.4.3.5 LWIP_MEMPOOL_PROTOTYPE . . . 194
2.2.5 Packet buffers (PBUF) . . . 195
2.2.5.1 Macros . . . 195
2.2.5.2 Enumerations . . . 195
2.2.5.3 Functions . . . 195
2.2.5.4 Detailed Description . . . 196
2.2.5.5 Macro Definition Documentation . . . 196
2.2.5.5.1 PBUF_NEEDS_COPY . . . 196
2.2.5.6 Enumeration Type Documentation . . . 196
2.2.5.6.1 pbuf_layer . . . 196
2.2.5.6.2 pbuf_type . . . 197
2.2.5.7 Function Documentation . . . 197
2.2.5.7.1 pbuf_alloc() . . . 197
2.2.5.7.2 pbuf_alloc_reference() . . . 198
2.2.5.7.3 pbuf_alloced_custom() . . . 198
2.2.5.7.4 pbuf_cat() . . . 199
2.2.5.7.5 pbuf_chain() . . . 199
2.2.5.7.6 pbuf_clone() . . . 199
2.2.5.7.7 pbuf_coalesce() . . . 200
2.2.5.7.8 pbuf_copy() . . . 200
2.2.5.7.9 pbuf_copy_partial() . . . 200
2.2.5.7.10 pbuf_copy_partial_pbuf() . . . 201

2.2.5.7.11 pbuf_free() . . . 201
2.2.5.7.12 pbuf_get_at() . . . 201
2.2.5.7.13 pbuf_get_contiguous() . . . 202
2.2.5.7.14 pbuf_memcmp() . . . 202
2.2.5.7.15 pbuf_memfind() . . . 202
2.2.5.7.16 pbuf_put_at() . . . 203
2.2.5.7.17 pbuf_realloc() . . . 203
2.2.5.7.18 pbuf_ref() . . . 203
2.2.5.7.19 pbuf_skip() . . . 203
2.2.5.7.20 pbuf_take() . . . 204
2.2.5.7.21 pbuf_take_at() . . . 204
2.2.5.7.22 pbuf_try_get_at() . . . 204
2.2.6 Error codes . . . 204
2.2.6.1 Typedefs . . . 204
2.2.6.2 Enumerations . . . 204
2.2.6.3 Detailed Description . . . 205
2.2.6.4 Typedef Documentation . . . 205
2.2.6.4.1 err_t . . . 205
2.2.6.5 Enumeration Type Documentation . . . 205
2.2.6.5.1 err_enum_t . . . 205
2.2.7 IANA assigned numbers . . . 205
2.2.7.1 Enumerations . . . 205
2.2.7.2 Detailed Description . . . 206
2.2.7.3 Enumeration Type Documentation . . . 206
2.2.7.3.1 lwip_iana_hwtype . . . 206
2.2.7.3.2 lwip_iana_port_number . . . 206
2.2.8 IEEE assigned numbers . . . 206
2.2.8.1 Enumerations . . . 206
2.2.8.2 Detailed Description . . . 206
2.2.8.3 Enumeration Type Documentation . . . 206
2.2.8.3.1 lwip_ieee_eth_type . . . 206
2.3 APIs . . . 207
2.3.1 Modules . . . 207
2.3.2 Detailed Description . . . 207
2.3.3 "raw" APIs . . . 208
2.3.3.1 Modules . . . 208
2.3.3.2 Detailed Description . . . 208
2.3.3.3 Application layered TCP Introduction . . . 208
2.3.3.3.1 Modules . . . 208

2.3.3.3.2 Detailed Description . . . 208
2.3.3.3.3 Application layered TCP Functions . . . 209
2.3.3.3.3.1 Modules . . . 209
2.3.3.3.3.2 Data Structures . . . 209
2.3.3.3.3.3 Typedefs . . . 210
2.3.3.3.3.4 Functions . . . 210
2.3.3.3.3.5 Detailed Description . . . 210
2.3.3.3.3.6 Typedef Documentation . . . 210
2.3.3.3.3.7 altcp_allocator_t . . . 210
2.3.3.3.3.8 Function Documentation . . . 211
2.3.3.3.3.9 altcp_abort() . . . 211
2.3.3.3.3.10 altcp_accept() . . . 211
2.3.3.3.3.11 altcp_arg() . . . 211
2.3.3.3.3.12 altcp_bind() . . . 211
2.3.3.3.3.13 altcp_close() . . . 211
2.3.3.3.3.14 altcp_connect() . . . 211
2.3.3.3.3.15 altcp_err() . . . 211
2.3.3.3.3.16 altcp_listen_with_backlog_and_err() . . . 211
2.3.3.3.3.17 altcp_mss() . . . 211
2.3.3.3.3.18 altcp_new() . . . 212
2.3.3.3.3.19 altcp_new_ip6() . . . 212
2.3.3.3.3.20 altcp_new_ip_type() . . . 212
2.3.3.3.3.21 altcp_output() . . . 212
2.3.3.3.3.22 altcp_poll() . . . 212
2.3.3.3.3.23 altcp_recv() . . . 212
2.3.3.3.3.24 altcp_recved() . . . 212
2.3.3.3.3.25 altcp_sent() . . . 212
2.3.3.3.3.26 altcp_setprio() . . . 213
2.3.3.3.3.27 altcp_shutdown() . . . 213
2.3.3.3.3.28 altcp_sndbuf() . . . 213
2.3.3.3.3.29 altcp_sndqueuelen() . . . 213
2.3.3.3.3.30 altcp_write() . . . 213
2.3.3.3.3.31 TLS layer . . . 213
2.3.3.3.3.32 Functions . . . 213
2.3.3.3.3.33 Detailed Description . . . 214
2.3.3.3.3.34 Function Documentation . . . 214
2.3.3.3.3.35 altcp_tls_alloc() . . . 214
2.3.3.3.3.36 altcp_tls_config_server_add_privkey_cert() . . . 214
2.3.3.3.3.37 altcp_tls_configure_alpn_protocols() . . . 214

2.3.3.3.3.38 altcp_tls_context() . . . 214
2.3.3.3.3.39 altcp_tls_create_config_client() . . . 214
2.3.3.3.3.40 altcp_tls_create_config_client_2wayauth() . . . 214
2.3.3.3.3.41 altcp_tls_create_config_server() . . . 215
2.3.3.3.3.42 altcp_tls_create_config_server_privkey_cert() . . . 215
2.3.3.3.3.43 altcp_tls_free_config() . . . 215
2.3.3.3.3.44 altcp_tls_free_entropy() . . . 215
2.3.3.3.3.45 altcp_tls_free_session() . . . 215
2.3.3.3.3.46 altcp_tls_get_session() . . . 215
2.3.3.3.3.47 altcp_tls_init_session() . . . 215
2.3.3.3.3.48 altcp_tls_new() . . . 215
2.3.3.3.3.49 altcp_tls_set_session() . . . 216
2.3.3.3.3.50 altcp_tls_wrap() . . . 216
2.3.3.4 DNS . . . 216
2.3.3.4.1 Functions . . . 216
2.3.3.4.2 Detailed Description . . . 216
2.3.3.4.3 Function Documentation . . . 216
2.3.3.4.3.1 dns_gethostbyname() . . . 216
2.3.3.4.3.2 dns_gethostbyname_addrtype() . . . 217
2.3.3.4.3.3 dns_getserver() . . . 217
2.3.3.4.3.4 dns_setserver() . . . 218
2.3.3.5 IP . . . 218
2.3.3.5.1 Modules . . . 218
2.3.3.5.2 Macros . . . 218
2.3.3.5.3 Detailed Description . . . 218
2.3.3.5.4 Macro Definition Documentation . . . 218
2.3.3.5.4.1 ip_netif_get_local_ip . . . 218
2.3.3.5.4.2 ip_output . . . 218
2.3.3.5.4.3 ip_output_if . . . 219
2.3.3.5.4.4 ip_output_if_src . . . 219
2.3.3.5.4.5 ip_route . . . 219
2.3.3.5.5 IPv4 . . . 219
2.3.3.5.5.1 Modules . . . 219
2.3.3.5.5.2 Functions . . . 219
2.3.3.5.5.3 Detailed Description . . . 219
2.3.3.5.5.4 Function Documentation . . . 219
2.3.3.5.5.5 ip4_set_default_multicast_netif() . . . 219
2.3.3.5.5.6 ACD . . . 220
2.3.3.5.5.7 Functions . . . 220

2.3.3.5.5.8 Detailed Description . . . 220
2.3.3.5.5.9 Function Documentation . . . 220
2.3.3.5.5.10 acd_add() . . . 220
2.3.3.5.5.11 acd_netif_ip_addr_changed() . . . 220
2.3.3.5.5.12 acd_network_changed_link_down() . . . 221
2.3.3.5.5.13 acd_remove() . . . 221
2.3.3.5.5.14 acd_start() . . . 221
2.3.3.5.5.15 acd_stop() . . . 221
2.3.3.5.5.16 AUTOIP . . . 221
2.3.3.5.5.17 Functions . . . 221
2.3.3.5.5.18 Detailed Description . . . 222
2.3.3.5.5.19 Function Documentation . . . 222
2.3.3.5.5.20 autoip_remove_struct() . . . 222
2.3.3.5.5.21 autoip_set_struct() . . . 222
2.3.3.5.5.22 autoip_start() . . . 222
2.3.3.5.5.23 autoip_stop() . . . 222
2.3.3.5.5.24 DHCPv4 . . . 223
2.3.3.5.5.25 Functions . . . 223
2.3.3.5.5.26 Detailed Description . . . 223
2.3.3.5.5.27 Function Documentation . . . 223
2.3.3.5.5.28 dhcp_cleanup() . . . 223
2.3.3.5.5.29 dhcp_inform() . . . 224
2.3.3.5.5.30 dhcp_release() . . . 224
2.3.3.5.5.31 dhcp_release_and_stop() . . . 224
2.3.3.5.5.32 dhcp_renew() . . . 224
2.3.3.5.5.33 dhcp_set_struct() . . . 224
2.3.3.5.5.34 dhcp_start() . . . 224
2.3.3.5.5.35 dhcp_stop() . . . 225
2.3.3.5.5.36 IGMP . . . 225
2.3.3.5.5.37 Macros . . . 225
2.3.3.5.5.38 Functions . . . 225
2.3.3.5.5.39 Detailed Description . . . 225
2.3.3.5.5.40 Macro Definition Documentation . . . 225
2.3.3.5.5.41 netif_igmp_data . . . 225
2.3.3.5.5.42 Function Documentation . . . 225
2.3.3.5.5.43 igmp_joingroup() . . . 225
2.3.3.5.5.44 igmp_joingroup_netif() . . . 226
2.3.3.5.5.45 igmp_leavegroup() . . . 226
2.3.3.5.5.46 igmp_leavegroup_netif() . . . 226

2.3.3.5.6 IPv6 226

2.3.3.5.6.1 Modules 226

2.3.3.5.6.2 Functions 226

2.3.3.5.6.3 Detailed Description 227

2.3.3.5.6.4 Function Documentation 227

2.3.3.5.6.5 ip6_select_source_address() 227

2.3.3.5.6.6 DHCPv6 227

2.3.3.5.6.7 Functions 227

2.3.3.5.6.8 Detailed Description 228

2.3.3.5.6.9 Function Documentation 228

2.3.3.5.6.10 dhcp6_cleanup() 228

2.3.3.5.6.11 dhcp6_disable() 228

2.3.3.5.6.12 dhcp6_enable_stateful() 228

2.3.3.5.6.13 dhcp6_enable_stateless() 228

2.3.3.5.6.14 dhcp6_set_struct() 229

2.3.3.5.6.15 MLD6 229

2.3.3.5.6.16 Macros 229

2.3.3.5.6.17 Functions 229

2.3.3.5.6.18 Detailed Description 229

2.3.3.5.6.19 Macro Definition Documentation 229

2.3.3.5.6.20 netif_mld6_data 229

2.3.3.5.6.21 Function Documentation 229

2.3.3.5.6.22 mld6_joingroup() 229

2.3.3.5.6.23 mld6_joingroup_netif() 230

2.3.3.5.6.24 mld6_leavegroup() 230

2.3.3.5.6.25 mld6_leavegroup_netif() 230

2.3.3.5.6.26 IPv6 Zones 230

2.3.3.5.6.27 Macros 230

2.3.3.5.6.28 Enumerations 231

2.3.3.5.6.29 Detailed Description 231

2.3.3.5.6.30 Macro Definition Documentation 231

2.3.3.5.6.31 ip6_addr_assign_zone 231

2.3.3.5.6.32 ip6_addr_clear_zone 231

2.3.3.5.6.33 ip6_addr_cmp_zone 232

2.3.3.5.6.34 ip6_addr_copy_zone 232

2.3.3.5.6.35 ip6_addr_equals_zone 232

2.3.3.5.6.36 ip6_addr_has_scope 232

2.3.3.5.6.37 ip6_addr_has_zone 232

2.3.3.5.6.38 ip6_addr_lacks_zone 232

2.3.3.5.6.39 ip6_addr_select_zone . . . 233
2.3.3.5.6.40 ip6_addr_set_zone . . . 233
2.3.3.5.6.41 ip6_addr_test_zone . . . 233
2.3.3.5.6.42 ip6_addr_zone . . . 233
2.3.3.5.6.43 ip6_addr_zone_eq . . . 234
2.3.3.5.6.44 IP6_NO_ZONE . . . 234
2.3.3.5.6.45 IPADDR6_ZONE_INIT . . . 234
2.3.3.5.6.46 IPV6_CUSTOM_SCOPES . . . 234
2.3.3.5.6.47 Enumeration Type Documentation . . . 234
2.3.3.5.6.48 lwip_ipv6_scope_type . . . 234
2.3.3.6 Network interface (NETIF) . . . 234
2.3.3.6.1 Modules . . . 234
2.3.3.6.2 Data Structures . . . 234
2.3.3.6.3 Macros . . . 235
2.3.3.6.4 Typedefs . . . 235
2.3.3.6.5 Functions . . . 235
2.3.3.6.6 Detailed Description . . . 235
2.3.3.6.7 Macro Definition Documentation . . . 235
2.3.3.6.7.1 netif_is_up . . . 235
2.3.3.6.7.2 netif_set_igmp_mac_filter . . . 236
2.3.3.6.7.3 netif_set_mld_mac_filter . . . 236
2.3.3.6.8 Typedef Documentation . . . 236
2.3.3.6.8.1 netif_ext_callback_fn . . . 236
2.3.3.6.8.2 netif_nsc_reason_t . . . 236
2.3.3.6.9 Function Documentation . . . 236
2.3.3.6.9.1 netif_add() . . . 236
2.3.3.6.9.2 netif_add_ext_callback() . . . 236
2.3.3.6.9.3 netif_add_noaddr() . . . 237
2.3.3.6.9.4 netif_find() . . . 237
2.3.3.6.9.5 netif_get_by_index() . . . 237
2.3.3.6.9.6 netif_index_to_name() . . . 237
2.3.3.6.9.7 netif_loop_output() . . . 238
2.3.3.6.9.8 netif_name_to_index() . . . 238
2.3.3.6.9.9 netif_remove() . . . 238
2.3.3.6.9.10 netif_remove_ext_callback() . . . 238
2.3.3.6.9.11 netif_set_default() . . . 238
2.3.3.6.9.12 netif_set_down() . . . 238
2.3.3.6.9.13 netif_set_link_callback() . . . 239
2.3.3.6.9.14 netif_set_link_down() . . . 239

2.3.3.6.9.15 netif_set_link_up() 239
2.3.3.6.9.16 netif_set_remove_callback() 239
2.3.3.6.9.17 netif_set_status_callback() 239
2.3.3.6.9.18 netif_set_up() 239
2.3.3.6.10 IPv4 address handling 239
2.3.3.6.10.1 Functions 239
2.3.3.6.10.2 Detailed Description 240
2.3.3.6.10.3 Function Documentation 240
2.3.3.6.10.4 netif_set_addr() 240
2.3.3.6.10.5 netif_set_gw() 240
2.3.3.6.10.6 netif_set_ipaddr() 240
2.3.3.6.10.7 netif_set_netmask() 241
2.3.3.6.11 IPv6 address handling 241
2.3.3.6.11.1 Functions 241
2.3.3.6.11.2 Detailed Description 241
2.3.3.6.11.3 Function Documentation 241
2.3.3.6.11.4 netif_add_ip6_address() 241
2.3.3.6.11.5 netif_create_ip6_linklocal_address() 241
2.3.3.6.11.6 netif_ip6_addr_set() 242
2.3.3.6.11.7 netif_ip6_addr_set_state() 242
2.3.3.6.12 Client data handling 242
2.3.3.6.12.1 Macros 242
2.3.3.6.12.2 Functions 242
2.3.3.6.12.3 Detailed Description 242
2.3.3.6.12.4 Macro Definition Documentation 243
2.3.3.6.12.5 netif_get_client_data 243
2.3.3.6.12.6 netif_set_client_data 243
2.3.3.6.12.7 Function Documentation 243
2.3.3.6.12.8 netif_alloc_client_data_id() 243
2.3.3.6.13 Flags 243
2.3.3.6.13.1 Macros 243
2.3.3.6.13.2 Detailed Description 243
2.3.3.6.13.3 Macro Definition Documentation 243
2.3.3.6.13.4 NETIF_FLAG_BROADCAST 243
2.3.3.6.13.5 NETIF_FLAG_ETHARP 243
2.3.3.6.13.6 NETIF_FLAG_ETHERNET 244
2.3.3.6.13.7 NETIF_FLAG_IGMP 244
2.3.3.6.13.8 NETIF_FLAG_LINK_UP 244
2.3.3.6.13.9 NETIF_FLAG_MLD6 244

2.3.3.6.13.10 NETIF_FLAG_UP . . . 244
2.3.3.6.14 MIB2 statistics . . . 244
2.3.3.6.14.1 Data Structures . . . 244
2.3.3.6.14.2 Macros . . . 244
2.3.3.6.14.3 Enumerations . . . 244
2.3.3.6.14.4 Detailed Description . . . 245
2.3.3.6.14.5 Macro Definition Documentation . . . 245
2.3.3.6.14.6 MIB2_INIT_NETIF . . . 245
2.3.3.6.14.7 MIB2_STATS_NETIF_ADD . . . 245
2.3.3.6.14.8 MIB2_STATS_NETIF_INC . . . 245
2.3.3.6.14.9 Enumeration Type Documentation . . . 245
2.3.3.6.14.10 snmp_ifType . . . 245
2.3.3.7 RAW . . . 245
2.3.3.7.1 Functions . . . 245
2.3.3.7.2 Detailed Description . . . 246
2.3.3.7.3 Function Documentation . . . 246
2.3.3.7.3.1 raw_bind() . . . 246
2.3.3.7.3.2 raw_bind_netif() . . . 246
2.3.3.7.3.3 raw_connect() . . . 247
2.3.3.7.3.4 raw_disconnect() . . . 247
2.3.3.7.3.5 raw_new() . . . 247
2.3.3.7.3.6 raw_new_ip_type() . . . 247
2.3.3.7.3.7 raw_recv() . . . 248
2.3.3.7.3.8 raw_remove() . . . 248
2.3.3.7.3.9 raw_send() . . . 248
2.3.3.7.3.10 raw_sendto() . . . 248
2.3.3.7.3.11 raw_sendto_if_src() . . . 249
2.3.3.8 TCP . . . 249
2.3.3.8.1 Modules . . . 249
2.3.3.8.2 Functions . . . 249
2.3.3.8.3 Detailed Description . . . 250
2.3.3.8.4 Function Documentation . . . 251
2.3.3.8.4.1 tcp_abort() . . . 251
2.3.3.8.4.2 tcp_accept() . . . 251
2.3.3.8.4.3 tcp_arg() . . . 251
2.3.3.8.4.4 tcp_backlog_accepted() . . . 252
2.3.3.8.4.5 tcp_backlog_delayed() . . . 252
2.3.3.8.4.6 tcp_bind() . . . 252
2.3.3.8.4.7 tcp_bind_netif() . . . 252

2.3.3.8.4.8 tcp_close() 253
2.3.3.8.4.9 tcp_connect() 253
2.3.3.8.4.10 tcp_err() 253
2.3.3.8.4.11 tcp_listen_with_backlog() 254
2.3.3.8.4.12 tcp_listen_with_backlog_and_err() 254
2.3.3.8.4.13 tcp_new() 255
2.3.3.8.4.14 tcp_new_ip_type() 255
2.3.3.8.4.15 tcp_output() 255
2.3.3.8.4.16 tcp_poll() 255
2.3.3.8.4.17 tcp_recv() 256
2.3.3.8.4.18 tcp_recved() 256
2.3.3.8.4.19 tcp_sent() 256
2.3.3.8.4.20 tcp_shutdown() 256
2.3.3.8.4.21 tcp_write() 257
2.3.3.8.5 ext arguments 257
2.3.3.8.5.1 Functions 257
2.3.3.8.5.2 Detailed Description 258
2.3.3.8.5.3 Function Documentation 258
2.3.3.8.5.4 tcp_ext_arg_alloc_id() 258
2.3.3.8.5.5 tcp_ext_arg_get() 258
2.3.3.8.5.6 tcp_ext_arg_set() 258
2.3.3.8.5.7 tcp_ext_arg_set_callbacks() 259
2.3.3.9 UDP 259
2.3.3.9.1 Functions 259
2.3.3.9.2 Detailed Description 259
2.3.3.9.3 Function Documentation 259
2.3.3.9.3.1 udp_bind() 259
2.3.3.9.3.2 udp_bind_netif() 260
2.3.3.9.3.3 udp_connect() 260
2.3.3.9.3.4 udp_disconnect() 260
2.3.3.9.3.5 udp_new() 260
2.3.3.9.3.6 udp_new_ip_type() 261
2.3.3.9.3.7 udp_recv() 261
2.3.3.9.3.8 udp_remove() 261
2.3.3.9.3.9 udp_send() 261
2.3.3.9.3.10 udp_sendto() 262
2.3.3.9.3.11 udp_sendto_if() 262
2.3.3.9.3.12 udp_sendto_if_src() 262
2.3.3.10 Ethernet 263

2.3.3.10.1 Functions . . . 263
2.3.3.10.2 Detailed Description . . . 263
2.3.3.10.3 Function Documentation . . . 263
2.3.3.10.3.1 ethernet_output() . . . 263
2.3.4 Sequential-style APIs . . . 263
2.3.4.1 Modules . . . 263
2.3.4.2 Detailed Description . . . 263
2.3.4.3 Netconn API . . . 264
2.3.4.3.1 Modules . . . 264
2.3.4.3.2 Detailed Description . . . 264
2.3.4.3.3 Common functions . . . 264
2.3.4.3.3.1 Macros . . . 264
2.3.4.3.3.2 Enumerations . . . 264
2.3.4.3.3.3 Functions . . . 264
2.3.4.3.3.4 Detailed Description . . . 264
2.3.4.3.3.5 Macro Definition Documentation . . . 264
2.3.4.3.3.6 netconn_get_ipv6only . . . 264
2.3.4.3.3.7 netconn_new . . . 265
2.3.4.3.3.8 netconn_set_ipv6only . . . 265
2.3.4.3.3.9 Enumeration Type Documentation . . . 265
2.3.4.3.3.10 netconn_type . . . 265
2.3.4.3.3.11 Function Documentation . . . 265
2.3.4.3.3.12 netconn_bind() . . . 265
2.3.4.3.3.13 netconn_bind_if() . . . 266
2.3.4.3.3.14 netconn_connect() . . . 266
2.3.4.3.3.15 netconn_delete() . . . 266
2.3.4.3.3.16 netconn_err() . . . 266
2.3.4.3.3.17 netconn_gethostbyname_addrtype() . . . 266
2.3.4.3.3.18 netconn_prepare_delete() . . . 267
2.3.4.3.3.19 netconn_recv() . . . 267
2.3.4.3.4 TCP only . . . 267
2.3.4.3.4.1 Functions . . . 267
2.3.4.3.4.2 Detailed Description . . . 267
2.3.4.3.4.3 Function Documentation . . . 267
2.3.4.3.4.4 netconn_accept() . . . 267
2.3.4.3.4.5 netconn_close() . . . 268
2.3.4.3.4.6 netconn_listen_with_backlog() . . . 268
2.3.4.3.4.7 netconn_recv_tcp_pbuf() . . . 268
2.3.4.3.4.8 netconn_recv_tcp_pbuf_flags() . . . 268

2.3.4.3.4.9 netconn_shutdown() 268
2.3.4.3.4.10 netconn_write_partly() 269
2.3.4.3.5 UDP only 269
2.3.4.3.5.1 Functions 269
2.3.4.3.5.2 Detailed Description 269
2.3.4.3.5.3 Function Documentation 269
2.3.4.3.5.4 netconn_disconnect() 269
2.3.4.3.5.5 netconn_join_leave_group() 269
2.3.4.3.5.6 netconn_join_leave_group_netif() 271
2.3.4.3.5.7 netconn_send() 271
2.3.4.3.5.8 netconn_sendto() 271
2.3.4.3.6 Network buffers 271
2.3.4.3.6.1 Functions 271
2.3.4.3.6.2 Detailed Description 272
2.3.4.3.6.3 Function Documentation 272
2.3.4.3.6.4 netbuf_alloc() 272
2.3.4.3.6.5 netbuf_chain() 272
2.3.4.3.6.6 netbuf_data() 272
2.3.4.3.6.7 netbuf_delete() 273
2.3.4.3.6.8 netbuf_first() 273
2.3.4.3.6.9 netbuf_free() 273
2.3.4.3.6.10 netbuf_new() 273
2.3.4.3.6.11 netbuf_next() 273
2.3.4.3.6.12 netbuf_ref() 273
2.3.4.4 NETIF API 274
2.3.4.4.1 Modules 274
2.3.4.4.2 Detailed Description 274
2.3.4.4.3 NETIF related 274
2.3.4.4.3.1 Macros 274
2.3.4.4.3.2 Functions 274
2.3.4.4.3.3 Detailed Description 274
2.3.4.4.3.4 Macro Definition Documentation 274
2.3.4.4.3.5 netifapi_netif_remove 274
2.3.4.4.3.6 netifapi_netif_set_default 275
2.3.4.4.3.7 netifapi_netif_set_down 275
2.3.4.4.3.8 netifapi_netif_set_link_down 275
2.3.4.4.3.9 netifapi_netif_set_link_up 275
2.3.4.4.3.10 netifapi_netif_set_up 275
2.3.4.4.3.11 Function Documentation 275

2.3.4.4.3.12 netifapi_netif_add() . . . 275
2.3.4.4.3.13 netifapi_netif_index_to_name() . . . 275
2.3.4.4.3.14 netifapi_netif_name_to_index() . . . 275
2.3.4.4.3.15 netifapi_netif_set_addr() . . . 276
2.3.4.4.4 DHCPv4 . . . 276
2.3.4.4.4.1 Macros . . . 276
2.3.4.4.4.2 Detailed Description . . . 276
2.3.4.4.4.3 Macro Definition Documentation . . . 276
2.3.4.4.4.4 netifapi_dhcp_inform . . . 276
2.3.4.4.4.5 netifapi_dhcp_release . . . 276
2.3.4.4.4.6 netifapi_dhcp_release_and_stop . . . 277
2.3.4.4.4.7 netifapi_dhcp_renew . . . 277
2.3.4.4.4.8 netifapi_dhcp_start . . . 277
2.3.4.4.4.9 netifapi_dhcp_stop . . . 277
2.3.4.4.5 AUTOIP . . . 277
2.3.4.4.5.1 Macros . . . 277
2.3.4.4.5.2 Detailed Description . . . 277
2.3.4.4.5.3 Macro Definition Documentation . . . 277
2.3.4.4.5.4 netifapi_autoip_start . . . 277
2.3.4.4.5.5 netifapi_autoip_stop . . . 277
2.3.5 Socket API . . . 277
2.3.5.1 Modules . . . 277
2.3.5.2 Detailed Description . . . 278
2.3.5.3 Interface Identification API . . . 278
2.3.5.3.1 Functions . . . 278
2.3.5.3.2 Detailed Description . . . 278
2.3.5.3.3 Function Documentation . . . 278
2.3.5.3.3.1 lwip_if_indextoname() . . . 278
2.3.5.3.3.2 lwip_if_nametoindex() . . . 278
2.3.5.4 NETDB API . . . 278
2.3.5.4.1 Detailed Description . . . 278
2.4 NETIFs . . . 278
2.4.1 Modules . . . 278
2.4.2 Detailed Description . . . 279
2.4.3 IEEE 802.1D bridge . . . 279
2.4.3.1 Modules . . . 279
2.4.3.2 Data Structures . . . 279
2.4.3.3 Macros . . . 279
2.4.3.4 Typedefs . . . 279

2.4.3.5 Functions . . . 279
2.4.3.6 Detailed Description . . . 279
2.4.3.7 Macro Definition Documentation . . . 280
2.4.3.7.1 BRIDGEIF_INITDATA1 . . . 280
2.4.3.7.2 BRIDGEIF_INITDATA2 . . . 280
2.4.3.8 Typedef Documentation . . . 280
2.4.3.8.1 bridgeif_initdata_t . . . 280
2.4.3.9 Function Documentation . . . 280
2.4.3.9.1 bridgeif_add_port() . . . 280
2.4.3.9.2 bridgeif_fdb_add() . . . 281
2.4.3.9.3 bridgeif_fdb_remove() . . . 281
2.4.3.9.4 bridgeif_init() . . . 281
2.4.3.10 Options . . . 281
2.4.3.10.1 Macros . . . 281
2.4.3.10.2 Detailed Description . . . 281
2.4.3.10.3 Macro Definition Documentation . . . 281
2.4.3.10.3.1 BRIDGEIF_DEBUG . . . 281
2.4.3.10.3.2 BRIDGEIF_FDB_DEBUG . . . 281
2.4.3.10.3.3 BRIDGEIF_FW_DEBUG . . . 282
2.4.3.10.3.4 BRIDGEIF_MAX_PORTS . . . 282
2.4.3.10.3.5 BRIDGEIF_PORT_NETIFS_OUTPUT_DIRECT . . . 282
2.4.3.11 FDB example code . . . 282
2.4.3.11.1 Functions . . . 282
2.4.3.11.2 Detailed Description . . . 282
2.4.3.11.3 Function Documentation . . . 282
2.4.3.11.3.1 bridgeif_fdb_get_dst_ports() . . . 282
2.4.3.11.3.2 bridgeif_fdb_init() . . . 282
2.4.3.11.3.3 bridgeif_fdb_update_src() . . . 282
2.4.4 6LoWPAN (RFC4944) . . . 283
2.4.4.1 Functions . . . 283
2.4.4.2 Detailed Description . . . 283
2.4.4.3 Function Documentation . . . 283
2.4.4.3.1 lowpan6_input() . . . 283
2.4.4.3.2 lowpan6_output() . . . 283
2.4.4.3.3 lowpan6_set_context() . . . 283
2.4.4.3.4 lowpan6_set_pan_id() . . . 283
2.4.4.3.5 lowpan6_set_short_addr() . . . 284
2.4.4.3.6 tcpip_6lowpan_input() . . . 284
2.4.5 6LoWPAN over BLE (RFC7668) . . . 284

2.4.5.1 Functions . . . 284
2.4.5.2 Detailed Description . . . 284
2.4.5.3 Function Documentation . . . 285
2.4.5.3.1 ble_addr_to_eui64() . . . 285
2.4.5.3.2 eui64_to_ble_addr() . . . 285
2.4.5.3.3 rfc7668_if_init() . . . 285
2.4.5.3.4 rfc7668_input() . . . 285
2.4.5.3.5 rfc7668_output() . . . 286
2.4.5.3.6 rfc7668_set_context() . . . 286
2.4.6 PPP . . . 286
2.4.7 SLIP . . . 295
2.4.7.1 Functions . . . 295
2.4.7.2 Detailed Description . . . 295
2.4.7.3 Function Documentation . . . 295
2.4.7.3.1 slipif_init() . . . 295
2.4.7.3.2 slipif_poll() . . . 296
2.4.7.3.3 slipif_process_rxqueue() . . . 296
2.4.7.3.4 slipif_received_byte() . . . 296
2.4.7.3.5 slipif_received_bytes() . . . 296
2.4.8 ZEP - ZigBee Encapsulation Protocol . . . 296
2.4.8.1 Functions . . . 296
2.4.8.2 Detailed Description . . . 297
2.4.8.3 Function Documentation . . . 297
2.4.8.3.1 zepif_init() . . . 297
2.5 Applications . . . 297
2.5.1 Modules . . . 297
2.5.2 Detailed Description . . . 297
2.5.3 HTTP client . . . 297
2.5.3.1 Macros . . . 297
2.5.3.2 Typedefs . . . 298
2.5.3.3 Enumerations . . . 298
2.5.3.4 Functions . . . 298
2.5.3.5 Detailed Description . . . 298
2.5.3.6 Macro Definition Documentation . . . 298
2.5.3.6.1 HTTP_DEFAULT_PORT . . . 298
2.5.3.6.2 LWIP_HTTPC_HAVE_FILE_IO . . . 298
2.5.3.7 Typedef Documentation . . . 298
2.5.3.7.1 httpc_headers_done_fn . . . 298
2.5.3.7.2 httpc_result_fn . . . 299

2.5.3.7.3 httpc_result_t . . . 299
2.5.3.8 Enumeration Type Documentation . . . 299
2.5.3.8.1 ehttpc_result . . . 299
2.5.3.9 Function Documentation . . . 299
2.5.3.9.1 httpc_get_file() . . . 299
2.5.3.9.2 httpc_get_file_dns() . . . 300
2.5.4 HTTP server . . . 300
2.5.4.1 Modules . . . 300
2.5.4.2 Data Structures . . . 300
2.5.4.3 Typedefs . . . 300
2.5.4.4 Functions . . . 300
2.5.4.5 Detailed Description . . . 301
2.5.4.6 Typedef Documentation . . . 301
2.5.4.6.1 tCGIHandler . . . 301
2.5.4.6.2 tSSIHandler . . . 302
2.5.4.7 Function Documentation . . . 302
2.5.4.7.1 http_set_cgi_handlers() . . . 302
2.5.4.7.2 http_set_ssi_handler() . . . 302
2.5.4.7.3 httpd_init() . . . 302
2.5.4.7.4 httpd_inits() . . . 303
2.5.4.7.5 httpd_post_begin() . . . 303
2.5.4.7.6 httpd_post_data_recved() . . . 303
2.5.4.7.7 httpd_post_finished() . . . 303
2.5.4.7.8 httpd_post_receive_data() . . . 304
2.5.4.8 Options . . . 304
2.5.4.8.1 Macros . . . 304
2.5.4.8.2 Detailed Description . . . 305
2.5.4.8.3 Macro Definition Documentation . . . 305
2.5.4.8.3.1 HTTP_IS_TAG_VOLATILE . . . 305
2.5.4.8.3.2 HTTPD_DEBUG_TIMING . . . 305
2.5.4.8.3.3 HTTPD_ENABLE_HTTPS . . . 305
2.5.4.8.3.4 HTTPD_FSDATA_FILE . . . 305
2.5.4.8.3.5 HTTPD_MAX_RETRIES . . . 306
2.5.4.8.3.6 HTTPD_POLL_INTERVAL . . . 306
2.5.4.8.3.7 HTTPD_PRECALCULATED_CHECKSUM . . . 306
2.5.4.8.3.8 HTTPD_SERVER_AGENT . . . 306
2.5.4.8.3.9 HTTPD_SERVER_PORT . . . 306
2.5.4.8.3.10 HTTPD_SERVER_PORT_HTTPS . . . 306
2.5.4.8.3.11 HTTPD_TCP_PRIO . . . 306

2.5.4.8.3.12 HTTPD_USE_MEM_POOL 306
2.5.4.8.3.13 LWIP_HTTPD_ABORT_ON_CLOSE_MEM_ERROR 306
2.5.4.8.3.14 LWIP_HTTPD_CGI 307
2.5.4.8.3.15 LWIP_HTTPD_CGI_SSI 307
2.5.4.8.3.16 LWIP_HTTPD_CUSTOM_FILES 307
2.5.4.8.3.17 LWIP_HTTPD_DYNAMIC_FILE_READ 307
2.5.4.8.3.18 LWIP_HTTPD_DYNAMIC_HEADERS 307
2.5.4.8.3.19 LWIP_HTTPD_FILE_EXTENSION 307
2.5.4.8.3.20 LWIP_HTTPD_FILE_STATE 308
2.5.4.8.3.21 LWIP_HTTPD_FS_ASYNC_READ 308
2.5.4.8.3.22 LWIP_HTTPD_KILL_OLD_ON_CONNECTIONS_EXCEEDED 308
2.5.4.8.3.23 LWIP_HTTPD_MAX_REQ_LENGTH 308
2.5.4.8.3.24 LWIP_HTTPD_MAX_REQUEST_URI_LEN 308
2.5.4.8.3.25 LWIP_HTTPD_OMIT_HEADER_FOR_EXTENSIONLESS_URI 308
2.5.4.8.3.26 LWIP_HTTPD_POST_MAX_RESPONSE_URI_LEN 308
2.5.4.8.3.27 LWIP_HTTPD_REQ_BUFSIZE 308
2.5.4.8.3.28 LWIP_HTTPD_REQ_QUEUELEN 308
2.5.4.8.3.29 LWIP_HTTPD_SSI 309
2.5.4.8.3.30 LWIP_HTTPD_SSI_BY_FILE_EXTENSION 309
2.5.4.8.3.31 LWIP_HTTPD_SSI_EXTENSIONS 309
2.5.4.8.3.32 LWIP_HTTPD_SSI_INCLUDE_TAG 309
2.5.4.8.3.33 LWIP_HTTPD_SSI_MULTIPART 309
2.5.4.8.3.34 LWIP_HTTPD_SSI_RAW 309
2.5.4.8.3.35 LWIP_HTTPD_SUPPORT_11_KEEPALIVE 309
2.5.4.8.3.36 LWIP_HTTPD_SUPPORT_EXTSTATUS 310
2.5.4.8.3.37 LWIP_HTTPD_SUPPORT_POST 310
2.5.4.8.3.38 LWIP_HTTPD_SUPPORT_REQUESTLIST 310
2.5.4.8.3.39 LWIP_HTTPD_SUPPORT_V09 310
2.5.4.8.3.40 LWIP_HTTPD_TIMING 310
2.5.5 Iperf server 310
2.5.5.1 Functions 310
2.5.5.2 Detailed Description 310
2.5.5.3 Function Documentation 310
2.5.5.3.1 lwiperf_abort() 310
2.5.5.3.2 lwiperf_start_tcp_client() 311
2.5.5.3.3 lwiperf_start_tcp_client_default() 311
2.5.5.3.4 lwiperf_start_tcp_server() 311
2.5.5.3.5 lwiperf_start_tcp_server_default() 311
2.5.6 MDNS 311

2.5.6.1 Modules 311
2.5.6.2 Macros 311
2.5.6.3 Functions 311
2.5.6.4 Detailed Description 312
2.5.6.5 Macro Definition Documentation 314
2.5.6.5.1 mdns_resp_netif_settings_changed 314
2.5.6.6 Function Documentation 314
2.5.6.6.1 mdns_get_service_txt_userdata() 314
2.5.6.6.2 mdns_resp_add_netif() 315
2.5.6.6.3 mdns_resp_add_service() 315
2.5.6.6.4 mdns_resp_add_service_txtitem() 315
2.5.6.6.5 mdns_resp_announce() 315
2.5.6.6.6 mdns_resp_del_service() 316
2.5.6.6.7 mdns_resp_init() 316
2.5.6.6.8 mdns_resp_netif_active() 316
2.5.6.6.9 mdns_resp_remove_netif() 316
2.5.6.6.10 mdns_resp_rename_netif() 316
2.5.6.6.11 mdns_resp_rename_service() 317
2.5.6.6.12 mdns_resp_restart() 317
2.5.6.6.13 mdns_resp_restart_delay() 317
2.5.6.6.14 mdns_search_service() 317
2.5.6.6.15 mdns_search_stop() 317
2.5.6.7 Options 318
2.5.6.7.1 Macros 318
2.5.6.7.2 Detailed Description 318
2.5.6.7.3 Macro Definition Documentation 318
2.5.6.7.3.1 LWIP_MDNS_SEARCH 318
2.5.6.7.3.2 MDNS_DEBUG 318
2.5.6.7.3.3 MDNS_MAX_REQUESTS 318
2.5.6.7.3.4 MDNS_MAX_SERVICES 318
2.5.6.7.3.5 MDNS_MAX_STORED_PKTS 319
2.5.6.7.3.6 MDNS_OUTPUT_PACKET_SIZE 319
2.5.6.7.3.7 MDNS_PROBE_DELAY_MS 319
2.5.6.7.3.8 MDNS_RESP_USENETIF_EXTCALLBACK 319
2.5.7 MQTT client 319
2.5.7.1 Modules 319
2.5.7.2 Data Structures 319
2.5.7.3 Macros 320
2.5.7.4 Typedefs 320

2.5.7.5 Enumerations . . . 320
2.5.7.6 Functions . . . 320
2.5.7.7 Detailed Description . . . 320
2.5.7.8 Macro Definition Documentation . . . 324
2.5.7.8.1 MQTT_PORT . . . 324
2.5.7.8.2 mqtt_subscribe . . . 324
2.5.7.8.3 MQTT_TLS_PORT . . . 324
2.5.7.8.4 mqtt_unsubscribe . . . 324
2.5.7.9 Typedef Documentation . . . 324
2.5.7.9.1 mqtt_connection_cb_t . . . 324
2.5.7.9.2 mqtt_incoming_data_cb_t . . . 324
2.5.7.9.3 mqtt_incoming_publish_cb_t . . . 325
2.5.7.9.4 mqtt_request_cb_t . . . 325
2.5.7.10 Enumeration Type Documentation . . . 325
2.5.7.10.1 anonymous enum . . . 325
2.5.7.10.2 mqtt_connection_status_t . . . 325
2.5.7.11 Function Documentation . . . 326
2.5.7.11.1 mqtt_client_connect() . . . 326
2.5.7.11.2 mqtt_client_free() . . . 326
2.5.7.11.3 mqtt_client_is_connected() . . . 326
2.5.7.11.4 mqtt_client_new() . . . 326
2.5.7.11.5 mqtt_disconnect() . . . 326
2.5.7.11.6 mqtt_publish() . . . 327
2.5.7.11.7 mqtt_set_inpub_callback() . . . 327
2.5.7.11.8 mqtt_sub_unsub() . . . 327
2.5.7.12 Options . . . 327
2.5.7.12.1 Macros . . . 327
2.5.7.12.2 Detailed Description . . . 328
2.5.7.12.3 Macro Definition Documentation . . . 328
2.5.7.12.3.1 MQTT_CONNECT_TIMOUT . . . 328
2.5.7.12.3.2 MQTT_CYCLIC_TIMER_INTERVAL . . . 328
2.5.7.12.3.3 MQTT_OUTPUT_RINGBUF_SIZE . . . 328
2.5.7.12.3.4 MQTT_REQ_MAX_IN_FLIGHT . . . 328
2.5.7.12.3.5 MQTT_REQ_TIMEOUT . . . 328
2.5.7.12.3.6 MQTT_VAR_HEADER_BUFFER_LEN . . . 328
2.5.8 NETBIOS responder . . . 329
2.5.8.1 Modules . . . 329
2.5.8.2 Functions . . . 329
2.5.8.3 Detailed Description . . . 329

2.5.8.4 Function Documentation . . . 329
2.5.8.4.1 netbiosns_init() . . . 329
2.5.8.4.2 netbiosns_stop() . . . 329
2.5.8.5 Options . . . 329
2.5.8.5.1 Macros . . . 329
2.5.8.5.2 Detailed Description . . . 329
2.5.8.5.3 Macro Definition Documentation . . . 329
2.5.8.5.3.1 LWIP_NETBIOS_RESPOND_NAME_QUERY . . . 329
2.5.8.5.3.2 NETBIOS_LWIP_NAME . . . 330
2.5.9 SMTP client . . . 330
2.5.9.1 Modules . . . 330
2.5.9.2 Functions . . . 330
2.5.9.3 Detailed Description . . . 330
2.5.9.4 Function Documentation . . . 331
2.5.9.4.1 smtp_send_mail() . . . 331
2.5.9.4.2 smtp_send_mail_int() . . . 331
2.5.9.4.3 smtp_send_mail_static() . . . 332
2.5.9.4.4 smtp_set_auth() . . . 332
2.5.9.4.5 smtp_set_server_addr() . . . 332
2.5.9.4.6 smtp_set_server_port() . . . 332
2.5.9.4.7 smtp_set_tls_config() . . . 332
2.5.9.5 Options . . . 333
2.5.9.5.1 Macros . . . 333
2.5.9.5.2 Detailed Description . . . 333
2.5.9.5.3 Macro Definition Documentation . . . 333
2.5.9.5.3.1 SMTP_BODYDH . . . 333
2.5.9.5.3.2 SMTP_CHECK_DATA . . . 333
2.5.9.5.3.3 SMTP_COPY_AUTHDATA . . . 333
2.5.9.5.3.4 SMTP_DEBUG . . . 333
2.5.9.5.3.5 SMTP_MAX_PASS_LEN . . . 333
2.5.9.5.3.6 SMTP_MAX_SERVERNAME_LEN . . . 334
2.5.9.5.3.7 SMTP_MAX_USERNAME_LEN . . . 334
2.5.9.5.3.8 SMTP_SUPPORT_AUTH_LOGIN . . . 334
2.5.9.5.3.9 SMTP_SUPPORT_AUTH_PLAIN . . . 334
2.5.10 SNMPv2c/v3 agent . . . 334
2.5.10.1 Modules . . . 334
2.5.10.2 Detailed Description . . . 334
2.5.10.3 Core . . . 336
2.5.10.3.1 Functions . . . 336

2.5.10.3.2 Detailed Description . . . 337
2.5.10.3.3 Function Documentation . . . 337
2.5.10.3.3.1 snmp_get_community() . . . 337
2.5.10.3.3.2 snmp_get_community_write() . . . 337
2.5.10.3.3.3 snmp_get_device_enterprise_oid() . . . 337
2.5.10.3.3.4 snmp_init() . . . 337
2.5.10.3.3.5 snmp_set_community() . . . 337
2.5.10.3.3.6 snmp_set_community_write() . . . 337
2.5.10.3.3.7 snmp_set_device_enterprise_oid() . . . 338
2.5.10.3.3.8 snmp_set_inform_callback() . . . 338
2.5.10.3.3.9 snmp_set_mibs() . . . 338
2.5.10.3.3.10 snmp_set_write_callback() . . . 338
2.5.10.4 Traps . . . 338
2.5.10.4.1 Functions . . . 338
2.5.10.4.2 Detailed Description . . . 339
2.5.10.4.3 Function Documentation . . . 339
2.5.10.4.3.1 snmp_authfail_trap() . . . 339
2.5.10.4.3.2 snmp_coldstart_trap() . . . 339
2.5.10.4.3.3 snmp_get_auth_traps_enabled() . . . 339
2.5.10.4.3.4 snmp_get_community_trap() . . . 339
2.5.10.4.3.5 snmp_get_default_trap_version() . . . 339
2.5.10.4.3.6 snmp_send_inform() . . . 339
2.5.10.4.3.7 snmp_send_inform_generic() . . . 340
2.5.10.4.3.8 snmp_send_inform_specific() . . . 340
2.5.10.4.3.9 snmp_send_trap() . . . 340
2.5.10.4.3.10 snmp_send_trap_generic() . . . 341
2.5.10.4.3.11 snmp_send_trap_specific() . . . 341
2.5.10.4.3.12 snmp_set_auth_traps_enabled() . . . 341
2.5.10.4.3.13 snmp_set_community_trap() . . . 341
2.5.10.4.3.14 snmp_set_default_trap_version() . . . 341
2.5.10.4.3.15 snmp_trap_dst_enable() . . . 341
2.5.10.4.3.16 snmp_trap_dst_ip_set() . . . 342
2.5.10.5 MIB2 . . . 342
2.5.10.5.1 Functions . . . 342
2.5.10.5.2 Detailed Description . . . 342
2.5.10.5.3 Function Documentation . . . 342
2.5.10.5.3.1 snmp_mib2_set_syscontact() . . . 342
2.5.10.5.3.2 snmp_mib2_set_syscontact_readonly() . . . 342
2.5.10.5.3.3 snmp_mib2_set_sysdescr() . . . 342

2.5.10.5.3.4 snmp_mib2_set_syslocation() 343
2.5.10.5.3.5 snmp_mib2_set_syslocation_readonly() 343
2.5.10.5.3.6 snmp_mib2_set_sysname() 343
2.5.10.5.3.7 snmp_mib2_set_sysname_readonly() 343
2.5.10.6 Options 343
2.5.10.6.1 Macros 343
2.5.10.6.2 Detailed Description 345
2.5.10.6.3 Macro Definition Documentation 345
2.5.10.6.3.1 LWIP_SNMP 345
2.5.10.6.3.2 SNMP_COMMUNITY 345
2.5.10.6.3.3 SNMP_COMMUNITY_TRAP 345
2.5.10.6.3.4 SNMP_COMMUNITY_WRITE 345
2.5.10.6.3.5 SNMP_DEBUG 345
2.5.10.6.3.6 SNMP_DEVICE_ENTERPRISE_OID 345
2.5.10.6.3.7 SNMP_DEVICE_ENTERPRISE_OID_LEN 345
2.5.10.6.3.8 SNMP_LWIP_ENTERPRISE_OID 346
2.5.10.6.3.9 SNMP_LWIP_GETBULK_MAX_REPETITIONS 346
2.5.10.6.3.10 SNMP_LWIP_MIB2 346
2.5.10.6.3.11 SNMP_LWIP_MIB2_SYSCONTACT 346
2.5.10.6.3.12 SNMP_LWIP_MIB2_SYSDESC 346
2.5.10.6.3.13 SNMP_LWIP_MIB2_SYSLOCATION 346
2.5.10.6.3.14 SNMP_LWIP_MIB2_SYSNAME 346
2.5.10.6.3.15 SNMP_MAX_COMMUNITY_STR_LEN 346
2.5.10.6.3.16 SNMP_MAX_OBJ_ID_LEN 347
2.5.10.6.3.17 SNMP_MAX_OCTET_STRING_LEN 347
2.5.10.6.3.18 SNMP_MAX_VALUE_SIZE 347
2.5.10.6.3.19 SNMP_MIB_DEBUG 347
2.5.10.6.3.20 SNMP_MIN_VALUE_SIZE 347
2.5.10.6.3.21 SNMP_SAFE_REQUESTS 347
2.5.10.6.3.22 SNMP_STACK_SIZE 347
2.5.10.6.3.23 SNMP_THREAD_PRIO 347
2.5.10.6.3.24 SNMP_TRAP_DESTINATIONS 347
2.5.10.6.3.25 SNMP_USE_NETCONN 348
2.5.10.6.3.26 SNMP_USE_RAW 348
2.5.11 SNTP 348
2.5.11.1 Modules 348
2.5.11.2 Functions 348
2.5.11.3 Detailed Description 348
2.5.11.4 Function Documentation 348

2.5.11.4.1 sntp_enabled() 348
2.5.11.4.2 sntp_getkodreceived() 349
2.5.11.4.3 sntp_getoperatingmode() 349
2.5.11.4.4 sntp_getreachability() 349
2.5.11.4.5 sntp_getserver() 349
2.5.11.4.6 sntp_init() 349
2.5.11.4.7 sntp_setoperatingmode() 349
2.5.11.4.8 sntp_setserver() 350
2.5.11.4.9 sntp_stop() 350
2.5.11.5 Options 350
2.5.11.5.1 Macros 350
2.5.11.5.2 Detailed Description 351
2.5.11.5.3 Macro Definition Documentation 351
2.5.11.5.3.1 SNTP_CHECK_RESPONSE 351
2.5.11.5.3.2 SNTP_COMP_ROUNDTRIP 351
2.5.11.5.3.3 SNTP_DEBUG 351
2.5.11.5.3.4 SNTP_GET_SERVERS_FROM_DHCP 351
2.5.11.5.3.5 SNTP_GET_SERVERS_FROM_DHCPV6 351
2.5.11.5.3.6 SNTP_GET_SYSTEM_TIME 351
2.5.11.5.3.7 SNTP_MAX_SERVERS 352
2.5.11.5.3.8 SNTP_MONITOR_SERVER_REACHABILITY 352
2.5.11.5.3.9 SNTP_PORT 352
2.5.11.5.3.10 SNTP_RECV_TIMEOUT 352
2.5.11.5.3.11 SNTP_RETRY_TIMEOUT 352
2.5.11.5.3.12 SNTP_RETRY_TIMEOUT_EXP 352
2.5.11.5.3.13 SNTP_RETRY_TIMEOUT_MAX 352
2.5.11.5.3.14 SNTP_SERVER_DNS 352
2.5.11.5.3.15 SNTP_SET_SYSTEM_TIME 352
2.5.11.5.3.16 SNTP_STARTUP_DELAY 353
2.5.11.5.3.17 SNTP_STARTUP_DELAY_FUNC 353
2.5.11.5.3.18 SNTP_UPDATE_DELAY 353
2.5.12 TFTP client/server 353
2.5.12.1 Modules 353
2.5.12.2 Data Structures 353
2.5.12.3 Functions 353
2.5.12.4 Detailed Description 353
2.5.12.5 Function Documentation 353
2.5.12.5.1 tftp_cleanup() 353
2.5.12.5.2 tftp_init_client() 353

2.5.12.5.3 tftp_init_server() . . . 354
2.5.12.6 Options . . . 354
2.5.12.6.1 Macros . . . 354
2.5.12.6.2 Detailed Description . . . 354
2.5.12.6.3 Macro Definition Documentation . . . 354
2.5.12.6.3.1 TFTP_DEBUG . . . 354
2.5.12.6.3.2 TFTP_MAX_FILENAME_LEN . . . 354
2.5.12.6.3.3 TFTP_MAX_MODE_LEN . . . 354
2.5.12.6.3.4 TFTP_MAX_RETRIES . . . 354
2.5.12.6.3.5 TFTP_PORT . . . 355
2.5.12.6.3.6 TFTP_TIMEOUT_MSECS . . . 355
2.5.12.6.3.7 TFTP_TIMER_MSECS . . . 355

3 Data Structure Documentation 356
3.1 _lwiperf_settings Struct Reference . . . 356
3.1.1 Detailed Description . . . 356
3.2 _lwiperf_state_tcp Struct Reference . . . 356
3.2.1 Detailed Description . . . 356
3.3 acd Struct Reference . . . 356
3.3.1 Data Fields . . . 356
3.3.2 Detailed Description . . . 357
3.3.3 Field Documentation . . . 357
3.3.3.1 acd_conflict_callback . . . 357
3.3.3.2 ipaddr . . . 357
3.3.3.3 lastconflict . . . 357
3.3.3.4 next . . . 357
3.3.3.5 num_conflicts . . . 357
3.3.3.6 sent_num . . . 357
3.3.3.7 state . . . 357
3.3.3.8 ttw . . . 357
3.4 altcp_allocator_s Struct Reference . . . 358
3.4.1 Data Fields . . . 358
3.4.2 Detailed Description . . . 358
3.4.3 Field Documentation . . . 358
3.4.3.1 alloc . . . 358
3.4.3.2 arg . . . 358
3.5 api_msg Struct Reference . . . 358
3.5.1 Data Fields . . . 358
3.5.2 Detailed Description . . . 359

3.5.3 Field Documentation . . . 359
3.5.3.1 [struct] . . . 359
3.5.3.2 b . . . 359
3.5.3.3 [struct] . . . 359
3.5.3.4 conn . . . 359
3.5.3.5 err . . . 359
3.5.3.6 [struct] . . . 360
3.5.3.7 len . . . 360
3.5.3.8 [union] . . . 360
3.5.3.9 [struct] . . . 360
3.5.3.10 offset . . . 360
3.5.3.11 [struct] . . . 360
3.5.3.12 [struct] . . . 360
3.5.3.13 vector . . . 360
3.5.3.14 vector_cnt . . . 360
3.5.3.15 vector_off . . . 361
3.5.3.16 [struct] . . . 361
3.6 autoip Struct Reference . . . 361
3.6.1 Data Fields . . . 361
3.6.2 Detailed Description . . . 361
3.6.3 Field Documentation . . . 361
3.6.3.1 acd . . . 361
3.6.3.2 llipaddr . . . 361
3.6.3.3 state . . . 361
3.6.3.4 tried_llipaddr . . . 362
3.7 bridgeif_initdata_s Struct Reference . . . 362
3.7.1 Data Fields . . . 362
3.7.2 Detailed Description . . . 362
3.7.3 Field Documentation . . . 362
3.7.3.1 ethaddr . . . 362
3.7.3.2 max_fdb_dynamic_entries . . . 362
3.7.3.3 max_fdb_static_entries . . . 362
3.7.3.4 max_ports . . . 362
3.8 dhcp6_msg Struct Reference . . . 363
3.8.1 Detailed Description . . . 363
3.9 dhcp_msg Struct Reference . . . 363
3.9.1 Detailed Description . . . 363
3.10 dns_answer Struct Reference . . . 363
3.10.1 Detailed Description . . . 363

3.11 dns_api_msg Struct Reference . . . 363
3.11.1 Data Fields . . . 363
3.11.2 Detailed Description . . . 363
3.11.3 Field Documentation . . . 364
3.11.3.1 addr . . . 364
3.11.3.2 dns_addrtype . . . 364
3.11.3.3 err . . . 364
3.11.3.4 name . . . 364
3.11.3.5 sem . . . 364
3.12 dns_hdr Struct Reference . . . 364
3.12.1 Detailed Description . . . 364
3.13 dns_query Struct Reference . . . 364
3.13.1 Detailed Description . . . 364
3.14 dns_req_entry Struct Reference . . . 365
3.14.1 Detailed Description . . . 365
3.15 dns_table_entry Struct Reference . . . 365
3.15.1 Detailed Description . . . 365
3.16 eth_addr Struct Reference . . . 365
3.16.1 Detailed Description . . . 365
3.17 eth_hdr Struct Reference . . . 365
3.17.1 Detailed Description . . . 365
3.18 eth_vlan_hdr Struct Reference . . . 365
3.18.1 Detailed Description . . . 365
3.19 etharp_hdr Struct Reference . . . 365
3.19.1 Detailed Description . . . 366
3.20 etharp_q_entry Struct Reference . . . 366
3.20.1 Detailed Description . . . 366
3.21 gethostbyname_r_helper Struct Reference . . . 366
3.21.1 Detailed Description . . . 366
3.22 icmp6_echo_hdr Struct Reference . . . 366
3.22.1 Detailed Description . . . 366
3.23 icmp6_hdr Struct Reference . . . 366
3.23.1 Detailed Description . . . 366
3.24 icmp_echo_hdr Struct Reference . . . 366
3.24.1 Detailed Description . . . 367
3.25 icmp_hdr Struct Reference . . . 367
3.25.1 Detailed Description . . . 367
3.26 ieee_802154_hdr Struct Reference . . . 367
3.26.1 Data Fields . . . 367

3.26.2 Detailed Description . . . 367
3.26.3 Field Documentation . . . 367
3.26.3.1 destination_address . . . 367
3.26.3.2 destination_pan_id . . . 367
3.26.3.3 frame_control . . . 368
3.26.3.4 sequence_number . . . 368
3.26.3.5 source_address . . . 368
3.26.3.6 source_pan_id . . . 368
3.27 igmp_group Struct Reference . . . 368
3.27.1 Data Fields . . . 368
3.27.2 Detailed Description . . . 368
3.27.3 Field Documentation . . . 368
3.27.3.1 group_address . . . 368
3.27.3.2 group_state . . . 369
3.27.3.3 last_reporter_flag . . . 369
3.27.3.4 next . . . 369
3.27.3.5 timer . . . 369
3.27.3.6 use . . . 369
3.28 igmp_msg Struct Reference . . . 369
3.28.1 Detailed Description . . . 369
3.29 ip4_addr Struct Reference . . . 369
3.29.1 Detailed Description . . . 369
3.30 ip4_addr_packed Struct Reference . . . 370
3.30.1 Detailed Description . . . 370
3.31 ip4_addr_wordaligned Struct Reference . . . 370
3.31.1 Detailed Description . . . 370
3.32 ip6_addr Struct Reference . . . 370
3.32.1 Detailed Description . . . 370
3.33 ip6_addr_packed Struct Reference . . . 370
3.33.1 Detailed Description . . . 370
3.34 ip6_hdr Struct Reference . . . 370
3.34.1 Data Fields . . . 371
3.34.2 Detailed Description . . . 371
3.34.3 Field Documentation . . . 371
3.34.3.1 _hoplim . . . 371
3.34.3.2 _nexth . . . 371
3.34.3.3 _plen . . . 371
3.34.3.4 _v_tc_fl . . . 371
3.34.3.5 src . . . 371

3.35 ip6_reass_helper Struct Reference . . . 371
3.35.1 Detailed Description . . . 371
3.36 ip6_reassdata Struct Reference . . . 372
3.36.1 Detailed Description . . . 372
3.37 ip_addr Struct Reference . . . 372
3.37.1 Data Fields . . . 372
3.37.2 Detailed Description . . . 372
3.37.3 Field Documentation . . . 372
3.37.3.1 type . . . 372
3.38 ip_globals Struct Reference . . . 372
3.38.1 Data Fields . . . 372
3.38.2 Detailed Description . . . 373
3.38.3 Field Documentation . . . 373
3.38.3.1 current_input_netif . . . 373
3.38.3.2 current_ip4_header . . . 373
3.38.3.3 current_ip6_header . . . 373
3.38.3.4 current_ip_header_tot_len . . . 373
3.38.3.5 current_iphdr_dest . . . 373
3.38.3.6 current_iphdr_src . . . 373
3.38.3.7 current_netif . . . 373
3.39 ip_reass_helper Struct Reference . . . 373
3.39.1 Detailed Description . . . 373
3.40 ip_reassdata Struct Reference . . . 374
3.40.1 Detailed Description . . . 374
3.41 netif_ext_callback_args_t::ipv4_changed_s Struct Reference . . . 374
3.41.1 Data Fields . . . 374
3.41.2 Detailed Description . . . 374
3.41.3 Field Documentation . . . 374
3.41.3.1 old_address . . . 374
3.42 netif_ext_callback_args_t::ipv6_addr_state_changed_s Struct Reference . . . 374
3.42.1 Data Fields . . . 374
3.42.2 Detailed Description . . . 374
3.42.3 Field Documentation . . . 375
3.42.3.1 addr_index . . . 375
3.42.3.2 address . . . 375
3.42.3.3 old_state . . . 375
3.43 netif_ext_callback_args_t::ipv6_set_s Struct Reference . . . 375
3.43.1 Data Fields . . . 375
3.43.2 Detailed Description . . . 375

3.43.3 Field Documentation . . . 375
3.43.3.1 addr_index . . . 375
3.43.3.2 old_address . . . 375
3.44 netif_ext_callback_args_t::link_changed_s Struct Reference . . . 376
3.44.1 Data Fields . . . 376
3.44.2 Detailed Description . . . 376
3.44.3 Field Documentation . . . 376
3.44.3.1 state . . . 376
3.45 lowpan6_ieee802154_data Struct Reference . . . 376
3.45.1 Data Fields . . . 376
3.45.2 Detailed Description . . . 376
3.45.3 Field Documentation . . . 376
3.45.3.1 ieee_802154_pan_id . . . 376
3.45.3.2 lowpan6_context . . . 376
3.45.3.3 reass_list . . . 377
3.45.3.4 tx_datagram_tag . . . 377
3.45.3.5 tx_frame_seq_num . . . 377
3.46 lowpan6_link_addr Struct Reference . . . 377
3.46.1 Detailed Description . . . 377
3.47 lowpan6_reass_helper Struct Reference . . . 377
3.47.1 Detailed Description . . . 377
3.48 lwip_cyclic_timer Struct Reference . . . 377
3.48.1 Detailed Description . . . 377
3.49 lwip_select_cb Struct Reference . . . 377
3.49.1 Data Fields . . . 378
3.49.2 Detailed Description . . . 378
3.49.3 Field Documentation . . . 378
3.49.3.1 exceptset . . . 378
3.49.3.2 next . . . 378
3.49.3.3 poll_fds . . . 378
3.49.3.4 poll_nfds . . . 378
3.49.3.5 prev . . . 378
3.49.3.6 readset . . . 379
3.49.3.7 sem . . . 379
3.49.3.8 sem_signalled . . . 379
3.49.3.9 writeset . . . 379
3.50 lwip_sock Struct Reference . . . 379
3.50.1 Data Fields . . . 379
3.50.2 Detailed Description . . . 379

3.50.3 Field Documentation . . . 379
3.50.3.1 conn . . . 379
3.50.3.2 errevent . . . 380
3.50.3.3 lastdata . . . 380
3.50.3.4 rcvevent . . . 380
3.50.3.5 select_waiting . . . 380
3.50.3.6 sendevent . . . 380
3.51 mdns_delayed_msg Struct Reference . . . 380
3.51.1 Data Fields . . . 380
3.51.2 Detailed Description . . . 380
3.51.3 Field Documentation . . . 381
3.51.3.1 delayed_msg_multicast . . . 381
3.51.3.2 delayed_msg_unicast . . . 381
3.51.3.3 multicast_msg_waiting . . . 381
3.51.3.4 multicast_probe_timeout . . . 381
3.51.3.5 multicast_timeout . . . 381
3.51.3.6 multicast_timeout_25TTL . . . 381
3.51.3.7 unicast_msg_in_use . . . 381
3.52 mdns_host Struct Reference . . . 381
3.52.1 Data Fields . . . 382
3.52.2 Detailed Description . . . 382
3.52.3 Field Documentation . . . 382
3.52.3.1 conflict_time . . . 382
3.52.3.2 index . . . 382
3.52.3.3 ipv4 . . . 382
3.52.3.4 ipv6 . . . 382
3.52.3.5 name . . . 382
3.52.3.6 num_conflicts . . . 383
3.52.3.7 rate_limit_activated . . . 383
3.52.3.8 sent_num . . . 383
3.52.3.9 services . . . 383
3.52.3.10 state . . . 383
3.53 mdns_outmsg Struct Reference . . . 383
3.53.1 Data Fields . . . 383
3.53.2 Detailed Description . . . 383
3.53.3 Field Documentation . . . 384
3.53.3.1 cache_flush . . . 384
3.53.3.2 dest_addr . . . 384
3.53.3.3 flags . . . 384

3.53.3.4 legacy_query . . . 384
3.53.3.5 probe_query_recv . . . 384
3.53.3.6 query . . . 384
3.53.3.7 tx_id . . . 384
3.53.3.8 unicast_reply_requested . . . 384
3.54 mdns_outpacket Struct Reference . . . 384
3.54.1 Data Fields . . . 385
3.54.2 Detailed Description . . . 385
3.54.3 Field Documentation . . . 385
3.54.3.1 additional . . . 385
3.54.3.2 answers . . . 385
3.54.3.3 authoritative . . . 385
3.54.3.4 domain_offsets . . . 385
3.54.3.5 pbuf . . . 385
3.54.3.6 questions . . . 385
3.54.3.7 write_offset . . . 386
3.55 mdns_packet Struct Reference . . . 386
3.55.1 Data Fields . . . 386
3.55.2 Detailed Description . . . 386
3.55.3 Field Documentation . . . 386
3.55.3.1 additional . . . 386
3.55.3.2 additional_left . . . 386
3.55.3.3 answers . . . 387
3.55.3.4 answers_left . . . 387
3.55.3.5 authoritative . . . 387
3.55.3.6 authoritative_left . . . 387
3.55.3.7 next_answer . . . 387
3.55.3.8 next_tc_question . . . 387
3.55.3.9 parse_offset . . . 387
3.55.3.10 pbuf . . . 387
3.55.3.11 questions . . . 387
3.55.3.12 questions_left . . . 388
3.55.3.13 recv_unicast . . . 388
3.55.3.14 source_addr . . . 388
3.55.3.15 tx_id . . . 388
3.56 mdns_request Struct Reference . . . 388
3.56.1 Data Fields . . . 388
3.56.2 Detailed Description . . . 388
3.56.3 Field Documentation . . . 388

3.56.3.1 name . . . 388
3.56.3.2 only_ptr . . . 389
3.56.3.3 proto . . . 389
3.56.3.4 qtype . . . 389
3.56.3.5 result_fn . . . 389
3.56.3.6 service . . . 389
3.57 mdns_rr_info Struct Reference . . . 389
3.57.1 Detailed Description . . . 389
3.58 mdns_service Struct Reference . . . 389
3.58.1 Data Fields . . . 389
3.58.2 Detailed Description . . . 390
3.58.3 Field Documentation . . . 390
3.58.3.1 name . . . 390
3.58.3.2 port . . . 390
3.58.3.3 proto . . . 390
3.58.3.4 service . . . 390
3.58.3.5 txt_fn . . . 390
3.58.3.6 txtdata . . . 390
3.59 mem Struct Reference . . . 390
3.59.1 Data Fields . . . 390
3.59.2 Detailed Description . . . 391
3.59.3 Field Documentation . . . 391
3.59.3.1 next . . . 391
3.59.3.2 prev . . . 391
3.59.3.3 used . . . 391
3.60 memp_desc Struct Reference . . . 391
3.60.1 Data Fields . . . 391
3.60.2 Detailed Description . . . 391
3.60.3 Field Documentation . . . 391
3.60.3.1 base . . . 391
3.60.3.2 num . . . 392
3.60.3.3 size . . . 392
3.60.3.4 stats . . . 392
3.60.3.5 tab . . . 392
3.61 mld_group Struct Reference . . . 392
3.61.1 Data Fields . . . 392
3.61.2 Detailed Description . . . 392
3.61.3 Field Documentation . . . 392
3.61.3.1 group_address . . . 392

3.61.3.2 group_state . . . 393
3.61.3.3 last_reporter_flag . . . 393
3.61.3.4 next . . . 393
3.61.3.5 timer . . . 393
3.61.3.6 use . . . 393
3.62 mld_header Struct Reference . . . 393
3.62.1 Detailed Description . . . 393
3.63 mqtt_client_s Struct Reference . . . 393
3.63.1 Data Fields . . . 394
3.63.2 Detailed Description . . . 394
3.63.3 Field Documentation . . . 394
3.63.3.1 conn_state . . . 394
3.63.3.2 connect_arg . . . 394
3.63.3.3 cyclic_tick . . . 394
3.63.3.4 data_cb . . . 394
3.63.3.5 inpub_pkt_id . . . 394
3.63.3.6 msg_idx . . . 395
3.63.3.7 output . . . 395
3.63.3.8 pend_req_queue . . . 395
3.63.3.9 pkt_id_seq . . . 395
3.64 mqtt_connect_client_info_t Struct Reference . . . 395
3.64.1 Data Fields . . . 395
3.64.2 Detailed Description . . . 395
3.64.3 Field Documentation . . . 396
3.64.3.1 client_id . . . 396
3.64.3.2 client_pass . . . 396
3.64.3.3 client_user . . . 396
3.64.3.4 keep_alive . . . 396
3.64.3.5 tls_config . . . 396
3.64.3.6 will_msg . . . 396
3.64.3.7 will_msg_len . . . 396
3.64.3.8 will_qos . . . 396
3.64.3.9 will_retain . . . 396
3.64.3.10 will_topic . . . 397
3.65 mqtt_request_t Struct Reference . . . 397
3.65.1 Data Fields . . . 397
3.65.2 Detailed Description . . . 397
3.65.3 Field Documentation . . . 397
3.65.3.1 cb . . . 397

3.65.3.2 next . . . 397
3.65.3.3 pkt_id . . . 397
3.65.3.4 timeout_diff . . . 397
3.66 mqtt_ringbuf_t Struct Reference . . . 398
3.66.1 Detailed Description . . . 398
3.67 na_header Struct Reference . . . 398
3.67.1 Detailed Description . . . 398
3.68 nd6_neighbor_cache_entry Struct Reference . . . 398
3.68.1 Data Fields . . . 398
3.68.2 Detailed Description . . . 398
3.68.3 Field Documentation . . . 398
3.68.3.1 q . . . 398
3.69 nd6_q_entry Struct Reference . . . 398
3.69.1 Detailed Description . . . 398
3.70 netbios_answer Struct Reference . . . 399
3.70.1 Data Fields . . . 399
3.70.2 Detailed Description . . . 399
3.70.3 Field Documentation . . . 399
3.70.3.1 answer_name . . . 399
3.70.3.2 answer_name_flags . . . 400
3.70.3.3 jumpers . . . 400
3.70.3.4 max_number_of_pending_sessions . . . 400
3.70.3.5 max_total_number_of_command_blocks . . . 400
3.70.3.6 max_total_sessions_possible . . . 400
3.70.3.7 name_size . . . 400
3.70.3.8 number_of_alignment_errors . . . 400
3.70.3.9 number_of_collisions . . . 400
3.70.3.10 number_of_crcs . . . 400
3.70.3.11 number_of_free_command_blocks . . . 401
3.70.3.12 number_of_good_receives . . . 401
3.70.3.13 number_of_good_sends . . . 401
3.70.3.14 number_of_names . . . 401
3.70.3.15 number_of_no_resource_condition . . . 401
3.70.3.16 number_of_pending_sessions . . . 401
3.70.3.17 number_of_retransmits . . . 401
3.70.3.18 number_of_send_aborts . . . 401
3.70.3.19 period_of_statistics . . . 401
3.70.3.20 query_name . . . 402
3.70.3.21 session_data_packet_size . . . 402

3.70.3.22 test_result . . . 402
3.70.3.23 total_number_of_command_blocks . . . 402
3.70.3.24 unit_id . . . 402
3.70.3.25 version_number . . . 402
3.71 netbios_hdr Struct Reference . . . 402
3.71.1 Detailed Description . . . 402
3.72 netbios_name_hdr Struct Reference . . . 402
3.72.1 Detailed Description . . . 402
3.73 netbios_question_hdr Struct Reference . . . 403
3.73.1 Detailed Description . . . 403
3.74 netbios_resp Struct Reference . . . 403
3.74.1 Detailed Description . . . 403
3.75 netbuf Struct Reference . . . 403
3.75.1 Detailed Description . . . 403
3.76 netconn Struct Reference . . . 403
3.76.1 Data Fields . . . 403
3.76.2 Detailed Description . . . 404
3.76.3 Field Documentation . . . 404
3.76.3.1 acceptmbox . . . 404
3.76.3.2 callback . . . 404
3.76.3.3 current_msg . . . 404
3.76.3.4 flags . . . 404
3.76.3.5 linger . . . 404
3.76.3.6 op_completed . . . 404
3.76.3.7 [union] . . . 404
3.76.3.8 pending_err . . . 405
3.76.3.9 recv_avail . . . 405
3.76.3.10 recv_bufsize . . . 405
3.76.3.11 recvmbox . . . 405
3.76.3.12 send_timeout . . . 405
3.76.3.13 state . . . 405
3.76.3.14 type . . . 405
3.77 netif Struct Reference . . . 405
3.77.1 Data Fields . . . 406
3.77.2 Detailed Description . . . 406
3.77.3 Field Documentation . . . 407
3.77.3.1 flags . . . 407
3.77.3.2 hwaddr . . . 407
3.77.3.3 hwaddr_len . . . 407

3.77.3.4 igmp_mac_filter 407
3.77.3.5 input 407
3.77.3.6 ip6_addr 407
3.77.3.7 ip6_addr_state 407
3.77.3.8 ip6_addr_valid_life 407
3.77.3.9 ip6_autoconfig_enabled 407
3.77.3.10 ip_addr 408
3.77.3.11 link_callback 408
3.77.3.12 link_speed 408
3.77.3.13 link_type 408
3.77.3.14 linkoutput 408
3.77.3.15 mib2_counters 408
3.77.3.16 mld_mac_filter 408
3.77.3.17 mtu 408
3.77.3.18 mtu6 408
3.77.3.19 name 409
3.77.3.20 next 409
3.77.3.21 num 409
3.77.3.22 output 409
3.77.3.23 output_ip6 409
3.77.3.24 remove_callback 409
3.77.3.25 rs_count 409
3.77.3.26 state 409
3.77.3.27 status_callback 409
3.77.3.28 ts 410
3.78 netif_ext_callback_args_t Union Reference 410
3.78.1 Data Structures 410
3.78.2 Detailed Description 410
3.79 netvector Struct Reference 410
3.79.1 Data Fields 410
3.79.2 Detailed Description 410
3.79.3 Field Documentation 410
3.79.3.1 len 410
3.79.3.2 ptr 411
3.80 ns_header Struct Reference 411
3.80.1 Detailed Description 411
3.81 pbuf Struct Reference 411
3.81.1 Data Fields 411
3.81.2 Detailed Description 411

3.81.3 Field Documentation . . . 411
3.81.3.1 flags . . . 411
3.81.3.2 if_idx . . . 412
3.81.3.3 len . . . 412
3.81.3.4 next . . . 412
3.81.3.5 payload . . . 412
3.81.3.6 ref . . . 412
3.81.3.7 tot_len . . . 412
3.81.3.8 type_internal . . . 412
3.82 pbuf_custom Struct Reference . . . 412
3.82.1 Data Fields . . . 412
3.82.2 Detailed Description . . . 413
3.82.3 Field Documentation . . . 413
3.82.3.1 custom_free_function . . . 413
3.82.3.2 pbuf . . . 413
3.83 pbuf_custom_ref Struct Reference . . . 413
3.83.1 Data Fields . . . 413
3.83.2 Detailed Description . . . 413
3.83.3 Field Documentation . . . 413
3.83.3.1 original . . . 413
3.83.3.2 pc . . . 413
3.84 pbuf_rom Struct Reference . . . 413
3.84.1 Data Fields . . . 414
3.84.2 Detailed Description . . . 414
3.84.3 Field Documentation . . . 414
3.84.3.1 next . . . 414
3.84.3.2 payload . . . 414
3.85 raw_pcb Struct Reference . . . 414
3.85.1 Data Fields . . . 414
3.85.2 Detailed Description . . . 414
3.85.3 Field Documentation . . . 414
3.85.3.1 mcast_ifindex . . . 414
3.85.3.2 mcast_ttl . . . 415
3.85.3.3 recv . . . 415
3.86 redirect_header Struct Reference . . . 415
3.86.1 Detailed Description . . . 415
3.87 rs_header Struct Reference . . . 415
3.87.1 Detailed Description . . . 415
3.88 smtp_send_request Struct Reference . . . 415

3.88.1 Data Fields . . . 415
3.88.2 Detailed Description . . . 415
3.88.3 Field Documentation . . . 416
3.88.3.1 static_data . . . 416
3.89 smtp_session Struct Reference . . . 416
3.89.1 Data Fields . . . 416
3.89.2 Detailed Description . . . 416
3.89.3 Field Documentation . . . 416
3.89.3.1 auth_plain . . . 416
3.89.3.2 auth_plain_len . . . 417
3.89.3.3 body . . . 417
3.89.3.4 body_len . . . 417
3.89.3.5 body_sent . . . 417
3.89.3.6 callback_arg . . . 417
3.89.3.7 callback_fn . . . 417
3.89.3.8 from . . . 417
3.89.3.9 from_len . . . 417
3.89.3.10 pass . . . 417
3.89.3.11 state . . . 418
3.89.3.12 subject . . . 418
3.89.3.13 subject_len . . . 418
3.89.3.14 timer . . . 418
3.89.3.15 to . . . 418
3.89.3.16 to_len . . . 418
3.89.3.17 tx_buf . . . 418
3.89.3.18 username . . . 418
3.90 snmp_leaf_node Struct Reference . . . 418
3.90.1 Data Fields . . . 419
3.90.2 Detailed Description . . . 419
3.90.3 Field Documentation . . . 419
3.90.3.1 node . . . 419
3.91 snmp_mib Struct Reference . . . 419
3.91.1 Detailed Description . . . 419
3.92 snmp_next_oid_state Struct Reference . . . 419
3.92.1 Detailed Description . . . 419
3.93 snmp_node Struct Reference . . . 419
3.93.1 Data Fields . . . 419
3.93.2 Detailed Description . . . 419
3.93.3 Field Documentation . . . 420

3.93.3.1 node_type . . . 420
3.93.3.2 oid . . . 420
3.94 snmp_node_instance Struct Reference . . . 420
3.94.1 Data Fields . . . 420
3.94.2 Detailed Description . . . 420
3.94.3 Field Documentation . . . 420
3.94.3.1 access . . . 420
3.94.3.2 asn1_type . . . 420
3.94.3.3 get_value . . . 421
3.94.3.4 instance_oid . . . 421
3.94.3.5 node . . . 421
3.94.3.6 reference . . . 421
3.94.3.7 reference_len . . . 421
3.94.3.8 release_instance . . . 421
3.94.3.9 set_test . . . 421
3.94.3.10 set_value . . . 421
3.95 snmp_obj_id Struct Reference . . . 421
3.95.1 Detailed Description . . . 422
3.96 snmp_oid_range Struct Reference . . . 422
3.96.1 Detailed Description . . . 422
3.97 snmp_scalar_array_node Struct Reference . . . 422
3.97.1 Data Fields . . . 422
3.97.2 Detailed Description . . . 422
3.97.3 Field Documentation . . . 422
3.97.3.1 node . . . 422
3.98 snmp_scalar_array_node_def Struct Reference . . . 422
3.98.1 Detailed Description . . . 422
3.99 snmp_scalar_node Struct Reference . . . 423
3.99.1 Data Fields . . . 423
3.99.2 Detailed Description . . . 423
3.99.3 Field Documentation . . . 423
3.99.3.1 node . . . 423
3.100 snmp_table_col_def Struct Reference . . . 423
3.100.1 Detailed Description . . . 423
3.101 snmp_table_node Struct Reference . . . 423
3.101.1 Data Fields . . . 423
3.101.2 Detailed Description . . . 423
3.101.3 Field Documentation . . . 424
3.101.3.1 get_value . . . 424

3.101.3.2 node . . . 424
3.101.3.3 set_test . . . 424
3.101.3.4 set_value . . . 424
3.102 snmp_table_simple_node Struct Reference . . . 424
3.102.1 Detailed Description . . . 424
3.103 snmp_threadsync_instance Struct Reference . . . 424
3.103.1 Detailed Description . . . 424
3.104 snmp_threadsync_node Struct Reference . . . 424
3.104.1 Detailed Description . . . 425
3.105 snmp_tree_node Struct Reference . . . 425
3.105.1 Data Fields . . . 425
3.105.2 Detailed Description . . . 425
3.105.3 Field Documentation . . . 425
3.105.3.1 node . . . 425
3.106 snmp_varbind Struct Reference . . . 425
3.106.1 Data Fields . . . 425
3.106.2 Detailed Description . . . 425
3.106.3 Field Documentation . . . 425
3.106.3.1 next . . . 425
3.106.3.2 oid . . . 426
3.106.3.3 type . . . 426
3.106.3.4 value . . . 426
3.106.3.5 value_len . . . 426
3.107 snmp_varbind_len Struct Reference . . . 426
3.107.1 Detailed Description . . . 426
3.108 snmp_variant_value Union Reference . . . 426
3.108.1 Detailed Description . . . 426
3.109 sntp_msg Struct Reference . . . 426
3.109.1 Detailed Description . . . 426
3.110 sntp_server Struct Reference . . . 427
3.110.1 Data Fields . . . 427
3.110.2 Detailed Description . . . 427
3.110.3 Field Documentation . . . 427
3.110.3.1 reachability . . . 427
3.111 sntp_time Struct Reference . . . 427
3.111.1 Detailed Description . . . 427
3.112 sntp_timestamps Struct Reference . . . 427
3.112.1 Detailed Description . . . 427
3.113 sockaddr_aligned Union Reference . . . 427

3.113.1 Detailed Description 427
3.114 stats_ Struct Reference 427
3.114.1 Data Fields 428
3.114.2 Detailed Description 428
3.114.3 Field Documentation 428
3.114.3.1 etharp 428
3.114.3.2 icmp 428
3.114.3.3 icmp6 428
3.114.3.4 igmp 429
3.114.3.5 ip 429
3.114.3.6 ip6 429
3.114.3.7 ip6_frag 429
3.114.3.8 ip_frag 429
3.114.3.9 link 429
3.114.3.10 mem 429
3.114.3.11 memp 429
3.114.3.12 mib2 429
3.114.3.13 mld6 430
3.114.3.14 nd6 430
3.114.3.15 sys 430
3.114.3.16 tcp 430
3.114.3.17 udp 430
3.115 stats_igmp Struct Reference 430
3.115.1 Detailed Description 430
3.116 stats_mem Struct Reference 430
3.116.1 Detailed Description 430
3.117 stats_mib2 Struct Reference 431
3.117.1 Detailed Description 431
3.118 stats_mib2_netif_ctrs Struct Reference 431
3.118.1 Data Fields 431
3.118.2 Detailed Description 431
3.118.3 Field Documentation 431
3.118.3.1 ifindiscards 431
3.118.3.2 ifinerrors 431
3.118.3.3 ifinnucastpkts 432
3.118.3.4 ifinoctets 432
3.118.3.5 ifinucastpkts 432
3.118.3.6 ifinunknownprotos 432
3.118.3.7 ifoutdiscards 432

3.118.3.8 ifouterrors 432
3.118.3.9 ifoutnucastpkts 432
3.118.3.10 ifoutoctets 432
3.118.3.11 ifoutucastpkts 433
3.119 stats_proto Struct Reference 433
3.119.1 Detailed Description 433
3.120 stats_sys Struct Reference 433
3.120.1 Detailed Description 433
3.121 stats_syselem Struct Reference 433
3.121.1 Detailed Description 433
3.122 netif_ext_callback_args_t::status_changed_s Struct Reference 433
3.122.1 Data Fields 433
3.122.2 Detailed Description 433
3.122.3 Field Documentation 434
3.122.3.1 state 434
3.123 tCGI Struct Reference 434
3.123.1 Detailed Description 434
3.124 tcp_ext_arg_callbacks Struct Reference 434
3.124.1 Data Fields 434
3.124.2 Detailed Description 434
3.124.3 Field Documentation 434
3.124.3.1 destroy 434
3.124.3.2 passive_open 434
3.125 tcp_pcb Struct Reference 434
3.125.1 Data Fields 435
3.125.2 Detailed Description 435
3.125.3 Field Documentation 435
3.125.3.1 local_ip 435
3.125.3.2 next 435
3.126 tcp_pcb_listen Struct Reference 435
3.126.1 Data Fields 435
3.126.2 Detailed Description 435
3.126.3 Field Documentation 435
3.126.3.1 local_ip 435
3.126.3.2 next 435
3.127 tftp_context Struct Reference 436
3.127.1 Data Fields 436
3.127.2 Detailed Description 436
3.127.3 Field Documentation 436

3.127.3.1 close . . . 436
3.127.3.2 error . . . 436
3.127.3.3 open . . . 436
3.127.3.4 read . . . 437
3.127.3.5 write . . . 437
3.128 threadsync_data Struct Reference . . . 437
3.128.1 Detailed Description . . . 437
3.129 udp_pcb Struct Reference . . . 437
3.129.1 Data Fields . . . 438
3.129.2 Detailed Description . . . 438
3.129.3 Field Documentation . . . 438
3.129.3.1 chksum_len_rx . . . 438
3.129.3.2 local_ip . . . 438
3.129.3.3 local_port . . . 438
3.129.3.4 mcast_ifindex . . . 438
3.129.3.5 mcast_ip4 . . . 438
3.129.3.6 mcast_ttl . . . 438
3.129.3.7 recv . . . 439
3.129.3.8 recv_arg . . . 439
3.130 zepif_init Struct Reference . . . 439
3.130.1 Data Fields . . . 439
3.130.2 Detailed Description . . . 439
3.130.3 Field Documentation . . . 439
3.130.3.1 addr . . . 439
3.130.3.2 zep_dst_ip_addr . . . 439
3.130.3.3 zep_dst_udp_port . . . 439
3.130.3.4 zep_netif . . . 440
3.130.3.5 zep_src_ip_addr . . . 440
3.130.3.6 zep_src_udp_port . . . 440

4 File Documentation **441**
4.1 src/api/api_lib.c File Reference . . . 441
4.1.1 Functions . . . 441
4.1.2 Detailed Description . . . 442
4.1.3 Function Documentation . . . 442
4.1.3.1 netconn_getaddr() . . . 442
4.1.3.2 netconn_new_with_proto_and_callback() . . . 442
4.1.3.3 netconn_recv_udp_raw_netbuf() . . . 443
4.1.3.4 netconn_recv_udp_raw_netbuf_flags() . . . 443

4.1.3.5 netconn_write_vectors_partly() . . . 443
4.2 src/api/api_msg.c File Reference . . . 444
4.2.1 Functions . . . 444
4.2.2 Detailed Description . . . 444
4.2.3 Function Documentation . . . 444
4.2.3.1 lwip_netconn_do_accepted() . . . 444
4.2.3.2 lwip_netconn_do_bind() . . . 445
4.2.3.3 lwip_netconn_do_bind_if() . . . 445
4.2.3.4 lwip_netconn_do_close() . . . 445
4.2.3.5 lwip_netconn_do_connect() . . . 445
4.2.3.6 lwip_netconn_do_delconn() . . . 445
4.2.3.7 lwip_netconn_do_disconnect() . . . 445
4.2.3.8 lwip_netconn_do_getaddr() . . . 446
4.2.3.9 lwip_netconn_do_gethostbyname() . . . 446
4.2.3.10 lwip_netconn_do_join_leave_group() . . . 446
4.2.3.11 lwip_netconn_do_join_leave_group_netif() . . . 446
4.2.3.12 lwip_netconn_do_listen() . . . 446
4.2.3.13 lwip_netconn_do_newconn() . . . 446
4.2.3.14 lwip_netconn_do_recv() . . . 447
4.2.3.15 lwip_netconn_do_send() . . . 447
4.2.3.16 lwip_netconn_do_write() . . . 447
4.2.3.17 netconn_alloc() . . . 447
4.2.3.18 netconn_free() . . . 447
4.3 src/api/err.c File Reference . . . 447
4.3.1 Detailed Description . . . 448
4.4 src/api/if_api.c File Reference . . . 448
4.4.1 Functions . . . 448
4.4.2 Detailed Description . . . 448
4.5 src/api/netbuf.c File Reference . . . 448
4.5.1 Functions . . . 448
4.5.2 Detailed Description . . . 449
4.6 src/api/netdb.c File Reference . . . 449
4.6.1 Data Structures . . . 449
4.6.2 Macros . . . 449
4.6.3 Functions . . . 449
4.6.4 Variables . . . 449
4.6.5 Detailed Description . . . 449
4.6.6 Macro Definition Documentation . . . 450
4.6.6.1 LWIP_DNS_API_HOSTENT_STORAGE . . . 450

4.6.7 Function Documentation . . . 450
4.6.7.1 lwip_freeaddrinfo() . . . 450
4.6.7.2 lwip_getaddrinfo() . . . 450
4.6.7.3 lwip_gethostbyname() . . . 450
4.6.7.4 lwip_gethostbyname_r() . . . 451
4.6.8 Variable Documentation . . . 451
4.6.8.1 h_errno . . . 451
4.7 src/api/netifapi.c File Reference . . . 451
4.7.1 Functions . . . 451
4.7.2 Detailed Description . . . 452
4.7.3 Function Documentation . . . 452
4.7.3.1 netifapi_arp_add() . . . 452
4.7.3.2 netifapi_arp_remove() . . . 452
4.7.3.3 netifapi_netif_common() . . . 452
4.8 src/api/sockets.c File Reference . . . 453
4.8.1 Data Structures . . . 453
4.8.2 Enumerations . . . 453
4.8.3 Functions . . . 453
4.8.4 Detailed Description . . . 453
4.8.5 Enumeration Type Documentation . . . 453
4.8.5.1 lwip_pollscan_opts . . . 453
4.8.6 Function Documentation . . . 454
4.8.6.1 lwip_fcntl() . . . 454
4.8.6.2 lwip_listen() . . . 454
4.8.6.3 lwip_shutdown() . . . 454
4.8.6.4 lwip_socket_thread_cleanup() . . . 454
4.8.6.5 lwip_socket_thread_init() . . . 454
4.9 src/api/tcpip.c File Reference . . . 454
4.9.1 Functions . . . 454
4.9.2 Variables . . . 455
4.9.3 Detailed Description . . . 455
4.9.4 Function Documentation . . . 455
4.9.4.1 mem_free_callback() . . . 455
4.9.4.2 pbuf_free_callback() . . . 455
4.9.4.3 tcpip_api_call() . . . 456
4.9.4.4 tcpip_callback_wait() . . . 456
4.9.4.5 tcpip_inpkt() . . . 456
4.9.4.6 tcpip_send_msg_wait_sem() . . . 456
4.9.5 Variable Documentation . . . 457

4.9.5.1 lock_tcpip_core 457
4.10 src/apps/altcp_tls/altcp_tls_mbedtls.c File Reference 457
4.10.1 Detailed Description 457
4.11 src/apps/altcp_tls/altcp_tls_mbedtls_mem.c File Reference 457
4.11.1 Detailed Description 457
4.12 src/apps/altcp_tls/altcp_tls_mbedtls_mem.h File Reference 457
4.12.1 Detailed Description 457
4.13 src/apps/altcp_tls/altcp_tls_mbedtls_structs.h File Reference 458
4.13.1 Detailed Description 458
4.14 src/apps/http/altcp_proxyconnect.c File Reference 458
4.14.1 Macros 458
4.14.2 Functions 458
4.14.3 Detailed Description 458
4.14.4 Macro Definition Documentation 458
4.14.4.1 ALTCP_PROXYCONNECT_CLIENT_AGENT 458
4.14.5 Function Documentation 458
4.14.5.1 altcp_proxyconnect_alloc() 458
4.14.5.2 altcp_proxyconnect_new() 459
4.14.5.3 altcp_proxyconnect_new_tcp() 459
4.14.5.4 altcp_proxyconnect_tls_alloc() 459
4.15 src/apps/http/http_client.c File Reference 459
4.15.1 Macros 459
4.15.2 Functions 460
4.15.3 Detailed Description 460
4.15.4 Macro Definition Documentation 460
4.15.4.1 HTTPC_CLIENT_AGENT 460
4.15.4.2 HTTPC_DEBUG 460
4.15.4.3 HTTPC_DEBUG_REQUEST 460
4.16 src/apps/http/httpd.c File Reference 460
4.16.1 Macros 460
4.16.2 Functions 461
4.16.3 Detailed Description 461
4.16.4 Macro Definition Documentation 461
4.16.4.1 HTTP_IS_DATA_VOLATILE 461
4.16.4.2 HTTP_IS_HDR_VOLATILE 461
4.16.4.3 MIN_REQ_LEN 461
4.17 src/apps/lwiperf/lwiperf.c File Reference 461
4.17.1 Data Structures 461
4.17.2 Macros 461

4.17.3 Typedefs . . . 462
4.17.4 Functions . . . 462
4.17.5 Detailed Description . . . 462
4.17.6 Macro Definition Documentation . . . 462
4.17.6.1 LWIPERF_CHECK_RX_DATA . . . 462
4.17.6.2 LWIPERF_SERVER_IP_TYPE . . . 462
4.17.6.3 LWIPERF_TCP_MAX_IDLE_SEC . . . 462
4.17.7 Typedef Documentation . . . 462
4.17.7.1 lwiperf_settings_t . . . 462
4.17.7.2 lwiperf_state_tcp_t . . . 462
4.18 src/apps/mdns/mdns.c File Reference . . . 463
4.18.1 Data Structures . . . 463
4.18.2 Macros . . . 463
4.18.3 Functions . . . 463
4.18.4 Detailed Description . . . 464
4.18.5 Macro Definition Documentation . . . 464
4.18.5.1 MDNS_PROBE_COUNT . . . 464
4.18.5.2 MDNS_RESPONSE_DELAY_MAX . . . 464
4.18.6 Function Documentation . . . 464
4.18.6.1 get_mdns_pcb() . . . 464
4.18.6.2 mdns_resp_register_name_result_cb() . . . 464
4.18.6.3 netif_mdns_data() . . . 464
4.19 src/apps/mdns/mdns_domain.c File Reference . . . 464
4.19.1 Functions . . . 465
4.19.2 Detailed Description . . . 465
4.19.3 Function Documentation . . . 465
4.19.3.1 mdns_build_dnssd_domain() . . . 465
4.19.3.2 mdns_build_host_domain() . . . 465
4.19.3.3 mdns_build_request_domain() . . . 466
4.19.3.4 mdns_build_reverse_v4_domain() . . . 466
4.19.3.5 mdns_build_reverse_v6_domain() . . . 466
4.19.3.6 mdns_build_service_domain() . . . 466
4.19.3.7 mdns_compress_domain() . . . 467
4.19.3.8 mdns_domain_add_domain() . . . 467
4.19.3.9 mdns_domain_add_label() . . . 467
4.19.3.10 mdns_domain_add_string() . . . 467
4.19.3.11 mdns_domain_debug_print() . . . 468
4.19.3.12 mdns_domain_eq() . . . 468
4.19.3.13 mdns_readname() . . . 468

4.19.3.14 mdns_write_domain() . . . 468
4.20 src/apps/mdns/mdns_out.c File Reference . . . 468
4.20.1 Functions . . . 469
4.20.2 Detailed Description . . . 469
4.20.3 Function Documentation . . . 469
4.20.3.1 mdns_create_outpacket() . . . 469
4.20.3.2 mdns_multicast_probe_timeout_reset_ipv4() . . . 470
4.20.3.3 mdns_multicast_probe_timeout_reset_ipv6() . . . 470
4.20.3.4 mdns_multicast_timeout_25ttl_reset_ipv4() . . . 470
4.20.3.5 mdns_multicast_timeout_25ttl_reset_ipv6() . . . 470
4.20.3.6 mdns_multicast_timeout_reset_ipv4() . . . 470
4.20.3.7 mdns_multicast_timeout_reset_ipv6() . . . 470
4.20.3.8 mdns_prepare_txtdata() . . . 471
4.20.3.9 mdns_send_multicast_msg_delayed_ipv4() . . . 471
4.20.3.10 mdns_send_multicast_msg_delayed_ipv6() . . . 471
4.20.3.11 mdns_send_outpacket() . . . 471
4.20.3.12 mdns_send_request() . . . 471
4.20.3.13 mdns_send_unicast_msg_delayed_ipv4() . . . 471
4.20.3.14 mdns_send_unicast_msg_delayed_ipv6() . . . 472
4.20.3.15 mdns_set_timeout() . . . 472
4.20.3.16 mdns_start_multicast_timeouts_ipv4() . . . 472
4.20.3.17 mdns_start_multicast_timeouts_ipv6() . . . 472
4.21 src/apps/mqtt/mqtt.c File Reference . . . 473
4.21.1 Macros . . . 473
4.21.2 Enumerations . . . 473
4.21.3 Functions . . . 473
4.21.4 Detailed Description . . . 474
4.21.5 Macro Definition Documentation . . . 474
4.21.5.1 MQTT_CTL_PACKET_TYPE . . . 474
4.21.5.2 MQTT_DEBUG . . . 474
4.21.5.3 mqtt_ringbuf_free . . . 474
4.21.5.4 mqtt_ringbuf_linear_read_length . . . 474
4.21.6 Enumeration Type Documentation . . . 474
4.21.6.1 anonymous enum . . . 474
4.21.6.2 mqtt_connect_flag . . . 474
4.21.6.3 mqtt_message_type . . . 475
4.22 src/apps/netbiosns/netbiosns.c File Reference . . . 475
4.22.1 Data Structures . . . 475
4.22.2 Macros . . . 475

4.22.3 Functions 475
4.22.4 Detailed Description 475
4.22.5 Macro Definition Documentation 475
4.22.5.1 NETB_HFLAG_RESPONSE 475
4.22.5.2 NETB_NFLAG_UNIQUE 476
4.22.5.3 NETBIOS_NAME_LEN 476
4.22.5.4 NETBIOS_NAME_TTL 476
4.23 src/apps/smtp/smtp.c File Reference 476
4.23.1 Data Structures 476
4.23.2 Macros 476
4.23.3 Enumerations 476
4.23.4 Functions 476
4.23.5 Detailed Description 477
4.23.6 Macro Definition Documentation 477
4.23.6.1 SMTP_POLL_INTERVAL 477
4.23.6.2 SMTP_TIMEOUT 477
4.23.6.3 SMTP_TIMEOUT_DATABLOCK 477
4.23.6.4 SMTP_TIMEOUT_DATATERM 477
4.23.7 Enumeration Type Documentation 477
4.23.7.1 smtp_session_state 477
4.24 src/apps/snmp/snmp_asn1.c File Reference 477
4.24.1 Functions 477
4.24.2 Detailed Description 478
4.24.3 Function Documentation 478
4.24.3.1 snmp_ans1_enc_tlv() 478
4.24.3.2 snmp_asn1_dec_oid() 478
4.24.3.3 snmp_asn1_dec_raw() 479
4.24.3.4 snmp_asn1_dec_s32t() 479
4.24.3.5 snmp_asn1_dec_tlv() 479
4.24.3.6 snmp_asn1_dec_u32t() 480
4.24.3.7 snmp_asn1_enc_length_cnt() 480
4.24.3.8 snmp_asn1_enc_oid() 480
4.24.3.9 snmp_asn1_enc_oid_cnt() 480
4.24.3.10 snmp_asn1_enc_raw() 481
4.24.3.11 snmp_asn1_enc_s32t() 481
4.24.3.12 snmp_asn1_enc_s32t_cnt() 481
4.24.3.13 snmp_asn1_enc_u32t() 482
4.24.3.14 snmp_asn1_enc_u32t_cnt() 482
4.25 src/apps/snmp/snmp_asn1.h File Reference 482

4.25.1 Functions . . . 482
4.25.2 Detailed Description . . . 483
4.25.3 Function Documentation . . . 483
4.25.3.1 snmp_ans1_enc_tlv() . . . 483
4.25.3.2 snmp_asn1_dec_oid() . . . 483
4.25.3.3 snmp_asn1_dec_raw() . . . 484
4.25.3.4 snmp_asn1_dec_s32t() . . . 484
4.25.3.5 snmp_asn1_dec_tlv() . . . 484
4.25.3.6 snmp_asn1_dec_u32t() . . . 485
4.25.3.7 snmp_asn1_enc_length_cnt() . . . 485
4.25.3.8 snmp_asn1_enc_oid() . . . 485
4.25.3.9 snmp_asn1_enc_oid_cnt() . . . 485
4.25.3.10 snmp_asn1_enc_raw() . . . 486
4.25.3.11 snmp_asn1_enc_s32t() . . . 486
4.25.3.12 snmp_asn1_enc_s32t_cnt() . . . 486
4.25.3.13 snmp_asn1_enc_u32t() . . . 487
4.25.3.14 snmp_asn1_enc_u32t_cnt() . . . 487
4.26 src/apps/snmp/snmp_core.c File Reference . . . 487
4.26.1 Functions . . . 487
4.26.2 Detailed Description . . . 488
4.26.3 Function Documentation . . . 488
4.26.3.1 netif_to_num() . . . 488
4.26.3.2 snmp_decode_bits() . . . 489
4.26.3.3 snmp_encode_bits() . . . 489
4.26.3.4 snmp_ip4_to_oid() . . . 489
4.26.3.5 snmp_ip6_to_oid() . . . 490
4.26.3.6 snmp_ip_port_to_oid() . . . 490
4.26.3.7 snmp_ip_to_oid() . . . 490
4.26.3.8 snmp_mib_tree_resolve_exact() . . . 490
4.26.3.9 snmp_next_oid_check() . . . 490
4.26.3.10 snmp_next_oid_init() . . . 491
4.26.3.11 snmp_next_oid_precheck() . . . 491
4.26.3.12 snmp_oid_append() . . . 491
4.26.3.13 snmp_oid_assign() . . . 491
4.26.3.14 snmp_oid_combine() . . . 491
4.26.3.15 snmp_oid_compare() . . . 492
4.26.3.16 snmp_oid_equal() . . . 492
4.26.3.17 snmp_oid_in_range() . . . 492
4.26.3.18 snmp_oid_prefix() . . . 492

4.26.3.19 snmp_oid_to_ip() 492
4.26.3.20 snmp_oid_to_ip4() 493
4.26.3.21 snmp_oid_to_ip6() 493
4.26.3.22 snmp_oid_to_ip_port() 493
4.27 src/apps/snmp/snmp_mib2.c File Reference 493
4.27.1 Detailed Description 493
4.28 src/apps/snmp/snmp_mib2_icmp.c File Reference 493
4.28.1 Detailed Description 494
4.29 src/apps/snmp/snmp_mib2_interfaces.c File Reference 494
4.29.1 Detailed Description 494
4.30 src/apps/snmp/snmp_mib2_ip.c File Reference 494
4.30.1 Detailed Description 494
4.31 src/apps/snmp/snmp_mib2_snmp.c File Reference 494
4.31.1 Detailed Description 494
4.32 src/apps/snmp/snmp_mib2_system.c File Reference 494
4.32.1 Functions 495
4.32.2 Detailed Description 495
4.33 src/apps/snmp/snmp_mib2_tcp.c File Reference 495
4.33.1 Detailed Description 495
4.34 src/apps/snmp/snmp_mib2_udp.c File Reference 495
4.34.1 Detailed Description 495
4.35 src/apps/snmp/snmp_msg.c File Reference 495
4.35.1 Functions 496
4.35.2 Variables 496
4.35.3 Detailed Description 496
4.35.4 Function Documentation 496
4.35.4.1 snmp_varbind_length() 496
4.35.5 Variable Documentation 496
4.35.5.1 snmp_community 496
4.35.5.2 snmp_community_trap 496
4.35.5.3 snmp_community_write 496
4.36 src/apps/snmp/snmp_msg.h File Reference 497
4.36.1 Data Structures 497
4.36.2 Functions 497
4.36.3 Variables 497
4.36.4 Detailed Description 497
4.36.5 Function Documentation 497
4.36.5.1 snmp_varbind_length() 497
4.36.6 Variable Documentation 497

4.36.6.1 snmp_community . . . 497
4.36.6.2 snmp_community_write . . . 497
4.36.6.3 snmp_traps_handle . . . 498
4.37 src/apps/snmp/snmp_netconn.c File Reference . . . 498
4.37.1 Functions . . . 498
4.37.2 Detailed Description . . . 498
4.37.3 Function Documentation . . . 498
4.37.3.1 snmp_init() . . . 498
4.38 src/apps/snmp/snmp_pbuf_stream.c File Reference . . . 498
4.38.1 Detailed Description . . . 498
4.39 src/apps/snmp/snmp_pbuf_stream.h File Reference . . . 498
4.39.1 Detailed Description . . . 498
4.40 src/apps/snmp/snmp_raw.c File Reference . . . 499
4.40.1 Functions . . . 499
4.40.2 Detailed Description . . . 499
4.41 src/apps/snmp/snmp_scalar.c File Reference . . . 499
4.41.1 Detailed Description . . . 499
4.42 src/apps/snmp/snmp_table.c File Reference . . . 499
4.42.1 Detailed Description . . . 499
4.43 src/apps/snmp/snmp_threadsync.c File Reference . . . 499
4.43.1 Functions . . . 499
4.43.2 Detailed Description . . . 499
4.43.3 Function Documentation . . . 500
4.43.3.1 snmp_threadsync_init() . . . 500
4.44 src/apps/snmp/snmp_traps.c File Reference . . . 500
4.44.1 Functions . . . 500
4.44.2 Variables . . . 500
4.44.3 Detailed Description . . . 500
4.44.4 Variable Documentation . . . 501
4.44.4.1 snmp_community_trap . . . 501
4.44.4.2 snmp_traps_handle . . . 501
4.45 src/apps/snmp/snmpv3.c File Reference . . . 501
4.45.1 Detailed Description . . . 501
4.46 src/apps/snmp/snmpv3_mbedtls.c File Reference . . . 501
4.46.1 Detailed Description . . . 501
4.47 src/apps/snmp/snmpv3_priv.h File Reference . . . 501
4.47.1 Detailed Description . . . 501
4.48 src/apps/sntp/sntp.c File Reference . . . 501
4.48.1 Data Structures . . . 502

4.48.2 Macros 502
4.48.3 Functions 502
4.48.4 Detailed Description 502
4.48.5 Macro Definition Documentation 502
4.48.5.1 SNTP_FRAC_TO_US 502
4.49 src/apps/tftp/tftp.c File Reference 502
4.49.1 Functions 503
4.49.2 Detailed Description 503
4.49.3 Function Documentation 503
4.49.3.1 tftp_init_common() 503
4.50 src/core/altcp.c File Reference 503
4.50.1 Functions 503
4.50.2 Function Documentation 504
4.50.2.1 altcp_alloc() 504
4.50.2.2 altcp_free() 504
4.51 src/core/altcp_alloc.c File Reference 504
4.51.1 Functions 504
4.51.2 Detailed Description 505
4.52 src/core/altcp_tcp.c File Reference 505
4.52.1 Functions 505
4.52.2 Detailed Description 505
4.52.3 Function Documentation 505
4.52.3.1 altcp_tcp_alloc() 505
4.53 src/core/def.c File Reference 505
4.53.1 Functions 505
4.53.2 Detailed Description 506
4.53.3 Function Documentation 506
4.53.3.1 lwip_htonl() 506
4.53.3.2 lwip_htons() 506
4.54 src/core/dns.c File Reference 506
4.54.1 Data Structures 507
4.54.2 Macros 507
4.54.3 Functions 507
4.54.4 Detailed Description 507
4.54.5 Macro Definition Documentation 507
4.54.5.1 DNS_MAX_TTL 507
4.54.5.2 DNS_PORT_ALLOWED 507
4.54.5.3 DNS_RAND_TXID 507
4.54.6 Function Documentation 508

4.54.6.1 dns_init() . . . 508
4.54.6.2 dns_tmr() . . . 508
4.55 src/core/inet_chksum.c File Reference . . . 508
4.55.1 Functions . . . 508
4.55.2 Detailed Description . . . 508
4.55.3 Function Documentation . . . 508
4.55.3.1 inet_chksum_pbuf() . . . 508
4.55.3.2 ip6_chksum_pseudo() . . . 509
4.55.3.3 ip6_chksum_pseudo_partial() . . . 509
4.56 src/core/init.c File Reference . . . 509
4.56.1 Functions . . . 509
4.56.2 Detailed Description . . . 509
4.57 src/core/ip.c File Reference . . . 510
4.57.1 Functions . . . 510
4.57.2 Variables . . . 510
4.57.3 Detailed Description . . . 510
4.57.4 Variable Documentation . . . 510
4.57.4.1 ip_data . . . 510
4.58 src/core/ipv4/acd.c File Reference . . . 510
4.58.1 Functions . . . 510
4.58.2 Detailed Description . . . 511
4.58.3 Function Documentation . . . 511
4.58.3.1 acd_arp_reply() . . . 511
4.58.3.2 acd_tmr() . . . 511
4.59 src/core/ipv4/autoip.c File Reference . . . 511
4.59.1 Macros . . . 511
4.59.2 Functions . . . 511
4.59.3 Detailed Description . . . 512
4.59.4 Macro Definition Documentation . . . 512
4.59.4.1 LWIP_AUTOIP_CREATE_SEED_ADDR . . . 512
4.59.5 Function Documentation . . . 512
4.59.5.1 autoip_network_changed_link_down() . . . 512
4.59.5.2 autoip_network_changed_link_up() . . . 512
4.59.5.3 autoip_supplied_address() . . . 512
4.60 src/core/ipv4/dhcp.c File Reference . . . 512
4.60.1 Macros . . . 513
4.60.2 Enumerations . . . 513
4.60.3 Functions . . . 513
4.60.4 Detailed Description . . . 513

4.60.5 Macro Definition Documentation . . . 513
4.60.5.1 DHCP_ADD_EXTRA_REQUEST_OPTIONS . . . 513
4.60.5.2 DHCP_CREATE_RAND_XID . . . 514
4.60.5.3 DHCP_MAX_MSG_LEN . . . 514
4.60.5.4 DHCP_MIN_REPLY_LEN . . . 514
4.60.5.5 SET_TIMEOUT_FROM_OFFERED . . . 514
4.60.6 Enumeration Type Documentation . . . 514
4.60.6.1 dhcp_option_idx . . . 514
4.60.7 Function Documentation . . . 515
4.60.7.1 dhcp_coarse_tmr() . . . 515
4.60.7.2 dhcp_fine_tmr() . . . 515
4.60.7.3 dhcp_network_changed_link_up() . . . 515
4.60.7.4 dhcp_supplied_address() . . . 515
4.61 src/core/ipv4/etharp.c File Reference . . . 515
4.61.1 Macros . . . 515
4.61.2 Enumerations . . . 515
4.61.3 Functions . . . 516
4.61.4 Detailed Description . . . 516
4.61.5 Macro Definition Documentation . . . 516
4.61.5.1 ARP_AGE_REREQUEST_USED_UNICAST . . . 516
4.61.5.2 ARP_MAXPENDING . . . 516
4.61.5.3 ETHARP_FLAG_TRY_HARD . . . 516
4.61.6 Enumeration Type Documentation . . . 516
4.61.6.1 etharp_state . . . 516
4.61.7 Function Documentation . . . 517
4.61.7.1 etharp_acd_announce() . . . 517
4.61.7.2 etharp_acd_probe() . . . 517
4.61.7.3 etharp_cleanup_netif() . . . 517
4.61.7.4 etharp_find_addr() . . . 517
4.61.7.5 etharp_get_entry() . . . 518
4.61.7.6 etharp_input() . . . 518
4.61.7.7 etharp_output() . . . 518
4.61.7.8 etharp_query() . . . 519
4.61.7.9 etharp_request() . . . 519
4.61.7.10 etharp_tmr() . . . 520
4.62 src/core/ipv4/icmp.c File Reference . . . 520
4.62.1 Macros . . . 520
4.62.2 Functions . . . 520
4.62.3 Detailed Description . . . 520

4.62.4 Macro Definition Documentation . . . 520
4.62.4.1 LWIP_ICMP_ECHO_CHECK_INPUT_PBUF_LEN . . . 520
4.62.5 Function Documentation . . . 520
4.62.5.1 icmp_dest_unreach() . . . 520
4.62.5.2 icmp_input() . . . 521
4.62.5.3 icmp_time_exceeded() . . . 521
4.63 src/core/ipv4/igmp.c File Reference . . . 521
4.63.1 Functions . . . 521
4.63.2 Detailed Description . . . 522
4.63.3 Function Documentation . . . 522
4.63.3.1 igmp_init() . . . 522
4.63.3.2 igmp_input() . . . 522
4.63.3.3 igmp_lookfor_group() . . . 522
4.63.3.4 igmp_report_groups() . . . 522
4.63.3.5 igmp_start() . . . 522
4.63.3.6 igmp_stop() . . . 523
4.63.3.7 igmp_tmr() . . . 523
4.64 src/core/ipv4/ip4.c File Reference . . . 523
4.64.1 Macros . . . 523
4.64.2 Functions . . . 523
4.64.3 Detailed Description . . . 524
4.64.4 Macro Definition Documentation . . . 524
4.64.4.1 IP_ACCEPT_LINK_LAYER_ADDRESSED_PORT . . . 524
4.64.4.2 LWIP_INLINE_IP_CHKSUM . . . 524
4.64.5 Function Documentation . . . 524
4.64.5.1 ip4_input() . . . 524
4.64.5.2 ip4_output() . . . 524
4.64.5.3 ip4_output_if() . . . 525
4.64.5.4 ip4_output_if_opt() . . . 525
4.64.5.5 ip4_output_if_opt_src() . . . 525
4.64.5.6 ip4_output_if_src() . . . 526
4.64.5.7 ip4_route() . . . 526
4.64.5.8 ip4_route_src() . . . 526
4.65 src/core/ipv4/ip4_addr.c File Reference . . . 526
4.65.1 Functions . . . 526
4.65.2 Detailed Description . . . 526
4.65.3 Function Documentation . . . 527
4.65.3.1 ip4_addr_isbroadcast_u32() . . . 527
4.65.3.2 ip4_addr_netmask_valid() . . . 527

4.65.3.3 ip4addr_aton() . . . 527
4.65.3.4 ip4addr_ntoa() . . . 527
4.65.3.5 ip4addr_ntoa_r() . . . 528
4.65.3.6 ipaddr_addr() . . . 528
4.66 src/core/ipv4/ip4_frag.c File Reference . . . 528
4.66.1 Data Structures . . . 528
4.66.2 Macros . . . 528
4.66.3 Functions . . . 528
4.66.4 Detailed Description . . . 529
4.66.5 Macro Definition Documentation . . . 529
4.66.5.1 IP_REASS_CHECK_OVERLAP . . . 529
4.66.5.2 IP_REASS_FREE_OLDEST . . . 529
4.66.6 Function Documentation . . . 529
4.66.6.1 ip4_frag() . . . 529
4.66.6.2 ip4_reass() . . . 529
4.66.6.3 ip_reass_tmr() . . . 530
4.67 src/core/ipv6/dhcp6.c File Reference . . . 530
4.67.1 Enumerations . . . 530
4.67.2 Functions . . . 530
4.67.3 Variables . . . 530
4.67.4 Enumeration Type Documentation . . . 530
4.67.4.1 dhcp6_option_idx . . . 530
4.67.5 Function Documentation . . . 531
4.67.5.1 dhcp6_nd6_ra_trigger() . . . 531
4.67.5.2 dhcp6_tmr() . . . 531
4.67.6 Variable Documentation . . . 531
4.67.6.1 dhcp6_rx_options . . . 531
4.68 src/core/ipv6/ethip6.c File Reference . . . 531
4.68.1 Functions . . . 531
4.68.2 Detailed Description . . . 531
4.68.3 Function Documentation . . . 531
4.68.3.1 ethip6_output() . . . 531
4.69 src/core/ipv6/icmp6.c File Reference . . . 532
4.69.1 Functions . . . 532
4.69.2 Detailed Description . . . 532
4.69.3 Function Documentation . . . 532
4.69.3.1 icmp6_dest_unreach() . . . 532
4.69.3.2 icmp6_input() . . . 532
4.69.3.3 icmp6_packet_too_big() . . . 533

4.69.3.4 icmp6_param_problem() . . . 533
4.69.3.5 icmp6_time_exceeded() . . . 533
4.69.3.6 icmp6_time_exceeded_with_addrs() . . . 533
4.70 src/core/ipv6/inet6.c File Reference . . . 534
4.70.1 Variables . . . 534
4.70.2 Detailed Description . . . 534
4.70.3 Variable Documentation . . . 534
4.70.3.1 in6addr_any . . . 534
4.71 src/core/ipv6/ip6.c File Reference . . . 534
4.71.1 Functions . . . 534
4.71.2 Detailed Description . . . 535
4.71.3 Function Documentation . . . 535
4.71.3.1 ip6_input() . . . 535
4.71.3.2 ip6_options_add_hbh_ra() . . . 535
4.71.3.3 ip6_output() . . . 535
4.71.3.4 ip6_output_if() . . . 536
4.71.3.5 ip6_output_if_src() . . . 536
4.71.3.6 ip6_route() . . . 536
4.72 src/core/ipv6/ip6_addr.c File Reference . . . 537
4.72.1 Functions . . . 537
4.72.2 Detailed Description . . . 537
4.72.3 Function Documentation . . . 537
4.72.3.1 ip6addr_aton() . . . 537
4.72.3.2 ip6addr_ntoa() . . . 537
4.72.3.3 ip6addr_ntoa_r() . . . 538
4.73 src/core/ipv6/ip6_frag.c File Reference . . . 538
4.73.1 Data Structures . . . 538
4.73.2 Macros . . . 538
4.73.3 Functions . . . 538
4.73.4 Detailed Description . . . 538
4.73.5 Macro Definition Documentation . . . 538
4.73.5.1 IP_REASS_CHECK_OVERLAP . . . 538
4.73.5.2 IP_REASS_FREE_OLDEST . . . 539
4.73.6 Function Documentation . . . 539
4.73.6.1 ip6_frag() . . . 539
4.73.6.2 ip6_reass() . . . 539
4.74 src/core/ipv6/mld6.c File Reference . . . 539
4.74.1 Functions . . . 539
4.74.2 Detailed Description . . . 540

4.74.3 Function Documentation 540
4.74.3.1 mld6_input() 540
4.74.3.2 mld6_lookfor_group() 540
4.74.3.3 mld6_report_groups() 540
4.74.3.4 mld6_stop() 541
4.74.3.5 mld6_tmr() 541
4.75 src/core/ipv6/nd6.c File Reference 541
4.75.1 Functions 541
4.75.2 Detailed Description 541
4.75.3 Function Documentation 542
4.75.3.1 nd6_adjust_mld_membership() 542
4.75.3.2 nd6_cleanup_netif() 542
4.75.3.3 nd6_clear_destination_cache() 542
4.75.3.4 nd6_find_route() 542
4.75.3.5 nd6_get_destination_mtu() 542
4.75.3.6 nd6_get_next_hop_addr_or_queue() 543
4.75.3.7 nd6_input() 543
4.75.3.8 nd6_reachability_hint() 543
4.75.3.9 nd6_restart_netif() 544
4.75.3.10 nd6_tmr() 544
4.76 src/core/mem.c File Reference 544
4.76.1 Data Structures 544
4.76.2 Macros 544
4.76.3 Functions 544
4.76.4 Variables 544
4.76.5 Detailed Description 545
4.76.6 Macro Definition Documentation 545
4.76.6.1 MIN_SIZE 545
4.76.7 Function Documentation 545
4.76.7.1 mem_calloc() 545
4.76.7.2 mem_free() 545
4.76.7.3 mem_init() 545
4.76.7.4 mem_malloc() 546
4.76.7.5 mem_trim() 546
4.76.8 Variable Documentation 546
4.76.8.1 ram_heap 546
4.77 src/core/memp.c File Reference 546
4.77.1 Functions 547
4.77.2 Detailed Description 547

4.77.3 Function Documentation . . . 547
4.77.3.1 memp_free() . . . 547
4.77.3.2 memp_free_pool() . . . 547
4.77.3.3 memp_init() . . . 547
4.77.3.4 memp_init_pool() . . . 548
4.77.3.5 memp_malloc() . . . 548
4.77.3.6 memp_malloc_pool() . . . 548
4.78 src/core/netif.c File Reference . . . 548
4.78.1 Functions . . . 548
4.78.2 Variables . . . 549
4.78.3 Detailed Description . . . 549
4.78.4 Function Documentation . . . 550
4.78.4.1 netif_get_ip6_addr_match() . . . 550
4.78.4.2 netif_invoke_ext_callback() . . . 550
4.78.4.3 netif_poll() . . . 550
4.78.5 Variable Documentation . . . 550
4.78.5.1 netif_default . . . 550
4.78.5.2 netif_list . . . 550
4.79 src/core/pbuf.c File Reference . . . 551
4.79.1 Functions . . . 551
4.79.2 Detailed Description . . . 552
4.79.3 Function Documentation . . . 552
4.79.3.1 pbuf_add_header() . . . 552
4.79.3.2 pbuf_add_header_force() . . . 552
4.79.3.3 pbuf_clen() . . . 552
4.79.3.4 pbuf_dechain() . . . 552
4.79.3.5 pbuf_free_header() . . . 553
4.79.3.6 pbuf_header() . . . 553
4.79.3.7 pbuf_header_force() . . . 553
4.79.3.8 pbuf_remove_header() . . . 553
4.79.3.9 pbuf_strstr() . . . 554
4.80 src/core/raw.c File Reference . . . 554
4.80.1 Functions . . . 554
4.80.2 Detailed Description . . . 555
4.80.3 Function Documentation . . . 555
4.80.3.1 raw_input() . . . 555
4.80.3.2 raw_netif_ip_addr_changed() . . . 555
4.81 src/core/stats.c File Reference . . . 555
4.81.1 Functions . . . 555

4.81.2 Variables . . . 555
4.81.3 Detailed Description . . . 556
4.81.4 Function Documentation . . . 556
4.81.4.1 stats_init() . . . 556
4.81.5 Variable Documentation . . . 556
4.81.5.1 lwip_stats . . . 556
4.82 src/core/sys.c File Reference . . . 556
4.82.1 Functions . . . 556
4.82.2 Detailed Description . . . 556
4.83 src/core/tcp.c File Reference . . . 556
4.83.1 Functions . . . 556
4.83.2 Variables . . . 558
4.83.3 Detailed Description . . . 558
4.83.4 Function Documentation . . . 558
4.83.4.1 tcp_abandon() . . . 558
4.83.4.2 tcp_alloc() . . . 558
4.83.4.3 tcp_eff_send_mss_netif() . . . 558
4.83.4.4 tcp_ext_arg_invoke_callbacks_passive_open() . . . 559
4.83.4.5 tcp_fasttmr() . . . 559
4.83.4.6 tcp_free() . . . 559
4.83.4.7 tcp_init() . . . 559
4.83.4.8 tcp_netif_ip_addr_changed() . . . 559
4.83.4.9 tcp_next_iss() . . . 559
4.83.4.10 tcp_pcb_purge() . . . 559
4.83.4.11 tcp_pcb_remove() . . . 560
4.83.4.12 tcp_process_refused_data() . . . 560
4.83.4.13 tcp_recv_null() . . . 560
4.83.4.14 tcp_seg_copy() . . . 560
4.83.4.15 tcp_seg_free() . . . 560
4.83.4.16 tcp_segs_free() . . . 560
4.83.4.17 tcp_setprio() . . . 561
4.83.4.18 tcp_slowtmr() . . . 561
4.83.4.19 tcp_tmr() . . . 561
4.83.4.20 tcp_txnow() . . . 561
4.83.4.21 tcp_update_rcv_ann_wnd() . . . 561
4.83.5 Variable Documentation . . . 561
4.83.5.1 tcp_active_pcbs . . . 561
4.83.5.2 tcp_bound_pcbs . . . 562
4.83.5.3 tcp_listen_pcbs . . . 562

4.83.5.4 tcp_pcb_lists . . . 562
4.83.5.5 tcp_tw_pcbs . . . 562
4.84 src/core/tcp_in.c File Reference . . . 562
4.84.1 Macros . . . 562
4.84.2 Functions . . . 562
4.84.3 Detailed Description . . . 562
4.84.4 Macro Definition Documentation . . . 563
4.84.4.1 LWIP_TCP_CALC_INITIAL_CWND . . . 563
4.84.5 Function Documentation . . . 563
4.84.5.1 tcp_input() . . . 563
4.85 src/core/tcp_out.c File Reference . . . 563
4.85.1 Macros . . . 563
4.85.2 Functions . . . 563
4.85.3 Detailed Description . . . 564
4.85.4 Macro Definition Documentation . . . 564
4.85.4.1 TCP_CHECKSUM_ON_COPY_SANITY_CHECK . . . 564
4.85.4.2 TCP_OVERSIZE_CALC_LENGTH . . . 564
4.85.5 Function Documentation . . . 565
4.85.5.1 tcp_enqueue_flags() . . . 565
4.85.5.2 tcp_keepalive() . . . 565
4.85.5.3 tcp_rexmit() . . . 565
4.85.5.4 tcp_rexmit_fast() . . . 565
4.85.5.5 tcp_rexmit_rto() . . . 565
4.85.5.6 tcp_rexmit_rto_commit() . . . 566
4.85.5.7 tcp_rexmit_rto_prepare() . . . 566
4.85.5.8 tcp_rst() . . . 566
4.85.5.9 tcp_rst_netif() . . . 566
4.85.5.10 tcp_send_empty_ack() . . . 567
4.85.5.11 tcp_send_fin() . . . 567
4.85.5.12 tcp_split_unsent_seg() . . . 567
4.85.5.13 tcp_zero_window_probe() . . . 567
4.86 src/core/timeouts.c File Reference . . . 568
4.86.1 Functions . . . 568
4.86.2 Variables . . . 568
4.86.3 Detailed Description . . . 568
4.86.4 Function Documentation . . . 568
4.86.4.1 sys_restart_timeouts() . . . 568
4.86.4.2 sys_timeout() . . . 569
4.86.4.3 sys_timeouts_init() . . . 569

4.86.4.4 sys_timeouts_sleeptime() 569
4.86.4.5 sys_untimeout() 569
4.86.4.6 tcp_timer_needed() 569
4.86.5 Variable Documentation 570
4.86.5.1 lwip_cyclic_timers 570
4.86.5.2 lwip_num_cyclic_timers 570
4.87 src/core/udp.c File Reference 570
4.87.1 Functions 570
4.87.2 Detailed Description 570
4.87.3 Function Documentation 571
4.87.3.1 udp_init() 571
4.87.3.2 udp_input() 571
4.87.3.3 udp_netif_ip_addr_changed() 571
4.88 src/include/compat/posix/arpa/inet.h File Reference 571
4.88.1 Detailed Description 571
4.89 src/include/lwip/inet.h File Reference 571
4.89.1 Macros 572
4.89.2 Variables 572
4.89.3 Detailed Description 572
4.89.4 Macro Definition Documentation 572
4.89.4.1 IN6ADDR_ANY_INIT 572
4.89.4.2 IN6ADDR_LOOPBACK_INIT 572
4.89.4.3 INADDR_ANY 572
4.89.4.4 INADDR_BROADCAST 572
4.89.4.5 INADDR_LOOPBACK 572
4.89.4.6 INADDR_NONE 573
4.89.5 Variable Documentation 573
4.89.5.1 in6addr_any 573
4.90 src/include/compat/posix/net/if.h File Reference 573
4.90.1 Detailed Description 573
4.91 src/include/compat/posix/netdb.h File Reference 573
4.91.1 Detailed Description 573
4.92 src/include/lwip/netdb.h File Reference 573
4.92.1 Macros 573
4.92.2 Functions 573
4.92.3 Variables 574
4.92.4 Detailed Description 574
4.92.5 Macro Definition Documentation 574
4.92.5.1 EAI_NONAME 574

4.92.6 Function Documentation . . . 574
4.92.6.1 lwip_freeaddrinfo() . . . 574
4.92.6.2 lwip_getaddrinfo() . . . 574
4.92.6.3 lwip_gethostbyname() . . . 575
4.92.6.4 lwip_gethostbyname_r() . . . 575
4.92.7 Variable Documentation . . . 575
4.92.7.1 h_errno . . . 575
4.93 src/include/compat/posix/sys/socket.h File Reference . . . 575
4.93.1 Detailed Description . . . 575
4.94 src/include/compat/stdc/errno.h File Reference . . . 575
4.94.1 Detailed Description . . . 576
4.95 src/include/lwip/errno.h File Reference . . . 576
4.95.1 Detailed Description . . . 576
4.96 src/include/lwip/acd.h File Reference . . . 576
4.96.1 Data Structures . . . 576
4.96.2 Macros . . . 576
4.96.3 Typedefs . . . 576
4.96.4 Functions . . . 576
4.96.5 Detailed Description . . . 576
4.96.6 Macro Definition Documentation . . . 577
4.96.6.1 ACD_TMR_INTERVAL . . . 577
4.96.7 Typedef Documentation . . . 577
4.96.7.1 acd_conflict_callback_t . . . 577
4.96.8 Function Documentation . . . 577
4.96.8.1 acd_arp_reply() . . . 577
4.96.8.2 acd_tmr() . . . 577
4.97 src/include/lwip/prot/acd.h File Reference . . . 577
4.97.1 Detailed Description . . . 577
4.98 src/include/lwip/altcp.h File Reference . . . 577
4.98.1 Data Structures . . . 578
4.98.2 Typedefs . . . 578
4.98.3 Functions . . . 578
4.98.4 Detailed Description . . . 578
4.99 src/include/lwip/altcp_tcp.h File Reference . . . 579
4.99.1 Functions . . . 579
4.99.2 Detailed Description . . . 579
4.99.3 Function Documentation . . . 579
4.99.3.1 altcp_tcp_alloc() . . . 579
4.100src/include/lwip/altcp_tls.h File Reference . . . 579

4.100.1 Functions 579
4.100.2 Detailed Description 580
4.101 src/include/lwip/api.h File Reference 580
4.101.1 Data Structures 580
4.101.2 Macros 580
4.101.3 Typedefs 581
4.101.4 Enumerations 581
4.101.5 Functions 581
4.101.6 Detailed Description 582
4.101.7 Macro Definition Documentation 582
4.101.7.1 API_EVENT 582
4.101.7.2 NETCONN_FIN_RX_PENDING 582
4.101.7.3 NETCONN_FLAG_CHECK_WRITESPACE 582
4.101.7.4 NETCONN_FLAG_IN_NONBLOCKING_CONNECT 582
4.101.7.5 NETCONN_FLAG_IPV6_V6ONLY 582
4.101.7.6 NETCONN_FLAG_MBOXCLOSED 582
4.101.7.7 NETCONN_FLAG_NON_BLOCKING 582
4.101.7.8 netconn_get_recvbufsize 583
4.101.7.9 netconn_get_sendtimeout 583
4.101.7.10 netconn_is_nonblocking 583
4.101.7.11 netconn_set_nonblocking 583
4.101.7.12 netconn_set_recvbufsize 583
4.101.7.13 netconn_set_sendtimeout 583
4.101.7.14 netconn_type 583
4.101.8 Typedef Documentation 583
4.101.8.1 netconn_callback 583
4.101.9 Enumeration Type Documentation 584
4.101.9.1 netconn_evt 584
4.101.9.2 netconn_igmp 584
4.101.9.3 netconn_state 584
4.101.10 Function Documentation 584
4.101.10.1 netconn_getaddr() 584
4.101.10.2 netconn_new_with_proto_and_callback() 585
4.101.10.3 netconn_recv_udp_raw_netbuf() 585
4.101.10.4 netconn_recv_udp_raw_netbuf_flags() 585
4.101.10.5 netconn_write_vectors_partly() 585
4.102 src/include/lwip/apps/altcp_proxyconnect.h File Reference 586
4.102.1 Functions 586
4.102.2 Detailed Description 586

4.102.3 Function Documentation . . . 586
4.102.3.1 altcp_proxyconnect_alloc() . . . 586
4.102.3.2 altcp_proxyconnect_new() . . . 587
4.102.3.3 altcp_proxyconnect_new_tcp() . . . 587
4.102.3.4 altcp_proxyconnect_tls_alloc() . . . 587
4.103 src/include/lwip/apps/altcp_tls_mbedtls_opts.h File Reference . . . 587
4.103.1 Macros . . . 587
4.103.2 Detailed Description . . . 588
4.103.3 Macro Definition Documentation . . . 588
4.103.3.1 ALTCP_MBEDTLS_AUTHMODE . . . 588
4.103.3.2 ALTCP_MBEDTLS_DEBUG . . . 588
4.103.3.3 ALTCP_MBEDTLS_LIB_DEBUG . . . 588
4.103.3.4 ALTCP_MBEDTLS_LIB_DEBUG_LEVEL_MIN . . . 588
4.103.3.5 ALTCP_MBEDTLS_SESSION_CACHE_SIZE . . . 588
4.103.3.6 ALTCP_MBEDTLS_SESSION_CACHE_TIMEOUT_SECONDS . . . 589
4.103.3.7 ALTCP_MBEDTLS_SESSION_TICKET_CIPHER . . . 589
4.103.3.8 ALTCP_MBEDTLS_SESSION_TICKET_TIMEOUT_SECONDS . . . 589
4.103.3.9 ALTCP_MBEDTLS_USE_SESSION_CACHE . . . 589
4.103.3.10 ALTCP_MBEDTLS_USE_SESSION_TICKETS . . . 589
4.103.3.11 LWIP_ALTCP_TLS_MBEDTLS . . . 589
4.104 src/include/lwip/apps/http_client.h File Reference . . . 589
4.104.1 Macros . . . 589
4.104.2 Typedefs . . . 589
4.104.3 Enumerations . . . 590
4.104.4 Functions . . . 590
4.104.5 Detailed Description . . . 590
4.105 src/include/lwip/apps/httpd.h File Reference . . . 590
4.105.1 Data Structures . . . 590
4.105.2 Macros . . . 590
4.105.3 Typedefs . . . 590
4.105.4 Functions . . . 590
4.105.5 Detailed Description . . . 591
4.105.6 Macro Definition Documentation . . . 591
4.105.6.1 HTTPD_SSI_TAG_UNKNOWN . . . 591
4.106 src/include/lwip/apps/httpd_opts.h File Reference . . . 591
4.106.1 Macros . . . 591
4.106.2 Detailed Description . . . 592
4.107 src/include/lwip/apps/lwiperf.h File Reference . . . 592
4.107.1 Typedefs . . . 592

4.107.2 Enumerations . . . 592
4.107.3 Functions . . . 593
4.107.4 Detailed Description . . . 593
4.107.5 Typedef Documentation . . . 593
4.107.5.1 lwiperf_report_fn . . . 593
4.107.6 Enumeration Type Documentation . . . 593
4.107.6.1 lwiperf_client_type . . . 593
4.107.6.2 lwiperf_report_type . . . 594
4.108 src/include/lwip/apps/mdns.h File Reference . . . 594
4.108.1 Data Structures . . . 594
4.108.2 Macros . . . 594
4.108.3 Typedefs . . . 594
4.108.4 Functions . . . 594
4.108.5 Detailed Description . . . 595
4.108.6 Typedef Documentation . . . 595
4.108.6.1 mdns_name_result_cb_t . . . 595
4.108.6.2 service_get_txt_fn_t . . . 595
4.108.7 Function Documentation . . . 595
4.108.7.1 mdns_resp_register_name_result_cb() . . . 595
4.109 src/include/lwip/apps/mdns_domain.h File Reference . . . 595
4.109.1 Functions . . . 595
4.109.2 Detailed Description . . . 596
4.109.3 Function Documentation . . . 596
4.109.3.1 mdns_build_dnssd_domain() . . . 596
4.109.3.2 mdns_build_host_domain() . . . 596
4.109.3.3 mdns_build_request_domain() . . . 596
4.109.3.4 mdns_build_reverse_v4_domain() . . . 597
4.109.3.5 mdns_build_reverse_v6_domain() . . . 597
4.109.3.6 mdns_build_service_domain() . . . 597
4.109.3.7 mdns_compress_domain() . . . 597
4.109.3.8 mdns_domain_add_domain() . . . 598
4.109.3.9 mdns_domain_add_label() . . . 598
4.109.3.10 mdns_domain_add_string() . . . 598
4.109.3.11 mdns_domain_debug_print() . . . 598
4.109.3.12 mdns_domain_eq() . . . 598
4.109.3.13 mdns_readname() . . . 599
4.109.3.14 mdns_write_domain() . . . 599
4.110 src/include/lwip/apps/mdns_opts.h File Reference . . . 599
4.110.1 Macros . . . 599

4.110.2 Detailed Description 599
4.111 src/include/lwip/apps/mdns_out.h File Reference 600
4.111.1 Macros 600
4.111.2 Functions 600
4.111.3 Detailed Description 601
4.111.4 Macro Definition Documentation 601
4.111.4.1 QUESTION_PROBE_HOST_ANY 601
4.111.5 Function Documentation 601
4.111.5.1 mdns_create_outpacket() 601
4.111.5.2 mdns_multicast_probe_timeout_reset_ipv4() 601
4.111.5.3 mdns_multicast_probe_timeout_reset_ipv6() 601
4.111.5.4 mdns_multicast_timeout_25ttl_reset_ipv4() 601
4.111.5.5 mdns_multicast_timeout_25ttl_reset_ipv6() 602
4.111.5.6 mdns_multicast_timeout_reset_ipv4() 602
4.111.5.7 mdns_multicast_timeout_reset_ipv6() 602
4.111.5.8 mdns_prepare_txtdata() 602
4.111.5.9 mdns_send_multicast_msg_delayed_ipv4() 602
4.111.5.10 mdns_send_multicast_msg_delayed_ipv6() 602
4.111.5.11 mdns_send_outpacket() 603
4.111.5.12 mdns_send_request() 603
4.111.5.13 mdns_send_unicast_msg_delayed_ipv4() 603
4.111.5.14 mdns_send_unicast_msg_delayed_ipv6() 603
4.111.5.15 mdns_set_timeout() 603
4.111.5.16 mdns_start_multicast_timeouts_ipv4() 603
4.111.5.17 mdns_start_multicast_timeouts_ipv6() 604
4.112 src/include/lwip/apps/mdns_priv.h File Reference 604
4.112.1 Data Structures 604
4.112.2 Functions 604
4.112.3 Detailed Description 605
4.112.4 Function Documentation 605
4.112.4.1 get_mdns_pcb() 605
4.112.4.2 netif_mdns_data() 605
4.113 src/include/lwip/apps/mqtt.h File Reference 605
4.113.1 Data Structures 605
4.113.2 Macros 605
4.113.3 Typedefs 606
4.113.4 Enumerations 606
4.113.5 Functions 606
4.113.6 Detailed Description 606

4.114 src/include/lwip/apps/mqtt_opts.h File Reference 606
4.114.1 Macros 607
4.114.2 Detailed Description 607
4.115 src/include/lwip/apps/mqtt_priv.h File Reference 607
4.115.1 Data Structures 607
4.115.2 Detailed Description 607
4.116 src/include/lwip/apps/netbiosns.h File Reference 607
4.116.1 Functions 607
4.116.2 Detailed Description 607
4.117 src/include/lwip/apps/netbiosns_opts.h File Reference 608
4.117.1 Macros 608
4.117.2 Detailed Description 608
4.118 src/include/lwip/apps/snmp.h File Reference 608
4.118.1 Data Structures 608
4.118.2 Macros 608
4.118.3 Functions 608
4.118.4 Detailed Description 609
4.118.5 Macro Definition Documentation 609
4.118.5.1 SNMP_GENTRAP_AUTH_FAILURE 609
4.118.5.2 SNMP_GENTRAP_COLDSTART 609
4.118.5.3 SNMP_GENTRAP_EGP_NEIGHBOR_LOSS 609
4.118.5.4 SNMP_GENTRAP_ENTERPRISE_SPECIFIC 610
4.118.5.5 SNMP_GENTRAP_LINKDOWN 610
4.118.5.6 SNMP_GENTRAP_LINKUP 610
4.118.5.7 SNMP_GENTRAP_WARMSTART 610
4.119 src/include/lwip/snmp.h File Reference 610
4.119.1 Macros 610
4.119.2 Enumerations 610
4.119.3 Detailed Description 610
4.119.4 Macro Definition Documentation 610
4.119.4.1 MIB2_COPY_SYSUPTIME_TO 610
4.120 src/include/lwip/apps/snmp_core.h File Reference 611
4.120.1 Data Structures 611
4.120.2 Macros 611
4.120.3 Enumerations 611
4.120.4 Functions 611
4.120.5 Detailed Description 612
4.120.6 Macro Definition Documentation 612
4.120.6.1 SNMP_NODE_TREE 612

4.120.7 Enumeration Type Documentation . . . 612
4.120.7.1 snmp_access_t . . . 612
4.120.7.2 snmp_err_t . . . 612
4.120.8 Function Documentation . . . 612
4.120.8.1 netif_to_num() . . . 612
4.120.8.2 snmp_decode_bits() . . . 613
4.120.8.3 snmp_encode_bits() . . . 613
4.120.8.4 snmp_ip4_to_oid() . . . 613
4.120.8.5 snmp_ip6_to_oid() . . . 614
4.120.8.6 snmp_ip_port_to_oid() . . . 614
4.120.8.7 snmp_ip_to_oid() . . . 614
4.120.8.8 snmp_next_oid_check() . . . 614
4.120.8.9 snmp_next_oid_init() . . . 614
4.120.8.10 snmp_next_oid_precheck() . . . 615
4.120.8.11 snmp_oid_append() . . . 615
4.120.8.12 snmp_oid_assign() . . . 615
4.120.8.13 snmp_oid_combine() . . . 615
4.120.8.14 snmp_oid_compare() . . . 615
4.120.8.15 snmp_oid_equal() . . . 616
4.120.8.16 snmp_oid_in_range() . . . 616
4.120.8.17 snmp_oid_prefix() . . . 616
4.120.8.18 snmp_oid_to_ip() . . . 616
4.120.8.19 snmp_oid_to_ip4() . . . 616
4.120.8.20 snmp_oid_to_ip6() . . . 617
4.120.8.21 snmp_oid_to_ip_port() . . . 617
4.121 src/include/lwip/apps/snmp_mib2.h File Reference . . . 617
4.121.1 Functions . . . 617
4.121.2 Detailed Description . . . 617
4.122 src/include/lwip/apps/snmp_opts.h File Reference . . . 618
4.122.1 Macros . . . 618
4.122.2 Detailed Description . . . 619
4.122.3 Macro Definition Documentation . . . 619
4.122.3.1 LWIP_SNMP_V3 . . . 619
4.123 src/include/lwip/apps/snmp_scalar.h File Reference . . . 619
4.123.1 Data Structures . . . 619
4.123.2 Detailed Description . . . 619
4.124 src/include/lwip/apps/snmp_table.h File Reference . . . 619
4.124.1 Data Structures . . . 619
4.124.2 Enumerations . . . 620

4.124.3 Detailed Description 620
4.124.4 Enumeration Type Documentation 620
4.124.4.1 snmp_table_column_data_type_t 620
4.125 src/include/lwip/apps/snmp_threadsync.h File Reference 620
4.125.1 Data Structures 620
4.125.2 Macros 620
4.125.3 Functions 620
4.125.4 Detailed Description 620
4.125.5 Macro Definition Documentation 620
4.125.5.1 SNMP_CREATE_THREAD_SYNC_NODE 620
4.125.6 Function Documentation 621
4.125.6.1 snmp_threadsync_init() 621
4.126 src/include/lwip/apps/snmpv3.h File Reference 621
4.126.1 Detailed Description 621
4.127 src/include/lwip/apps/sntp.h File Reference 621
4.127.1 Functions 621
4.127.2 Detailed Description 621
4.128 src/include/lwip/apps/sntp_opts.h File Reference 621
4.128.1 Macros 622
4.128.2 Detailed Description 622
4.129 src/include/lwip/apps/tftp_client.h File Reference 622
4.129.1 Functions 622
4.129.2 Detailed Description 622
4.130 src/include/lwip/apps/tftp_common.h File Reference 623
4.130.1 Data Structures 623
4.130.2 Functions 623
4.130.3 Detailed Description 623
4.130.4 Function Documentation 623
4.130.4.1 tftp_init_common() 623
4.131 src/include/lwip/apps/tftp_opts.h File Reference 623
4.131.1 Macros 624
4.131.2 Detailed Description 624
4.132 src/include/lwip/apps/tftp_server.h File Reference 624
4.132.1 Functions 624
4.132.2 Detailed Description 624
4.133 src/include/lwip/arch.h File Reference 624
4.133.1 Macros 624
4.133.2 Detailed Description 625
4.134 src/include/lwip/autoip.h File Reference 625

4.134.1 Data Structures . . . 625
4.134.2 Functions . . . 626
4.134.3 Detailed Description . . . 626
4.134.4 Function Documentation . . . 626
4.134.4.1 autoip_network_changed_link_down() . . . 626
4.134.4.2 autoip_network_changed_link_up() . . . 626
4.134.4.3 autoip_supplied_address() . . . 626
4.135 src/include/lwip/prot/autoip.h File Reference . . . 626
4.135.1 Detailed Description . . . 626
4.136 src/include/lwip/debug.h File Reference . . . 627
4.136.1 Macros . . . 627
4.136.2 Debug level (LWIP_DBG_MIN_LEVEL) . . . 627
4.136.3 Enable/disable debug messages completely (LWIP_DBG_TYPES_ON) . . . 627
4.136.4 Debug message types (LWIP_DBG_TYPES_ON) . . . 627
4.136.5 Detailed Description . . . 627
4.136.6 Macro Definition Documentation . . . 627
4.136.6.1 LWIP_DEBUG . . . 627
4.137 src/include/lwip/def.h File Reference . . . 627
4.137.1 Macros . . . 628
4.137.2 Functions . . . 628
4.137.3 Detailed Description . . . 628
4.137.4 Macro Definition Documentation . . . 628
4.137.4.1 LWIP_MAKEU32 . . . 628
4.137.5 Function Documentation . . . 628
4.137.5.1 lwip_htonl() . . . 628
4.137.5.2 lwip_htons() . . . 629
4.138 src/include/lwip/dhcp.h File Reference . . . 629
4.138.1 Macros . . . 629
4.138.2 Typedefs . . . 629
4.138.3 Functions . . . 629
4.138.4 Detailed Description . . . 630
4.138.5 Macro Definition Documentation . . . 630
4.138.5.1 DHCP_COARSE_TIMER_MSECS . . . 630
4.138.5.2 DHCP_COARSE_TIMER_SECS . . . 630
4.138.5.3 DHCP_FINE_TIMER_MSECS . . . 630
4.138.5.4 dhcp_remove_struct . . . 630
4.138.6 Typedef Documentation . . . 630
4.138.6.1 dhcp_timeout_t . . . 630
4.138.7 Function Documentation . . . 630

4.138.7.1 dhcp_coarse_tmr() . . . 630
4.138.7.2 dhcp_fine_tmr() . . . 630
4.138.7.3 dhcp_network_changed_link_up() . . . 631
4.138.7.4 dhcp_supplied_address() . . . 631
4.139 src/include/lwip/prot/dhcp.h File Reference . . . 631
4.139.1 Data Structures . . . 631
4.139.2 Macros . . . 631
4.139.3 Detailed Description . . . 631
4.139.4 Macro Definition Documentation . . . 631
4.139.4.1 DHCP_OPTIONS_LEN . . . 631
4.140 src/include/lwip/dhcp6.h File Reference . . . 631
4.140.1 Macros . . . 632
4.140.2 Functions . . . 632
4.140.3 Detailed Description . . . 632
4.140.4 Macro Definition Documentation . . . 632
4.140.4.1 dhcp6_remove_struct . . . 632
4.140.4.2 DHCP6_TIMER_MSECS . . . 632
4.140.5 Function Documentation . . . 632
4.140.5.1 dhcp6_nd6_ra_trigger() . . . 632
4.140.5.2 dhcp6_tmr() . . . 632
4.141 src/include/lwip/prot/dhcp6.h File Reference . . . 633
4.141.1 Data Structures . . . 633
4.141.2 Macros . . . 633
4.141.3 Detailed Description . . . 633
4.141.4 Macro Definition Documentation . . . 633
4.141.4.1 DHCP6_DUID_LLT . . . 633
4.141.4.2 DHCP6_STATUS_SUCCESS . . . 633
4.142 src/include/lwip/dns.h File Reference . . . 633
4.142.1 Macros . . . 633
4.142.2 Typedefs . . . 633
4.142.3 Functions . . . 634
4.142.4 Detailed Description . . . 634
4.142.5 Macro Definition Documentation . . . 634
4.142.5.1 DNS_TMR_INTERVAL . . . 634
4.142.6 Typedef Documentation . . . 634
4.142.6.1 dns_found_callback . . . 634
4.142.7 Function Documentation . . . 635
4.142.7.1 dns_init() . . . 635
4.142.7.2 dns_tmr() . . . 635

4.143 src/include/lwip/prot/dns.h File Reference 635
4.143.1 Data Structures 635
4.143.2 Macros 635
4.143.3 Detailed Description 635
4.143.4 Macro Definition Documentation 635
4.143.4.1 DNS_MQUERY_PORT 635
4.143.4.2 DNS_SERVER_PORT 635
4.144 src/include/lwip/err.h File Reference 636
4.144.1 Typedefs 636
4.144.2 Enumerations 636
4.144.3 Detailed Description 636
4.145 src/include/lwip/etharp.h File Reference 636
4.145.1 Data Structures 636
4.145.2 Macros 636
4.145.3 Functions 636
4.145.4 Detailed Description 637
4.145.5 Macro Definition Documentation 637
4.145.5.1 ARP_TMR_INTERVAL 637
4.145.5.2 etharp_gratuitous 637
4.145.6 Function Documentation 637
4.145.6.1 etharp_acd_announce() 637
4.145.6.2 etharp_acd_probe() 637
4.145.6.3 etharp_cleanup_netif() 637
4.145.6.4 etharp_find_addr() 638
4.145.6.5 etharp_get_entry() 638
4.145.6.6 etharp_input() 638
4.145.6.7 etharp_output() 639
4.145.6.8 etharp_query() 639
4.145.6.9 etharp_request() 640
4.145.6.10 etharp_tmr() 640
4.146 src/include/lwip/prot/etharp.h File Reference 640
4.146.1 Data Structures 640
4.146.2 Macros 640
4.146.3 Detailed Description 640
4.146.4 Macro Definition Documentation 641
4.146.4.1 IPADDR_WORDALIGNED_COPY_FROM_IP4_ADDR_T 641
4.146.4.2 IPADDR_WORDALIGNED_COPY_TO_IP4_ADDR_T 641
4.147 src/include/lwip/ethip6.h File Reference 641
4.147.1 Functions 641

4.147.2 Detailed Description . . . 641
4.147.3 Function Documentation . . . 641
4.147.3.1 ethip6_output() . . . 641
4.148 src/include/lwip/icmp.h File Reference . . . 642
4.148.1 Enumerations . . . 642
4.148.2 Functions . . . 642
4.148.3 Detailed Description . . . 642
4.148.4 Enumeration Type Documentation . . . 642
4.148.4.1 icmp_dur_type . . . 642
4.148.4.2 icmp_te_type . . . 642
4.148.5 Function Documentation . . . 643
4.148.5.1 icmp_dest_unreach() . . . 643
4.148.5.2 icmp_input() . . . 643
4.148.5.3 icmp_time_exceeded() . . . 643
4.149 src/include/lwip/prot/icmp.h File Reference . . . 643
4.149.1 Data Structures . . . 643
4.149.2 Detailed Description . . . 644
4.150 src/include/lwip/icmp6.h File Reference . . . 644
4.150.1 Functions . . . 644
4.150.2 Detailed Description . . . 644
4.150.3 Function Documentation . . . 644
4.150.3.1 icmp6_dest_unreach() . . . 644
4.150.3.2 icmp6_input() . . . 644
4.150.3.3 icmp6_packet_too_big() . . . 645
4.150.3.4 icmp6_param_problem() . . . 645
4.150.3.5 icmp6_time_exceeded() . . . 645
4.150.3.6 icmp6_time_exceeded_with_addrs() . . . 645
4.151 src/include/lwip/prot/icmp6.h File Reference . . . 646
4.151.1 Data Structures . . . 646
4.151.2 Enumerations . . . 646
4.151.3 Detailed Description . . . 646
4.151.4 Enumeration Type Documentation . . . 646
4.151.4.1 icmp6_dur_code . . . 646
4.151.4.2 icmp6_pp_code . . . 647
4.151.4.3 icmp6_te_code . . . 647
4.151.4.4 icmp6_type . . . 647
4.152 src/include/lwip/if_api.h File Reference . . . 648
4.152.1 Functions . . . 648
4.152.2 Detailed Description . . . 648

4.153 src/include/lwip/igmp.h File Reference 648
4.153.1 Data Structures 648
4.153.2 Macros 648
4.153.3 Functions 648
4.153.4 Detailed Description 649
4.153.5 Function Documentation 649
4.153.5.1 igmp_init() 649
4.153.5.2 igmp_input() 649
4.153.5.3 igmp_lookfor_group() 649
4.153.5.4 igmp_report_groups() 649
4.153.5.5 igmp_start() 649
4.153.5.6 igmp_stop() 650
4.153.5.7 igmp_tmr() 650
4.154 src/include/lwip/prot/igmp.h File Reference 650
4.154.1 Data Structures 650
4.154.2 Detailed Description 650
4.155 src/include/lwip/inet_chksum.h File Reference 650
4.155.1 Macros 650
4.155.2 Functions 650
4.155.3 Detailed Description 651
4.155.4 Macro Definition Documentation 651
4.155.4.1 FOLD_U32T 651
4.155.4.2 SWAP_BYTES_IN_WORD 651
4.155.5 Function Documentation 651
4.155.5.1 inet_chksum_pbuf() 651
4.155.5.2 ip6_chksum_pseudo() 651
4.155.5.3 ip6_chksum_pseudo_partial() 652
4.156 src/include/lwip/init.h File Reference 652
4.156.1 Macros 652
4.156.2 Functions 652
4.156.3 Detailed Description 652
4.157 src/include/lwip/ip.h File Reference 652
4.157.1 Data Structures 653
4.157.2 Macros 653
4.157.3 Functions 653
4.157.4 Variables 654
4.157.5 Detailed Description 654
4.157.6 Macro Definition Documentation 654
4.157.6.1 ip4_current_dest_addr 654

4.157.6.2 ip4_current_header . . . 654
4.157.6.3 ip4_current_src_addr . . . 654
4.157.6.4 ip6_current_dest_addr . . . 654
4.157.6.5 ip6_current_header . . . 654
4.157.6.6 ip6_current_src_addr . . . 654
4.157.6.7 ip_current_dest_addr [1/2] . . . 654
4.157.6.8 ip_current_dest_addr [2/2] . . . 655
4.157.6.9 ip_current_header_proto . . . 655
4.157.6.10 ip_current_header_tot_len . . . 655
4.157.6.11 ip_current_input_netif . . . 655
4.157.6.12 ip_current_is_v6 . . . 655
4.157.6.13 ip_current_netif . . . 655
4.157.6.14 ip_current_src_addr [1/2] . . . 655
4.157.6.15 ip_current_src_addr [2/2] . . . 655
4.157.6.16 ip_get_option . . . 656
4.157.6.17 ip_next_header_ptr . . . 656
4.157.6.18 ip_output_hinted . . . 656
4.157.6.19 ip_output_if_hdrincl . . . 656
4.157.6.20 IP_PCB . . . 656
4.157.6.21 ip_reset_option . . . 657
4.157.6.22 ip_set_option . . . 657
4.157.6.23 LWIP_IP_CHECK_PBUF_REF_COUNT_FOR_TX . . . 657
4.157.7 Variable Documentation . . . 657
4.157.7.1 ip_data . . . 657
4.158 src/include/lwip/prot/ip.h File Reference . . . 657
4.158.1 Macros . . . 657
4.158.2 Detailed Description . . . 657
4.158.3 Macro Definition Documentation . . . 657
4.158.3.1 IP_HDR_GET_VERSION . . . 657
4.159 src/include/lwip/ip4.h File Reference . . . 657
4.159.1 Macros . . . 658
4.159.2 Functions . . . 658
4.159.3 Detailed Description . . . 658
4.159.4 Macro Definition Documentation . . . 658
4.159.4.1 IP_OPTIONS_SEND . . . 658
4.159.5 Function Documentation . . . 658
4.159.5.1 ip4_input() . . . 658
4.159.5.2 ip4_output() . . . 659
4.159.5.3 ip4_output_if() . . . 659

4.159.5.4 ip4_output_if_opt() . . . 659
4.159.5.5 ip4_output_if_opt_src() . . . 660
4.159.5.6 ip4_output_if_src() . . . 660
4.159.5.7 ip4_route() . . . 660
4.159.5.8 ip4_route_src() . . . 660
4.160 src/include/lwip/prot/ip4.h File Reference . . . 660
4.160.1 Data Structures . . . 660
4.160.2 Detailed Description . . . 660
4.161 src/include/lwip/ip4_addr.h File Reference . . . 660
4.161.1 Data Structures . . . 661
4.161.2 Macros . . . 661
4.161.3 Typedefs . . . 661
4.161.4 Functions . . . 661
4.161.5 Detailed Description . . . 662
4.161.6 Macro Definition Documentation . . . 662
4.161.6.1 IP4_ADDR . . . 662
4.161.6.2 ip4_addr_cmp . . . 662
4.161.6.3 ip4_addr_copy . . . 662
4.161.6.4 ip4_addr_get_network . . . 662
4.161.6.5 ip4_addr_get_u32 . . . 662
4.161.6.6 ip4_addr_isloopback . . . 662
4.161.6.7 ip4_addr_net_eq . . . 662
4.161.6.8 ip4_addr_netcmp . . . 663
4.161.6.9 ip4_addr_set . . . 663
4.161.6.10 ip4_addr_set_any . . . 663
4.161.6.11 ip4_addr_set_hton . . . 663
4.161.6.12 ip4_addr_set_loopback . . . 663
4.161.6.13 ip4_addr_set_u32 . . . 663
4.161.6.14 ip4_addr_set_zero . . . 663
4.161.6.15 ip_ntoa . . . 663
4.161.6.16 IPADDR_ANY . . . 664
4.161.6.17 IPADDR_BROADCAST . . . 664
4.161.6.18 IPADDR_LOOPBACK . . . 664
4.161.6.19 IPADDR_NONE . . . 664
4.161.7 Typedef Documentation . . . 664
4.161.7.1 ip4_addr_t . . . 664
4.161.8 Function Documentation . . . 664
4.161.8.1 ip4_addr_isbroadcast_u32() . . . 664
4.161.8.2 ip4_addr_netmask_valid() . . . 664

4.161.8.3 ip4addr_aton() . . . 665
4.161.8.4 ip4addr_ntoa() . . . 665
4.161.8.5 ip4addr_ntoa_r() . . . 665
4.161.8.6 ipaddr_addr() . . . 665
4.162 src/include/lwip/ip4_frag.h File Reference . . . 665
4.162.1 Data Structures . . . 666
4.162.2 Functions . . . 666
4.162.3 Detailed Description . . . 666
4.162.4 Function Documentation . . . 666
4.162.4.1 ip4_frag() . . . 666
4.162.4.2 ip4_reass() . . . 666
4.162.4.3 ip_reass_tmr() . . . 667
4.163 src/include/lwip/ip6.h File Reference . . . 667
4.163.1 Functions . . . 667
4.163.2 Detailed Description . . . 667
4.163.3 Function Documentation . . . 667
4.163.3.1 ip6_input() . . . 667
4.163.3.2 ip6_options_add_hbh_ra() . . . 668
4.163.3.3 ip6_output() . . . 668
4.163.3.4 ip6_output_if() . . . 668
4.163.3.5 ip6_output_if_src() . . . 668
4.163.3.6 ip6_route() . . . 669
4.164 src/include/lwip/prot/ip6.h File Reference . . . 669
4.164.1 Data Structures . . . 669
4.164.2 Detailed Description . . . 669
4.165 src/include/lwip/ip6_addr.h File Reference . . . 669
4.165.1 Data Structures . . . 670
4.165.2 Macros . . . 670
4.165.3 Typedefs . . . 670
4.165.4 Functions . . . 671
4.165.5 Detailed Description . . . 671
4.165.6 Macro Definition Documentation . . . 671
4.165.6.1 IP6_ADDR . . . 671
4.165.6.2 IP6_ADDR_BLOCK1 . . . 671
4.165.6.3 IP6_ADDR_BLOCK2 . . . 671
4.165.6.4 IP6_ADDR_BLOCK3 . . . 671
4.165.6.5 IP6_ADDR_BLOCK4 . . . 671
4.165.6.6 IP6_ADDR_BLOCK5 . . . 671
4.165.6.7 IP6_ADDR_BLOCK6 . . . 672

4.165.6.8 IP6_ADDR_BLOCK7 . . . 672
4.165.6.9 IP6_ADDR_BLOCK8 . . . 672
4.165.6.10 ip6_addr_cmp . . . 672
4.165.6.11 ip6_addr_cmp_packed . . . 672
4.165.6.12 ip6_addr_cmp_zoneless . . . 672
4.165.6.13 ip6_addr_copy . . . 672
4.165.6.14 ip6_addr_copy_from_packed . . . 672
4.165.6.15 ip6_addr_copy_to_packed . . . 673
4.165.6.16 ip6_addr_eq . . . 673
4.165.6.17 ip6_addr_net_eq . . . 673
4.165.6.18 ip6_addr_net_zoneless_eq . . . 673
4.165.6.19 ip6_addr_netcmp . . . 674
4.165.6.20 ip6_addr_netcmp_zoneless . . . 674
4.165.6.21 ip6_addr_packed_eq . . . 674
4.165.6.22 IP6_ADDR_PART . . . 674
4.165.6.23 ip6_addr_set . . . 674
4.165.6.24 ip6_addr_set_any . . . 674
4.165.6.25 ip6_addr_set_hton . . . 675
4.165.6.26 ip6_addr_set_loopback . . . 675
4.165.6.27 ip6_addr_set_zero . . . 675
4.165.6.28 ip6_addr_zoneless_eq . . . 675
4.165.7 Typedef Documentation . . . 675
4.165.7.1 ip6_addr_t . . . 675
4.165.8 Function Documentation . . . 676
4.165.8.1 ip6addr_aton() . . . 676
4.165.8.2 ip6addr_ntoa() . . . 676
4.165.8.3 ip6addr_ntoa_r() . . . 676
4.166 src/include/lwip/ip6_frag.h File Reference . . . 676
4.166.1 Data Structures . . . 676
4.166.2 Macros . . . 677
4.166.3 Functions . . . 677
4.166.4 Detailed Description . . . 677
4.166.5 Macro Definition Documentation . . . 677
4.166.5.1 IP6_REASS_TMR_INTERVAL . . . 677
4.166.5.2 IPV6_FRAG_COPYHEADER . . . 677
4.166.6 Function Documentation . . . 677
4.166.6.1 ip6_frag() . . . 677
4.166.6.2 ip6_reass() . . . 678
4.167 src/include/lwip/ip6_zone.h File Reference . . . 678

4.167.1 Macros . . . 678
4.167.2 Enumerations . . . 678
4.167.3 Detailed Description . . . 679
4.168 src/include/lwip/ip_addr.h File Reference . . . 679
4.168.1 Data Structures . . . 679
4.168.2 Macros . . . 679
4.168.3 Typedefs . . . 680
4.168.4 Enumerations . . . 680
4.168.5 Functions . . . 680
4.168.6 Detailed Description . . . 680
4.169 src/include/lwip/mem.h File Reference . . . 680
4.169.1 Functions . . . 681
4.169.2 Detailed Description . . . 681
4.169.3 Function Documentation . . . 681
4.169.3.1 mem_calloc() . . . 681
4.169.3.2 mem_free() . . . 681
4.169.3.3 mem_init() . . . 681
4.169.3.4 mem_malloc() . . . 682
4.169.3.5 mem_trim() . . . 682
4.170 src/include/lwip/memp.h File Reference . . . 682
4.170.1 Macros . . . 682
4.170.2 Enumerations . . . 682
4.170.3 Functions . . . 682
4.170.4 Detailed Description . . . 683
4.170.5 Enumeration Type Documentation . . . 683
4.170.5.1 memp_t . . . 683
4.170.6 Function Documentation . . . 683
4.170.6.1 memp_free() . . . 683
4.170.6.2 memp_init() . . . 683
4.170.6.3 memp_malloc() . . . 683
4.171 src/include/lwip/mld6.h File Reference . . . 683
4.171.1 Data Structures . . . 683
4.171.2 Macros . . . 684
4.171.3 Functions . . . 684
4.171.4 Detailed Description . . . 684
4.171.5 Function Documentation . . . 684
4.171.5.1 mld6_input() . . . 684
4.171.5.2 mld6_lookfor_group() . . . 684
4.171.5.3 mld6_report_groups() . . . 685

4.171.5.4 mld6_stop() 685

4.171.5.5 mld6_tmr() 685

4.172 src/include/lwip/prot/mld6.h File Reference 685

4.172.1 Data Structures 685

4.172.2 Detailed Description 685

4.173 src/include/lwip/nd6.h File Reference 685

4.173.1 Macros 685

4.173.2 Functions 686

4.173.3 Detailed Description 686

4.173.4 Macro Definition Documentation 686

4.173.4.1 ND6_RTR_SOLICITATION_INTERVAL 686

4.173.4.2 ND6_TMR_INTERVAL 686

4.173.5 Function Documentation 686

4.173.5.1 nd6_adjust_mld_membership() 686

4.173.5.2 nd6_cleanup_netif() 687

4.173.5.3 nd6_clear_destination_cache() 687

4.173.5.4 nd6_find_route() 687

4.173.5.5 nd6_get_destination_mtu() 687

4.173.5.6 nd6_get_next_hop_addr_or_queue() 687

4.173.5.7 nd6_input() 688

4.173.5.8 nd6_reachability_hint() 688

4.173.5.9 nd6_restart_netif() 688

4.173.5.10 nd6_tmr() 689

4.174 src/include/lwip/prot/nd6.h File Reference 689

4.174.1 Data Structures 689

4.174.2 Macros 689

4.174.3 Detailed Description 689

4.174.4 Macro Definition Documentation 690

4.174.4.1 ND6_OPTION_TYPE_MTU 690

4.174.4.2 ND6_OPTION_TYPE_PREFIX_INFO 690

4.174.4.3 ND6_OPTION_TYPE_RDNSS 690

4.174.4.4 ND6_OPTION_TYPE_REDIR_HDR 690

4.174.4.5 ND6_OPTION_TYPE_ROUTE_INFO 690

4.174.4.6 ND6_OPTION_TYPE_SOURCE_LLADDR 690

4.174.4.7 ND6_RA_FLAG_MANAGED_ADDR_CONFIG 690

4.175 src/include/lwip/netbuf.h File Reference 690

4.175.1 Data Structures 690

4.175.2 Macros 691

4.175.3 Functions 691

4.175.4 Detailed Description . . . 691
4.175.5 Macro Definition Documentation . . . 691
4.175.5.1 NETBUF_FLAG_CHKSUM . . . 691
4.175.5.2 NETBUF_FLAG_DESTADDR . . . 691
4.176 src/include/lwip/netif.h File Reference . . . 691
4.176.1 Data Structures . . . 692
4.176.2 Macros . . . 692
4.176.3 Typedefs . . . 693
4.176.4 Enumerations . . . 693
4.176.5 Functions . . . 693
4.176.6 Variables . . . 694
4.176.7 Detailed Description . . . 694
4.176.8 Macro Definition Documentation . . . 694
4.176.8.1 LWIP_NSC_IPV4_ADDR_VALID . . . 694
4.176.8.2 LWIP_NSC_IPV4_ADDRESS_CHANGED . . . 694
4.176.8.3 LWIP_NSC_IPV4_GATEWAY_CHANGED . . . 694
4.176.8.4 LWIP_NSC_IPV4_NETMASK_CHANGED . . . 695
4.176.8.5 LWIP_NSC_IPV4_SETTINGS_CHANGED . . . 695
4.176.8.6 LWIP_NSC_IPV6_ADDR_STATE_CHANGED . . . 695
4.176.8.7 LWIP_NSC_IPV6_SET . . . 695
4.176.8.8 LWIP_NSC_LINK_CHANGED . . . 695
4.176.8.9 LWIP_NSC_NETIF_ADDED . . . 695
4.176.8.10 LWIP_NSC_NETIF_REMOVED . . . 695
4.176.8.11 LWIP_NSC_STATUS_CHANGED . . . 695
4.176.8.12 netif_get_igmp_mac_filter . . . 695
4.176.8.13 netif_get_mld_mac_filter . . . 696
4.176.8.14 netif_is_link_up . . . 696
4.176.8.15 NETIF_MAX_HWADDR_LEN . . . 696
4.176.8.16 NETIF_NAMESIZE . . . 696
4.176.9 Typedef Documentation . . . 696
4.176.9.1 netif_igmp_mac_filter_fn . . . 696
4.176.9.2 netif_init_fn . . . 696
4.176.9.3 netif_input_fn . . . 696
4.176.9.4 netif_linkoutput_fn . . . 697
4.176.9.5 netif_mld_mac_filter_fn . . . 697
4.176.9.6 netif_output_fn . . . 697
4.176.9.7 netif_output_ip6_fn . . . 697
4.176.9.8 netif_status_callback_fn . . . 697
4.176.10 Enumeration Type Documentation . . . 698

4.176.10.1 netif_mac_filter_action . . . 698
4.176.11 Function Documentation . . . 698
4.176.11.1 netif_get_ip6_addr_match() . . . 698
4.176.11.2 netif_invoke_ext_callback() . . . 698
4.176.11.3 netif_poll() . . . 698
4.176.12 Variable Documentation . . . 698
4.176.12.1 netif_default . . . 698
4.176.12.2 netif_list . . . 699
4.177 src/include/lwip/netifapi.h File Reference . . . 699
4.177.1 Macros . . . 699
4.177.2 Functions . . . 699
4.177.3 Detailed Description . . . 700
4.177.4 Function Documentation . . . 700
4.177.4.1 netifapi_arp_add() . . . 700
4.177.4.2 netifapi_arp_remove() . . . 700
4.177.4.3 netifapi_netif_common() . . . 700
4.178 src/include/lwip/opt.h File Reference . . . 700
4.178.1 Macros . . . 701
4.178.2 Detailed Description . . . 710
4.179 src/include/lwip/pbuf.h File Reference . . . 710
4.179.1 Data Structures . . . 710
4.179.2 Macros . . . 710
4.179.3 Typedefs . . . 711
4.179.4 Enumerations . . . 711
4.179.5 Functions . . . 711
4.179.6 Detailed Description . . . 712
4.179.7 Macro Definition Documentation . . . 712
4.179.7.1 LWIP_SUPPORT_CUSTOM_PBUF . . . 712
4.179.7.2 PBUF_ALLOC_FLAG_DATA_CONTIGUOUS . . . 712
4.179.7.3 PBUF_ALLOC_FLAG_RX . . . 712
4.179.7.4 PBUF_FLAG_IS_CUSTOM . . . 712
4.179.7.5 PBUF_FLAG_LLBCAST . . . 713
4.179.7.6 PBUF_FLAG_LLMCAST . . . 713
4.179.7.7 PBUF_FLAG_MCASTLOOP . . . 713
4.179.7.8 PBUF_FLAG_PUSH . . . 713
4.179.7.9 PBUF_FLAG_TCP_FIN . . . 713
4.179.7.10 PBUF_POOL_FREE_OOSEQ . . . 713
4.179.7.11 PBUF_TYPE_ALLOC_SRC_MASK . . . 713
4.179.7.12 PBUF_TYPE_ALLOC_SRC_MASK_APP_MAX . . . 713

4.179.7.13 PBUF_TYPE_ALLOC_SRC_MASK_APP_MIN . . . 713
4.179.7.14 PBUF_TYPE_FLAG_DATA_VOLATILE . . . 714
4.179.7.15 PBUF_TYPE_FLAG_STRUCT_DATA_CONTIGUOUS . . . 714
4.179.8 Typedef Documentation . . . 714
4.179.8.1 pbuf_free_custom_fn . . . 714
4.179.9 Function Documentation . . . 714
4.179.9.1 pbuf_add_header() . . . 714
4.179.9.2 pbuf_add_header_force() . . . 714
4.179.9.3 pbuf_clen() . . . 714
4.179.9.4 pbuf_dechain() . . . 715
4.179.9.5 pbuf_free_header() . . . 715
4.179.9.6 pbuf_header() . . . 715
4.179.9.7 pbuf_header_force() . . . 715
4.179.9.8 pbuf_remove_header() . . . 716
4.179.9.9 pbuf_strstr() . . . 716
4.180 src/include/lwip/priv/altcp_priv.h File Reference . . . 716
4.180.1 Functions . . . 716
4.180.2 Detailed Description . . . 716
4.180.3 Function Documentation . . . 717
4.180.3.1 altcp_alloc() . . . 717
4.180.3.2 altcp_free() . . . 717
4.181 src/include/lwip/priv/api_msg.h File Reference . . . 717
4.181.1 Data Structures . . . 717
4.181.2 Functions . . . 717
4.181.3 Detailed Description . . . 718
4.181.4 Function Documentation . . . 718
4.181.4.1 lwip_netconn_do_accepted() . . . 718
4.181.4.2 lwip_netconn_do_bind() . . . 718
4.181.4.3 lwip_netconn_do_bind_if() . . . 718
4.181.4.4 lwip_netconn_do_close() . . . 718
4.181.4.5 lwip_netconn_do_connect() . . . 718
4.181.4.6 lwip_netconn_do_delconn() . . . 719
4.181.4.7 lwip_netconn_do_disconnect() . . . 719
4.181.4.8 lwip_netconn_do_getaddr() . . . 719
4.181.4.9 lwip_netconn_do_gethostbyname() . . . 719
4.181.4.10 lwip_netconn_do_join_leave_group() . . . 719
4.181.4.11 lwip_netconn_do_join_leave_group_netif() . . . 719
4.181.4.12 lwip_netconn_do_listen() . . . 720
4.181.4.13 lwip_netconn_do_newconn() . . . 720

4.181.4.14 lwip_netconn_do_recv() . . . 720
4.181.4.15 lwip_netconn_do_send() . . . 720
4.181.4.16 lwip_netconn_do_write() . . . 720
4.181.4.17 netconn_alloc() . . . 720
4.181.4.18 netconn_free() . . . 721
4.182 src/include/lwip/priv/mem_priv.h File Reference . . . 721
4.182.1 Detailed Description . . . 721
4.183 src/include/lwip/priv/memp_priv.h File Reference . . . 721
4.183.1 Data Structures . . . 721
4.183.2 Functions . . . 721
4.183.3 Detailed Description . . . 721
4.183.4 Function Documentation . . . 722
4.183.4.1 memp_free_pool() . . . 722
4.183.4.2 memp_init_pool() . . . 722
4.183.4.3 memp_malloc_pool() . . . 722
4.184 src/include/lwip/priv/memp_std.h File Reference . . . 722
4.184.1 Detailed Description . . . 722
4.185 src/include/lwip/priv/nd6_priv.h File Reference . . . 722
4.185.1 Data Structures . . . 722
4.185.2 Detailed Description . . . 723
4.186 src/include/lwip/priv/raw_priv.h File Reference . . . 723
4.186.1 Typedefs . . . 723
4.186.2 Enumerations . . . 723
4.186.3 Functions . . . 723
4.186.4 Detailed Description . . . 723
4.186.5 Typedef Documentation . . . 723
4.186.5.1 raw_input_state_t . . . 723
4.186.6 Enumeration Type Documentation . . . 723
4.186.6.1 raw_input_state . . . 723
4.186.7 Function Documentation . . . 724
4.186.7.1 raw_input() . . . 724
4.186.7.2 raw_netif_ip_addr_changed() . . . 724
4.187 src/include/lwip/priv/sockets_priv.h File Reference . . . 724
4.187.1 Data Structures . . . 724
4.187.2 Macros . . . 724
4.187.3 Detailed Description . . . 724
4.187.4 Macro Definition Documentation . . . 725
4.187.4.1 SELWAIT_T . . . 725
4.188 src/include/lwip/priv/tcp_priv.h File Reference . . . 725

4.188.1 Macros 725
4.188.2 Functions 725
4.188.3 Variables 726
4.188.4 Detailed Description 727
4.188.5 Macro Definition Documentation 727
4.188.5.1 TCP_BUILD_MSS_OPTION 727
4.188.5.2 TCP_CHECKSUM_ON_COPY 727
4.188.5.3 tcp_do_output_nagle 727
4.188.5.4 TCP_OVERSIZE_DBGCHECK 727
4.188.5.5 TF_RESET 727
4.188.6 Function Documentation 728
4.188.6.1 tcp_abandon() 728
4.188.6.2 tcp_alloc() 728
4.188.6.3 tcp_eff_send_mss_netif() 728
4.188.6.4 tcp_enqueue_flags() 728
4.188.6.5 tcp_ext_arg_invoke_callbacks_passive_open() 728
4.188.6.6 tcp_fasttmr() 729
4.188.6.7 tcp_free() 729
4.188.6.8 tcp_init() 729
4.188.6.9 tcp_input() 729
4.188.6.10 tcp_keepalive() 729
4.188.6.11 tcp_netif_ip_addr_changed() 729
4.188.6.12 tcp_next_iss() 730
4.188.6.13 tcp_pcb_purge() 730
4.188.6.14 tcp_pcb_remove() 730
4.188.6.15 tcp_process_refused_data() 730
4.188.6.16 tcp_recv_null() 730
4.188.6.17 tcp_rexmit() 730
4.188.6.18 tcp_rexmit_fast() 731
4.188.6.19 tcp_rexmit_rto() 731
4.188.6.20 tcp_rexmit_rto_commit() 731
4.188.6.21 tcp_rexmit_rto_prepare() 731
4.188.6.22 tcp_rst() 731
4.188.6.23 tcp_rst_netif() 732
4.188.6.24 tcp_seg_copy() 732
4.188.6.25 tcp_seg_free() 732
4.188.6.26 tcp_segs_free() 732
4.188.6.27 tcp_send_empty_ack() 732
4.188.6.28 tcp_send_fin() 733

4.188.6.29 tcp_slowtmr() 733
4.188.6.30 tcp_split_unsent_seg() 733
4.188.6.31 tcp_timer_needed() 733
4.188.6.32 tcp_tmr() 733
4.188.6.33 tcp_txnow() 734
4.188.6.34 tcp_update_rcv_ann_wnd() 734
4.188.6.35 tcp_zero_window_probe() 734
4.188.7 Variable Documentation 734
4.188.7.1 tcp_active_pcbs 734
4.188.7.2 tcp_bound_pcbs 734
4.188.7.3 tcp_listen_pcbs 734
4.188.7.4 tcp_pcb_lists 734
4.188.7.5 tcp_tw_pcbs 735
4.189 src/include/lwip/priv/tcpip_priv.h File Reference 735
4.189.1 Functions 735
4.189.2 Detailed Description 735
4.189.3 Function Documentation 735
4.189.3.1 tcpip_api_call() 735
4.189.3.2 tcpip_send_msg_wait_sem() 735
4.190 src/include/lwip/prot/ethernet.h File Reference 736
4.190.1 Data Structures 736
4.190.2 Macros 736
4.190.3 Detailed Description 736
4.190.4 Macro Definition Documentation 736
4.190.4.1 ETH_ADDR 736
4.190.4.2 LL_IP4_MULTICAST_ADDR_0 736
4.190.4.3 LL_IP6_MULTICAST_ADDR_0 736
4.191 src/include/netif/ethernet.h File Reference 737
4.191.1 Macros 737
4.191.2 Functions 737
4.191.3 Detailed Description 737
4.191.4 Macro Definition Documentation 737
4.191.4.1 LWIP_ARP_FILTER_NETIF 737
4.192 src/include/lwip/prot/iana.h File Reference 737
4.192.1 Enumerations 737
4.192.2 Detailed Description 737
4.193 src/include/lwip/prot/ieee.h File Reference 738
4.193.1 Enumerations 738
4.193.2 Detailed Description 738

4.194 src/include/lwip/prot/tcp.h File Reference . . . 738
4.194.1 Detailed Description . . . 738
4.195 src/include/lwip/tcp.h File Reference . . . 738
4.195.1 Data Structures . . . 738
4.195.2 Macros . . . 738
4.195.3 Typedefs . . . 738
4.195.4 Functions . . . 739
4.195.5 Detailed Description . . . 739
4.195.6 Macro Definition Documentation . . . 740
4.195.6.1 TCP_PCB_COMMON . . . 740
4.195.7 Typedef Documentation . . . 740
4.195.7.1 tcp_accept_fn . . . 740
4.195.7.2 tcp_connected_fn . . . 740
4.195.7.3 tcp_err_fn . . . 741
4.195.7.4 tcp_extarg_callback_passive_open_fn . . . 741
4.195.7.5 tcp_extarg_callback_pcb_destroyed_fn . . . 741
4.195.7.6 tcp_poll_fn . . . 741
4.195.7.7 tcp_recv_fn . . . 742
4.195.7.8 tcp_sent_fn . . . 742
4.195.8 Function Documentation . . . 742
4.195.8.1 tcp_setprio() . . . 742
4.196 src/include/lwip/prot/udp.h File Reference . . . 742
4.196.1 Detailed Description . . . 742
4.197 src/include/lwip/udp.h File Reference . . . 743
4.197.1 Data Structures . . . 743
4.197.2 Typedefs . . . 743
4.197.3 Functions . . . 743
4.197.4 Detailed Description . . . 744
4.197.5 Typedef Documentation . . . 744
4.197.5.1 udp_recv_fn . . . 744
4.197.6 Function Documentation . . . 744
4.197.6.1 udp_init() . . . 744
4.197.6.2 udp_input() . . . 744
4.197.6.3 udp_netif_ip_addr_changed() . . . 744
4.198 src/include/lwip/raw.h File Reference . . . 745
4.198.1 Data Structures . . . 745
4.198.2 Typedefs . . . 745
4.198.3 Functions . . . 745
4.198.4 Detailed Description . . . 745

4.198.5 Typedef Documentation 746
4.198.5.1 raw_recv_fn 746
4.199 src/include/lwip/sockets.h File Reference 746
4.199.1 Macros 746
4.199.2 Functions 746
4.199.3 Detailed Description 746
4.199.4 Macro Definition Documentation 746
4.199.4.1 LWIP_TIMEVAL_PRIVATE 746
4.199.5 Function Documentation 747
4.199.5.1 lwip_fcntl() 747
4.199.5.2 lwip_listen() 747
4.199.5.3 lwip_shutdown() 747
4.199.5.4 lwip_socket_thread_cleanup() 747
4.199.5.5 lwip_socket_thread_init() 747
4.200 src/include/lwip/stats.h File Reference 747
4.200.1 Data Structures 747
4.200.2 Functions 748
4.200.3 Variables 748
4.200.4 Detailed Description 748
4.200.5 Function Documentation 748
4.200.5.1 stats_init() 748
4.200.6 Variable Documentation 748
4.200.6.1 lwip_stats 748
4.201 src/include/lwip/sys.h File Reference 748
4.201.1 Macros 748
4.201.2 Typedefs 749
4.201.3 Functions 749
4.201.4 Detailed Description 750
4.201.5 Macro Definition Documentation 750
4.201.5.1 LWIP_COMPAT_MUTEX 750
4.201.5.2 SYS_ARCH_TIMEOUT 750
4.201.5.3 SYS_MBOX_EMPTY 750
4.201.5.4 sys_mbox_set_invalid_val 750
4.201.5.5 sys_mbox_tryfetch 750
4.201.5.6 sys_mbox_valid_val 750
4.201.5.7 sys_sem_set_invalid_val 750
4.201.5.8 sys_sem_valid_val 750
4.201.5.9 sys_sem_wait 751
4.201.6 Typedef Documentation 751

4.201.6.1 lwip_thread_fn . . . 751
4.201.7 Function Documentation . . . 751
4.201.7.1 sys_jiffies() . . . 751
4.202 src/include/lwip/tcpbase.h File Reference . . . 751
4.202.1 Detailed Description . . . 751
4.203 src/include/lwip/tcpip.h File Reference . . . 751
4.203.1 Macros . . . 751
4.203.2 Typedefs . . . 751
4.203.3 Functions . . . 752
4.203.4 Variables . . . 752
4.203.5 Detailed Description . . . 752
4.203.6 Macro Definition Documentation . . . 752
4.203.6.1 LOCK_TCPIP_CORE . . . 752
4.203.6.2 UNLOCK_TCPIP_CORE . . . 752
4.203.7 Typedef Documentation . . . 752
4.203.7.1 tcpip_callback_fn . . . 752
4.203.7.2 tcpip_init_done_fn . . . 753
4.203.8 Function Documentation . . . 753
4.203.8.1 mem_free_callback() . . . 753
4.203.8.2 pbuf_free_callback() . . . 753
4.203.8.3 tcpip_callback_wait() . . . 753
4.203.8.4 tcpip_inpkt() . . . 753
4.203.9 Variable Documentation . . . 754
4.203.9.1 lock_tcpip_core . . . 754
4.204 src/include/lwip/timeouts.h File Reference . . . 754
4.204.1 Data Structures . . . 754
4.204.2 Macros . . . 754
4.204.3 Typedefs . . . 754
4.204.4 Functions . . . 754
4.204.5 Variables . . . 754
4.204.6 Detailed Description . . . 755
4.204.7 Macro Definition Documentation . . . 755
4.204.7.1 SYS_TIMEOUTS_SLEEPTIME_INFINITE . . . 755
4.204.8 Typedef Documentation . . . 755
4.204.8.1 lwip_cyclic_timer_handler . . . 755
4.204.8.2 sys_timeout_handler . . . 755
4.204.9 Function Documentation . . . 755
4.204.9.1 sys_restart_timeouts() . . . 755
4.204.9.2 sys_timeout() . . . 755

4.204.9.3 sys_timeouts_init() 755
4.204.9.4 sys_timeouts_sleeptime() 756
4.204.9.5 sys_untimeout() 756
4.204.10Variable Documentation 756
4.204.10.1lwip_cyclic_timers 756
4.204.10.2lwip_num_cyclic_timers 756
4.205src/include/netif/bridgeif.h File Reference 756
4.205.1 Data Structures 756
4.205.2 Macros 756
4.205.3 Typedefs 757
4.205.4 Functions 757
4.205.5 Detailed Description 757
4.206src/include/netif/bridgeif_opts.h File Reference 757
4.206.1 Macros 757
4.206.2 Detailed Description 757
4.207src/include/netif/ieee802154.h File Reference 757
4.207.1 Data Structures 757
4.207.2 Detailed Description 758
4.208src/include/netif/lowpan6.h File Reference 758
4.208.1 Macros 758
4.208.2 Functions 758
4.208.3 Detailed Description 758
4.208.4 Macro Definition Documentation 758
4.208.4.1 LOWPAN6_TMR_INTERVAL 758
4.208.5 Function Documentation 758
4.208.5.1 lowpan6_calc_crc() 758
4.208.5.2 lowpan6_tmr() 759
4.209src/include/netif/lowpan6_ble.h File Reference 759
4.209.1 Functions 759
4.209.2 Detailed Description 759
4.209.3 Function Documentation 759
4.209.3.1 rfc7668_set_local_addr_eui64() 759
4.209.3.2 rfc7668_set_local_addr_mac48() 759
4.209.3.3 rfc7668_set_peer_addr_eui64() 760
4.209.3.4 rfc7668_set_peer_addr_mac48() 760
4.209.3.5 tcpip_rfc7668_input() 760
4.210src/include/netif/lowpan6_common.h File Reference 760
4.210.1 Data Structures 760
4.210.2 Detailed Description 760

4.211 src/include/netif/lowpan6_opts.h File Reference . . . 760
4.211.1 Macros . . . 761
4.211.2 Detailed Description . . . 761
4.211.3 Macro Definition Documentation . . . 761
4.211.3.1 LWIP_6LOWPAN_802154_HW_CRC . . . 761
4.211.3.2 LWIP_6LOWPAN_CALC_CRC . . . 761
4.211.3.3 LWIP_6LOWPAN_INFER_SHORT_ADDRESS . . . 761
4.211.3.4 LWIP_6LOWPAN_IPHC . . . 761
4.211.3.5 LWIP_6LOWPAN_NUM_CONTEXTS . . . 762
4.211.3.6 LWIP_LOWPAN6_802154_DEBUG . . . 762
4.211.3.7 LWIP_LOWPAN6_DEBUG . . . 762
4.211.3.8 LWIP_LOWPAN6_DECOMPRESSION_DEBUG . . . 762
4.211.3.9 LWIP_LOWPAN6_IP_COMPRESSED_DEBUG . . . 762
4.211.3.10 LWIP_RFC7668_IP_UNCOMPRESSED_DEBUG . . . 762
4.211.3.11 LWIP_RFC7668_LINUX_WORKAROUND_PUBLIC_ADDRESS . . . 762
4.212 src/include/netif/ppp/pppol2tp.h File Reference . . . 762
4.212.1 Detailed Description . . . 762
4.213 src/include/netif/ppp/pppos.h File Reference . . . 763
4.213.1 Detailed Description . . . 763
4.214 src/include/netif/slipif.h File Reference . . . 763
4.214.1 Macros . . . 763
4.214.2 Functions . . . 763
4.214.3 Detailed Description . . . 763
4.214.4 Macro Definition Documentation . . . 763
4.214.4.1 SLIP_RX_QUEUE . . . 763
4.214.4.2 SLIP_USE_RX_THREAD . . . 763
4.215 src/include/netif/zepif.h File Reference . . . 764
4.215.1 Data Structures . . . 764
4.215.2 Functions . . . 764
4.215.3 Detailed Description . . . 764
4.216 src/netif/bridgeif.c File Reference . . . 764
4.216.1 Functions . . . 764
4.216.2 Detailed Description . . . 764
4.217 src/netif/bridgeif_fdb.c File Reference . . . 764
4.217.1 Functions . . . 764
4.217.2 Detailed Description . . . 765
4.218 src/netif/ethernet.c File Reference . . . 765
4.218.1 Functions . . . 765
4.218.2 Detailed Description . . . 765

4.219 src/netif/lowpan6.c File Reference . . . 765
4.219.1 Data Structures . . . 765
4.219.2 Functions . . . 765
4.219.3 Detailed Description . . . 766
4.219.4 Function Documentation . . . 766
4.219.4.1 lowpan6_calc_crc() . . . 766
4.219.4.2 lowpan6_tmr() . . . 766
4.220 src/netif/lowpan6_ble.c File Reference . . . 766
4.220.1 Functions . . . 766
4.220.2 Detailed Description . . . 766
4.220.3 Function Documentation . . . 767
4.220.3.1 rfc7668_set_local_addr_eui64() . . . 767
4.220.3.2 rfc7668_set_local_addr_mac48() . . . 767
4.220.3.3 rfc7668_set_peer_addr_eui64() . . . 767
4.220.3.4 rfc7668_set_peer_addr_mac48() . . . 767
4.220.3.5 tcpip_rfc7668_input() . . . 767
4.221 src/netif/lowpan6_common.c File Reference . . . 767
4.221.1 Detailed Description . . . 767
4.222 src/netif/ppp/pppapi.c File Reference . . . 768
4.222.1 Detailed Description . . . 768
4.223 src/netif/ppp/pppol2tp.c File Reference . . . 768
4.223.1 Detailed Description . . . 768
4.224 src/netif/ppp/pppos.c File Reference . . . 768
4.224.1 Detailed Description . . . 768
4.225 src/netif/slipif.c File Reference . . . 768
4.225.1 Macros . . . 768
4.225.2 Functions . . . 768
4.225.3 Detailed Description . . . 769
4.225.4 Macro Definition Documentation . . . 769
4.225.4.1 SLIP_MAX_SIZE . . . 769
4.225.4.2 SLIP_SIO_SPEED . . . 769
4.226 src/netif/zepif.c File Reference . . . 769
4.226.1 Macros . . . 769
4.226.2 Functions . . . 769
4.226.3 Macro Definition Documentation . . . 769
4.226.3.1 ZEPIF_LOOPBACK . . . 769

List of Tables

Chapter 1

Overview

```
INTRODUCTION

lwIP is a small independent implementation of the TCP/IP protocol suite.

The focus of the lwIP TCP/IP implementation is to reduce the RAM usage
while still having a full scale TCP. This making lwIP suitable for use
in embedded systems with tens of kilobytes of free RAM and room for
around 40 kilobytes of code ROM.

lwIP was originally developed by Adam Dunkels at the Computer and Networks
Architectures (CNA) lab at the Swedish Institute of Computer Science (SICS)
and is now developed and maintained by a worldwide network of developers.

FEATURES

  * IP (Internet Protocol, IPv4 and IPv6) including packet forwarding over
    multiple network interfaces
  * ICMP (Internet Control Message Protocol) for network maintenance and debugging
  * IGMP (Internet Group Management Protocol) for multicast traffic management
  * MLD (Multicast listener discovery for IPv6). Aims to be compliant with
    RFC 2710. No support for MLDv2
  * ND (Neighbor discovery and stateless address autoconfiguration for IPv6).
    Aims to be compliant with RFC 4861 (Neighbor discovery) and RFC 4862
    (Address autoconfiguration)
  * DHCP, AutoIP/APIPA (Zeroconf), ACD (Address Conflict Detection)
    and (stateless) DHCPv6
  * UDP (User Datagram Protocol) including experimental UDP-lite extensions
  * TCP (Transmission Control Protocol) with congestion control, RTT estimation
    fast recovery/fast retransmit and sending SACKs
  * raw/native API for enhanced performance
  * Optional Berkeley-like socket API
  * TLS: optional layered TCP ("altcp") for nearly transparent TLS for any
    TCP-based protocol (ported to mbedTLS) (see changelog for more info)
  * PPPoS and PPPoE (Point-to-point protocol over Serial/Ethernet)
  * DNS (Domain name resolver incl. mDNS)
  * 6LoWPAN (via IEEE 802.15.4, BLE or ZEP)


APPLICATIONS

  * HTTP server with SSI and CGI (HTTPS via altcp)
  * SNMPv2c agent with MIB compiler (Simple Network Management Protocol), v3 via ↩
     altcp
  * SNTP (Simple network time protocol)
  * NetBIOS name service responder
  * MDNS (Multicast DNS) responder
  * iPerf server implementation
  * MQTT client (TLS support via altcp)


LICENSE

lwIP is freely available under a BSD license.


DEVELOPMENT

lwIP has grown into an excellent TCP/IP stack for embedded devices,
and developers using the stack often submit bug fixes, improvements,
and additions to the stack to further increase its usefulness.

Development of lwIP is hosted on Savannah, a central point for
software development, maintenance and distribution. Everyone can
help improve lwIP by use of Savannah's interface, Git and the
mailing list. A core team of developers will commit changes to the
Git source tree.

The lwIP TCP/IP stack is maintained in the 'src' directory and
contributions (such as platform ports and applications) are in
the 'contrib' directory.

See doc/savannah.txt for details on Git server access for users and
developers.

The current Git tree is web-browsable:
  https://git.savannah.gnu.org/cgit/lwip.git

Submit patches and bugs via the lwIP project page:
  https://savannah.nongnu.org/projects/lwip/

Continuous integration builds (GCC, clang):
  https://github.com/lwip-tcpip/lwip/actions


DOCUMENTATION

Self documentation of the source code is regularly extracted from the current
Git sources and is available from this web page:
  https://www.nongnu.org/lwip/

Also, there are mailing lists you can subscribe at
  https://savannah.nongnu.org/mail/?group=lwip
plus searchable archives:
  https://lists.nongnu.org/archive/html/lwip-users/
  https://lists.nongnu.org/archive/html/lwip-devel/

There is a wiki about lwIP at
  https://lwip.wikia.com/wiki/LwIP_Wiki
```

```
You might get questions answered there, but unfortunately, it is not as
well maintained as it should be.

lwIP was originally written by Adam Dunkels:
  http://dunkels.com/adam/

Reading Adam's papers, the files in docs/, browsing the source code
documentation and browsing the mailing list archives is a good way to
become familiar with the design of lwIP.

Adam Dunkels <adam@sics.se>
Leon Woestenberg <leon.woestenberg@gmx.net>
```

1.1 Upgrading

```
This file lists major changes between release versions that require
ports or applications to be changed. Use it to update a port or an
application written for an older version of lwIP to correctly work
with newer versions.


(git master)

  * [Enter new changes just after this line - do not remove this line]
  * The eth_addr_cmp and ip_addr_cmp set of functions have been renamed to  ←
     eth_addr_eq, ip_addr_eq
    and so on, since they return non-zero on equality. Macros for the old names  ←
       exist.
  * The sio_write function used by PPP now takes the data argument as const.
  * The prev field in the snmp_varbind struct has been removed.

(2.2.0)

  ++ Repository changes:

  * The contrib repository has been added into the main repository in the  ←
     subdirectory 'contrib'
    (the old contrib repository remains online for reference but is not used any  ←
       more)

(2.1.0)

  ++ Application changes:

  * Use the new altcp API for seamless TLS integration into existing TCP  ←
     applications (see changelog)
  * TCP only kills existing connections with a LOWER priority than the one  ←
     currently being opened.
    Previous implementations also kill existing connections of the SAME priority.
  * ip4_route_src: parameter order is reversed: ip4_route_src(dest, src) ->  ←
     ip4_route_src(src, dest)
    to make parameter order consistent with other ip*_route*() functions.
    Same also applies to LWIP_HOOK_IP4_ROUTE_SRC() parameter order.
  * pbuf API: pbuf->type (an u8_t holding the enum 'pbuf_type') has changed to  ←
     only hold a
    description of the pbuf (e.g. data following pbuf struct, data volatile,  ↩
       allocation
    source heap/pool/etc.). As a consequence, applications can't test pbuf->type  ↩
       any more.
    Use pbuf_match_type(pbuf, type) instead.
  * socket API: according to the standard, SO_ERROR now only returns asynchronous  ↩
     errors.
    All other/normal/synchronous errors are (and always were) available via 'errno ↩
       '.
    LWIP_SOCKET_SET_ERRNO has been removed - 'errno' is always set - and required!
  * httpd LWIP_HTTPD_CGI_SSI: httpd_cgi_handler() has an additional parameter " ↩
     struct fs_file *"

  ++ Port changes:

  * tcpip_trycallback() was renamed to tcpip_callbackmsg_trycallback() to avoid  ↩
     confusion
    with tcpip_try_callback()
  * compatibility headers: moved from 'src/include/posix' to 'src/include/compat/ ↩
     posix',
    'src/include/compat/stdc' etc.
  * The IPv6 implementation now supports address scopes. (See LWIP_IPV6_SCOPES  ↩
     documentation
    and ip6_zone.h for more documentation)
  * LWIP_HOOK_DHCP_APPEND_OPTIONS() has changed, see description in opt.h ( ↩
     options_out_len is not
    available in struct dhcp any more)
  * Added debug helper asserts to ensure threading/locking requirements are met ( ↩
     define
    LWIP_MARK_TCPIP_THREAD() and LWIP_ASSERT_CORE_LOCKED()).
  * Added sys_mbox_trypost_fromisr() and tcpip_callbackmsg_trycallback_fromisr()
    These can be used to post preallocated messages from an ISR to the tcpip  ↩
       thread
    (e.g. when using FreeRTOS)

(2.0.2)

  ++ Application changes:

  * slipif: The way to pass serial port number has changed. netif->num is not
    supported any more, netif->state is interpreted as an u8_t port number now
    (it's not a POINTER to an u8_t any more!)

(2.0.1)

  ++ Application changes:

  * UDP does NOT receive multicast traffic from ALL netifs on an UDP PCB bound to  ↩
     a specific
    netif any more. Users need to bind to IP_ADDR_ANY to receive multicast traffic ↩
        and compare
    ip_current_netif() to the desired netif for every packet.
    See bug #49662 for an explanation.

(2.0.0)

  ++ Application changes:
```

```
* Changed netif "up" flag handling to be an administrative flag (as opposed to ↩
   the previous meaning of
  "ip4-address-valid", a netif will now not be used for transmission if not up) ↩
     -> even a DHCP netif
  has to be set "up" before starting the DHCP client
* Added IPv6 support (dual-stack or IPv4/IPv6 only)
* Changed ip_addr_t to be a union in dual-stack mode (use ip4_addr_t where ↩
   referring to IPv4 only).
* Major rewrite of SNMP (added MIB parser that creates code stubs for custom ↩
   MIBs);
  supports SNMPv2c (experimental v3 support)
* Moved some core applications from contrib repository to src/apps (and include/ ↩
   lwip/apps)

+++ Raw API:
  * Changed TCP listen backlog: removed tcp_accepted(), added the function pair ↩
     tcp_backlog_delayed()/
    tcp_backlog_accepted() to explicitly delay backlog handling on a connection ↩
       pcb

+++ Socket API:
  * Added an implementation for posix sendmsg()
  * Added LWIP_FIONREAD_LINUXMODE that makes ioctl/FIONREAD return the size of ↩
     the next pending datagram

++ Port changes

+++ new files:
  * MANY new and moved files!
  * Added src/Filelists.mk for use in Makefile projects
  * Continued moving stack-internal parts from abc.h to abc_priv.h in sub-folder ↩
      "priv"
    to let abc.h only contain the actual application programmer's API

+++ sys layer:
  * Made LWIP_TCPIP_CORE_LOCKING==1 the default as it usually performs better ↩
     than
    the traditional message passing (although with LWIP_COMPAT_MUTEX you are ↩
       still
    open to priority inversion, so this is not recommended any more)
  * Added LWIP_NETCONN_SEM_PER_THREAD to use one "op_completed" semaphore per ↩
     thread
    instead of using one per netconn (these semaphores are used even with core ↩
       locking
    enabled as some longer lasting functions like big writes still need to delay ↩
       )
  * Added generalized abstraction for itoa(), strnicmp(), stricmp() and strnstr ↩
     ()
    in def.h (to be overridden in cc.h) instead of config
    options for netbiosns, httpd, dns, etc. ...
  * New abstraction for hton* and ntoh* functions in def.h.
    To override them, use the following in cc.h:
    #define lwip_htons(x) <your_htons>
    #define lwip_htonl(x) <your_htonl>

+++ new options:
    * TODO

  +++ new pools:
    * Added LWIP_MEMPOOL_* (declare/init/alloc/free) to declare private memp ↩
       pools
      that share memp.c code but do not have to be made global via lwippools.h
    * Added pools for IPv6, MPU_COMPATIBLE, dns-api, netif-api, etc.
    * added hook LWIP_HOOK_MEMP_AVAILABLE() to get informed when a memp pool was ↩
       empty and an item
      is now available

  * Signature of LWIP_HOOK_VLAN_SET macro was changed

  * LWIP_DECLARE_MEMORY_ALIGNED() may be used to declare aligned memory buffers ( ↩
     mem/memp)
    or to move buffers to dedicated memory using compiler attributes

  * Standard C headers are used to define sized types and printf formatters
    (disable by setting LWIP_NO_STDINT_H=1 or LWIP_NO_INTTYPES_H=1 if your ↩
       compiler
    does not support these)


  ++ Major bugfixes/improvements

  * Added IPv6 support (dual-stack or IPv4/IPv6 only)
  * Major rewrite of PPP (incl. keep-up with apache pppd)
    see doc/ppp.txt for an upgrading how-to
  * Major rewrite of SNMP (incl. MIB parser)
  * Fixed timing issues that might have lead to losing a DHCP lease
  * Made rx processing path more robust against crafted errors
  * TCP window scaling support
  * modification of api modules to support FreeRTOS-MPU (don't pass stack-pointers ↩
      to other threads)
  * made DNS client more robust
  * support PBUF_REF for RX packets
  * LWIP_NETCONN_FULLDUPLEX allows netconn/sockets to be used for reading/writing ↩
     from separate
    threads each (needs LWIP_NETCONN_SEM_PER_THREAD)
  * Moved and reordered stats (mainly memp/mib2)

(1.4.0)

  ++ Application changes:

  * Replaced struct ip_addr by typedef ip_addr_t (struct ip_addr is kept for
    compatibility to old applications, but will be removed in the future).

  * Renamed mem_realloc() to mem_trim() to prevent confusion with realloc()

  +++ Raw API:
    * Changed the semantics of tcp_close() (since it was rather a
      shutdown before): Now the application does *NOT* get any calls to the recv
      callback (aside from NULL/closed) after calling tcp_close()

    * When calling tcp_abort() from a raw API TCP callback function,
      make sure you return ERR_ABRT to prevent accessing unallocated memory.
     (ERR_ABRT now means the applicaiton has called tcp_abort!)

  +++ Netconn API:
    * Changed netconn_receive() and netconn_accept() to return
      err_t, not a pointer to new data/netconn.

  +++ Socket API:
    * LWIP_SO_RCVTIMEO: when accept() or recv() time out, they
      now set errno to EWOULDBLOCK/EAGAIN, not ETIMEDOUT.

    * Added a minimal version of posix fctl() to have a
      standardised way to set O_NONBLOCK for nonblocking sockets.

  +++ all APIs:
    * correctly implemented SO(F)_REUSEADDR

  ++ Port changes

  +++ new files:

    * Added 4 new files: def.c, timers.c, timers.h, tcp_impl.h:

    * Moved stack-internal parts of tcp.h to tcp_impl.h, tcp.h now only contains
      the actual application programmer's API

    * Separated timer implementation from sys.h/.c, moved to timers.h/.c;
      Added timer implementation for NO_SYS==1, set NO_SYS_NO_TIMERS==1 if you
      still want to use your own timer implementation for NO_SYS==0 (as before).

  +++ sys layer:

    * Converted mbox- and semaphore-functions to take pointers to sys_mbox_t/
      sys_sem_t;

    * Converted sys_mbox_new/sys_sem_new to take pointers and return err_t;

    * Added Mutex concept in sys_arch (define LWIP_COMPAT_MUTEX to let sys.h use
      binary semaphores instead of mutexes - as before)

  +++ new options:

     * Don't waste memory when chaining segments, added option TCP_OVERSIZE to
       prevent creating many small pbufs when calling tcp_write with many small
       blocks of data. Instead, pbufs are allocated larger than needed and the
       space is used for later calls to tcp_write.

     * Added LWIP_NETIF_TX_SINGLE_PBUF to always copy to try to create single  ↩
        pbufs
       in tcp_write/udp_send.

    * Added an additional option LWIP_ETHERNET to support ethernet without ARP
      (necessary for pure PPPoE)

    * Add MEMP_SEPARATE_POOLS to place memory pools in separate arrays. This may
      be used to place these pools into user-defined memory by using external
      declaration.
    * Added TCP_SNDQUEUELOWAT corresponding to TCP_SNDLOWAT

  +++ new pools:

     * Netdb uses a memp pool for allocating memory when getaddrinfo() is called,
       so MEMP_NUM_NETDB has to be set accordingly.

     * DNS_LOCAL_HOSTLIST_IS_DYNAMIC uses a memp pool instead of the heap, so
       MEMP_NUM_LOCALHOSTLIST has to be set accordingly.

     * Snmp-agent uses a memp pools instead of the heap, so MEMP_NUM_SNMP_* have
       to be set accordingly.

     * PPPoE uses a MEMP pool instead of the heap, so MEMP_NUM_PPPOE_INTERFACES
       has to be set accordingly

  * Integrated loopif into netif.c - loopif does not have to be created by the
    port any more, just define LWIP_HAVE_LOOPIF to 1.

  * Added define LWIP_RAND() for lwip-wide randomization (needs to be defined
    in cc.h, e.g. used by igmp)

  * Added printf-formatter X8_F to printf u8_t as hex

  * The heap now may be moved to user-defined memory by defining
    LWIP_RAM_HEAP_POINTER as a void pointer to that memory's address

  * added autoip_set_struct() and dhcp_set_struct() to let autoip and dhcp work
    with user-allocated structs instead of calling mem_malloc

  * Added const char* name to mem- and memp-stats for easier debugging.

  * Calculate the TCP/UDP checksum while copying to only fetch data once:
    Define LWIP_CHKSUM_COPY to a memcpy-like function that returns the checksum

  * Added SO_REUSE_RXTOALL to pass received UDP broadcast/multicast packets to
    more than one pcb.

  * Changed the semantics of ARP_QUEUEING==0: ARP_QUEUEING now cannot be turned
    off any more, if this is set to 0, only one packet (the most recent one) is
    queued (like demanded by RFC 1122).


  ++ Major bugfixes/improvements

  * Implemented tcp_shutdown() to only shut down one end of a connection
  * Implemented shutdown() at socket- and netconn-level
  * Added errorset support to select() + improved select speed overhead
  * Merged pppd to v2.3.11 (including some backported bugfixes from 2.4.x)
  * Added timer implementation for NO_SYS==1 (may be disabled with  ←
     NO_SYS_NO_TIMERS==1
  * Use macros defined in ip_addr.h to work with IP addresses
  * Implemented many nonblocking socket/netconn functions
  * Fixed ARP input processing: only add a new entry if a request was directed as  ←
     us
  * mem_realloc() to mem_trim() to prevent confusion with realloc()
  * Some improvements for AutoIP (don't route/forward link-local addresses, don't ↩
     break
    existing connections when assigning a routable address)
  * Correctly handle remote side overrunning our rcv_wnd in ooseq case
  * Removed packing from ip_addr_t, the packed version is now only used in ↩
     protocol headers
  * Corrected PBUF_POOL_BUFSIZE for ports where ETH_PAD_SIZE > 0
  * Added support for static ARP table entries

(STABLE-1.3.2)

  * initial version of this file
```

1.2 Changelog

2.1.0

- Support TLS via new Application layered TCP Introduction connection API (https, smtps, mqtt over TLS)
- Switch to cmake as the main build system (Makefile file lists are still maintained for now)
- Improve IPv6 support: support address scopes, support stateless DHCPv6, bugfixes
- Add debug helper asserts to ensure threading/locking requirements are met
- Add sys_mbox_trypost_fromisr() and tcpip_callbackmsg_trycallback_fromisr() (for FreeRTOS, mainly)
- socket API: support poll(), sendmsg() and recvmsg(); fix problems on close

Detailed Changelog

```
HISTORY
 * These are only the most important changes. For a full list, use git log:
   http://git.savannah.nongnu.org/cgit/lwip.git

(git master)

  * [Enter new changes just after this line - do not remove this line]

  ++ Bugfixes:

  2025-06-03: Simon Goldschmidt
  * ip4_frag/ip6_frag: fix potential NULL-pointer access on memory errors

(STABLE-2.2.1):

  ++ New features:

  2023-10-11: Faidon Liambotis
  * Add MEM_CUSTOM_ALLOCATOR and make LIBC a subset of it

  2023-09-29: Jiri Findejs
  * mqtt: support binary Will Message

  ++ Bugfixes:
```

```
  2024-02-19: Simon Goldschmidt
  * tcpip: fix that TCPIP_CORE_LOCK is not released for LWIP_TIMERS==0

  2024-01-09: Simon Goldschmidt
  * snmp v3, ppp: prevent possible timing attacks by using a constant-runtime- ↩
     memcmp when
    checking credentials

  2023-01-04: Simon Goldschmidt
  * makefsdata: update tinydir.h to newest version (1.2.6)

  2023-10-12: Borys Szefler
  * dhcp: fix memory corruption when LWIP_DHCP_MAX_DNS_SERVERS  > DNS_MAX_SERVERS

  2023-10-11: Mazakazu
  * sockets: fix socket leak when using setsockopt/getsockopt hook with  ↩
     LWIP_NETCONN_FULLDUPLEX==1

  2023-10-10: Simon Goldschmidt
  * sockets: fix bug #63898: allow socket option IPV6_CHECKSUM for both  ↩
     IPPROTO_IPV6 and IPPROTO_RAW

  2023-10-10: Simon Goldschmidt
  * ipv6: fix ip6_current_header() after reassembly

  2023-10-07: Simon Goldschmidt
  * ipv6 reassembly: fix detecting holes in reassembled packets

  2023-10-04: Simon Goldschmidt
  * apps: http client: improve the HTTP client; ensure connection settings are  ↩
     passed

  2023-10-03: Simon Goldschmidt
  * ipv6: frag: fix bogus icmp6 response on reassembly timeout

  2023-09-28: Szabolcs Szekelyi
  * httpd: ensure headers are parsed case-insensitive

  2023-09-28: Erik Ekman
  * Fix ND6 Router Advertisement parsing when NETIF_MAX_HWADDR_LEN is above 6.

  2023-09-27: Hardy Griech
  * Fix iperf byte counting mode

(STABLE-2.2.0):

  2018-10-02: Dirk Ziegelmeier
  * Integrate contrib repository into main lwIP repository

  ++ New features:

  2022-04-05: David Cermak
  * contrib/addons: Add example of using DHCP extra options hooks

  2023-05-11: David Cermak
  * dhcp: Add macro for appending extra client's request options
```

```
2023-05-11: xueyunfei
* dhcp: Enable custom config for timeouts, thresholds, backoff time

2021-04-26
* test/unit: added more unit tests

2020-03-27: Simon Goldschmidt
* test/fuzz: improve fuzz test

2019-12-11: Simon Goldschmidt
* ip6addr_aton: support scoped address strings (via '%')

2019-08-28: Joan Lledó
* Contrib: Add kFreeBSD to the Unix port

2019-07-14: Joan Lledó
* Unix port: improve support for the Hurd

++ Bugfixes:

2019-12-11: David Girault
* altcp_tls: support for saving/restoring session information

2018-11-16: Craig McQUeen
* dns: allow a DNS look-up with a trailing dot in the name

2018-10-19: Timmy Brolin
* Add outgoing VLAN PCP support for Ethernet level QoS

2018-10-08: Ben Wijen
* apps/tftp: added TFTP client

2018-10-04: Jasper Verschueren
* Implement IPv4 ACD (Address Conflict Detection)

2023-05-10
* altcp_tls_mbedtls: note which version of mbedtls we are compatible to
* altcp_tls_mbedtls: multiple compatibility fixes

2023-04-26: Jan Breuer, Harrold Spier, Ognjen Bjelica, Dirk Ziegelmeier, Simon  ↩
    Goldschmidt
* apps/snmp: multiple fixes and improvements to snmp

2022-01-12: Simon Goldschmidt
* httpd: clean up custom file handling

2021-11-25: quanjia
* ping: fix sockaddr len in ping_send() for PING_USE_SOCKETS==1

2021-11-12: Bas Prins
* http_client: reset timeout when receiving data

2020-07-07: Erik Ekman
* Rename IP and Ethernet equality checkers from _cmp to _eq

2020-03-05: Simon Goldschmidt
* tcp: tighten up checks for received SYN
```

```
  2020-01-30: Simon Goldschmidt, David Girault, David J. Fiddes, Tom Ferrin
  * apps/sntp: multiple fixes and improvements for sntp

  2020-01-30: Simon Goldschmidt
  * ip_forward: fix IPv4 forwarding with multiple netifs/offloading

  2019-06-11: David Girault, Giuseppe Modugno
  * apps/mqtt: multiple fixes for mqtt

  2019-05-19: Joan Lledó
  * New function tcpip_callback_wait()
    Call a function inside the tcpip thread and block the calling thread until
    the callback finishes

  2018-08-15: Jasper Verschueren, David Girault, Our Air Quality
  * apps/mdns: greatly improved the mdns client

(STABLE-2.1.2):

  ++ Bugfixes:

  2018-11-21: Jens Nielsen
  * netbiosns.c: fix expecting too large packet (bug #55069)

  2018-11-19: Dirk Ziegelmeier
  * smtp.c: fix compiling with strict C compatibility because of strnlen (bug ↩
     #55034)

  2018-11-12: Simon Goldschmidt
  * tcp.c: fix overflow check in tcp_recved triggering invalid assertion (bug ↩
     #55015)

  2018-11-12: Simon Goldschmidt
  * tcp.c: fix a bug in sending RST segments (sent from port 0)

(STABLE-2.1.1):

  ++ Bugfixes:

  2018-11-01: Joan Lledó
  * sockets.c: fix bad assertion in lwip_poll_dec_sockets_used() (bug #54933)

  2018-11-01: Dirk Ziegelmeier
  * ip4.c: don’t send 127.* to default netif (bug #54670)

  2018-10-23: David Girault
  * altcp_tls_mbedtls.c: fix use-after free (bug #54774)

  2018-10-23: Ognjen Bjelica, Dirk Ziegelmeier
  * snmp_scalar.c: Avoid NULL pointer dereference (bug #54886)

  2018-10-23: Simon Goldschmidt
  * Fix missing standard includes in multiple files

  2018-10-17: Ivan Warren
  * def.h: fix casting htonX and ntohX to u16_t (bug #54850)
  2018-10-12: Simon Goldschmidt
  * Revert "tcp_abandon: no need to buffer pcb->local_port" (fix that source port ↩
     was 0 for RST
    called when aborting a connection)

  2018-10-11: Jonas Rabenstein
  * tcp.c: tcp_recved: check for overflow and warn about too big values (patch  ↩
     #9699)

  2018-10-06: Joan Lledó
  * sockets.c: alloc_socket(): Check for LWIP_SOCKET_POLL when setting select-
    related variables (patch #9696)

  2018-10-04: Spencer
  * tcp.c: Update prev pointer when skipping entries in tcp_slowtmr (patch #9694)

  2018-09-27: Martine Lenders
  * lowpan6.c: Fix IEEE 802.15.4 address setting (bug #54749)

(STABLE-2.1.0):

  ++ New features:

  2018-06-17: Simon Goldschmidt
  * lwiperf: implemented iPerf client mode

  2018-04-23: Dirk Ziegelmeier
  * added cmake build files

  2018-03-04: Ray Abram
  * netbios responder: respond to '*' queries

  2018-02-23: Benjamin Aigner
  * 6lowpan: add 6lowpan-over-BLE netif (based on existing 6lowpan netif)

  2018-02-22: Simon Goldschmidt
  * ipv6: add support for stateless DHCPv6 (to get DNS servers in SLAAC nets)

  2018-02-16: Simon Goldschmidt
  * add raw API http(s) client (with proxy support)

  2018-02-01: Simon Goldschmidt
  * tcp: add hooks to implement additional socket options

  2018-02-01: Simon Goldschmidt
  * tcp: add hooks to implement tcp md5 signatures or similar (see contrib/addons  ↩
     for an example)

  2018-01-05: Simon Goldschmidt
  * Added sys_mbox_trypost_fromisr() and tcpip_callbackmsg_trycallback_fromisr()
    These can be used to post preallocated messages from an ISR to the tcpip  ↩
       thread
    (e.g. when using FreeRTOS)

  2018-01-02: Dirk Ziegelmeier
  * task #14780: Add debug helper asserts to ensure threading/locking requirements ↩
```

```
      are met

  2017-11-21: Simon Goldschmidt
  * task #14600: tcp_alloc(): kill TF_CLOSEPEND connections before other  ←
     ESTABLISHED

  2017-11-21: Simon Goldschmidt
  * makefsdata: added option "-ssi:<filename>" to control SSI tag checking/ ←
     insertion
    through a list of filenames, not by checking the file extension at runtime

  2017-11-20: Joel Cunningham
  * netconn: add LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE to use external DNS resolver ( ←
     patch #9427)

  2017-11-14: Joel Cunningham
  * netifapi: Add thread safe ARP cache APIs (task #14724)

  2017-11-06: Axel Lin
  * TCP: kill existing connections with a LOWER priority than the one currently  ←
     being opened.
    Previous implementations also kill existing connections of the SAME priority.

  2017-09-21: Kalle Olavi Niemitalo
  * sockets: add poll() implementation (patch #9450)

  2017-09-10: Joel Cunningham
  * sockets: add readv() implementation (task #14610)

  2017-08-04: Simon Goldschmidt
  * Clean up DHCP a bit: no need keep msg_out and msg_in as members in struct
    dhcp - they are used in a call stack only (p_out and options_out_len as well)

  2017-08-04: Simon Goldschmidt
  * pbuf: split pbuf_header(s16_t) into pbuf_add_header(size_t) and
    pbuf_remove_header(size_t)

  2017-07-20: Douglas
  * sys: deprecate sys_arch_sem_wait and sys_arch_mbox_fetch returning the
    time waited rather they are now defined to return != SYS_ARCH_TIMEOUT
    on success.

  2017-07-03: Jakub Schmidtke
  * tcp: added support for sending TCP SACKs

  2017-06-20: Joel Cunningham
  * netconn/netdb: added core locking support to netconn_gethostbyname (task  ←
     #14523)

  2017-04-25: Simon Goldschmidt
  * dhcp: added two hooks for adding and parsing user defined DHCP options

  2017-04-25: Joel Cunningham
  * sockets: added recvmsg for UDP (together with CMSG and IP_PKTINFO) (task  ←
     #14247)

  2017-04-20: Joel Cunningham
```

```
* tcp: added Appropriate Byte Counting support (task #14128)

2017-04-11: Simon Goldschmidt
* netconn/sockets: remove fatal error handling, fix asynchronous error handling,
  ensure data before RST can be received

2017-03-30: Simon Goldschmidt
* added "application layered TCP" connection API (altcp) for seamless ↩
   integration
  of TLS or proxy connections

2017-03-09: Simon Goldschmidt
* sockets: add recvmsg for TCP

2017-03-02: Joel Cunningham
* netconn/sockets: vectorize netconn_write for TCP, treating a vectored I/O ↩
   write
  atomically in regards to TCP segmentation (patch #8882)

2017-03-02: Simon Goldschmidt
* netconn: added nonblocking accept/recv to netconn API (task #14396)

2017-02-28: Simon Goldschmidt
* Added LWIP_SINGLE_NETIF for small targets with only one netif

2017-02-10: David van Moolenbroek
* Implement UDP and RAW multicast support for IPv6 (core API, not netconn/ ↩
   sockets)

2017-02-04: David van Moolenbroek
* IPv6 scopes support

2017-01-20: Joel Cunningham
* sockets: add interface name/index APIs (task #14314)

2017-01-08: David van Moolenbroek
* Extensions to RAW API (patch #9208)
  - Connected RAW PCBs
  - Add raw_sendto_if_src()
  - Support IP_HDRINCL socket option

++ Bugfixes:

2018-06-19: Simon Goldschmidt
* tcp: fix RTO timer not working if link is down

2018-06-15: Sylvain Rochet
* ppp: multiple smaller bugfixes

2018-05-17: Simon Goldschmidt
* etharp: arp table can now be bigger than 127 entries

2018-04-25: Jens Nielsen
* tftp server: correctly handle retransmissions

2018-04-18: Simon Goldschmidt
sockets: fix race conditions when closing full-duplex sockets
```

```
2018-03-09: Simon Goldschmidt
* 6lowpan: fix to work against contiki; added ZigBee encapsulation netif for ↩
   tests

2018-02-04: Simon Goldschmidt
* sockets: fix inconsistencies on close (inconsistent error codes, double FIN)

2018-01-05: Dirk Ziegelmeier
* Fix bug #52748: the bug in timeouts.c by reimplementing timer logic to use
  absolute instead of relative timeout values

2017-12-31: Dirk Ziegelmeier
* Fix bug #52704: DHCP and bad OFFER: Stop timeout only if offer is accepted

2017-11-08: Joel Cunningham
* netif: ensure link and admin states are up in issue reports (bug #52353)

2017-09-12: David Lockyer
* select: allocate select_cb from memp for LWIP_MPU_COMPATIBLE = 1 (bug #51990)

2017-09-11: Simon Goldschmidt
* tcp_in.c: fix bug #51937 (leaking tcp_pcbs on passive close with unacked data)

2017-08-11: Joel Cunningham
* lwip_itoa: fix converting the number 0 (previously converted to '\0') (bug ↩
   #51729)

2017-08-08: Dirk Ziegelmeier
* ip4_route_src: parameter order is reversed: ip4_route_src(dest, src) -> ↩
   ip4_route_src(src, dest)
  to make parameter order consistent with other ip*_route*() functions
  Same also applies to LWIP_HOOK_IP4_ROUTE_SRC() parameter order.

2017-08-04: Joel Cunningham
* tcp: re-work persist timer to fully close window (details in bug #50837)

2017-07-26: Simon Goldschmidt
* snmp_msg.c: fix bug #51578 (SNMP failed to decode some values on non 32bit ↩
   platforms)

2017-07-20: Simon Goldschmidt
* compatibility headers: moved from 'src/include/posix' to 'src/include/compat/ ↩
   posix',
  'src/include/compat/stdc' etc.

2017-05-09: Joel Cunningham
* tcp: add zero-window probe timeout (bug #50837)

2017-04-11: Simon Goldschmidt
* sockets.c: task #14420 (Remove sys_sem_signal from inside SYS_ARCH_PROTECT
  crit section) done for LWIP_TCPIP_CORE_LOCKING==1

2017-02-24: Simon Goldschmidt
* sockets.c: fixed close race conditions in lwip_select (for ↩
   LWIP_NETCONN_FULLDUPLEX)
  2017-02-24: Simon Goldschmidt
  * sockets.c: fixed that select ignored invalid/not open sockets in the fd_sets ( ↩
      bug #50392)

  2017-01-11: David van Moolenbroek
  * Lots of IPv6 related fixes and improvements

(STABLE-2.0.3)

  ++ Bugfixes:

  2017-09-11: Simon Goldschmidt
  * tcp_in.c: fix bug #51937 (leaking tcp_pcbs on passive close with unacked data)

  2017-08-02: Abroz Bizjak/Simon Goldschmidt
  * multiple fixes in IPv4 reassembly (leading to corrupted datagrams received)

  2017-03-30: Simon Goldschmidt
  * dhcp.c: return ERR_VAL instead of asserting on offset-out-of-pbuf

  2017-03-23: Dirk Ziegelmeier
  * dhcp.h: fix bug #50618 (dhcp_remove_struct() macro does not work)

(STABLE-2.0.2)

  ++ New features:

  2017-02-10: Dirk Ziegelmeier
  * Implement task #14367: Hooks need a better place to be defined:
    We now have a #define for a header file name that is #included in every .c
    file that provides hooks.

  2017-02-10: Simon Goldschmidt
  * tcp_close does not fail on memory error (instead, FIN is sent from tcp_tmr)

  ++ Bugfixes:

  2017-03-08
  * tcp: do not keep sending SYNs when getting ACKs

  2017-03-08: Joel Cunningham
  * tcp: Initialize ssthresh to TCP_SND_BUF (bug #50476)

  2017-03-01: Simon Goldschmidt
  * httpd: LWIP_HTTPD_POST_MANUAL_WND: fixed double-free when  ↩
      httpd_post_data_recved
    is called nested from httpd_post_receive_data() (bug #50424)

  2017-02-28: David van Moolenbroek/Simon Goldschmidt
  * tcp: fixed bug #50418: LWIP_EVENT_API: fix invalid calbacks for SYN_RCVD pcb

  2017-02-17: Simon Goldschmidt
  * dns: Improved DNS_LOCAL_HOSTLIST interface (bug #50325)

  2017-02-16: Simon Goldschmidt
  * LWIP_NETCONN_FULLDUPLEX: fixed shutdown during write (bug #50274)
  2017-02-13: Simon Goldschmidt/Dirk Ziegelmeier
  * For tiny targtes, LWIP_RAND is optional (fix compile time checks)

  2017-02-10: Simon Goldschmidt
  * tcp: Fixed bug #47485 (tcp_close() should not fail on memory error) by ↩
     retrying
    to send FIN from tcp_fasttmr

  2017-02-09: Simon Goldschmidt
  * sockets: Fixed bug #44032 (LWIP_NETCONN_FULLDUPLEX: select might work on
    invalid/reused socket) by not allowing to reallocate a socket that has
    "select_waiting != 0"

  2017-02-09: Simon Goldschmidt
  * httpd: Fixed bug #50059 (httpd LWIP_HTTPD_SUPPORT_11_KEEPALIVE vs.
    LWIP_HTTPD_KILL_OLD_ON_CONNECTIONS_EXCEEDED)

  2017-02-08: Dirk Ziegelmeier
  * Rename "IPv6 mapped IPv4 addresses" to their correct name from RFC4191:
    "IPv4-mapped IPv6 address"

  2017-02-08: Luc Revardel
  * mld6.c: Fix bug #50220 (mld6_leavegroup does not send ICMP6_TYPE_MLD, even
    if last reporter)

  2017-02-08: David van Moolenbroek
  * ip6.c: Patch #9250: fix source substitution in ip6_output_if()

  2017-02-08: Simon Goldschmidt
  * tcp_out.c: Fixed bug #50090 (last_unsent->oversize_left can become wrong value
    in tcp_write error path)

  2017-02-02: Dirk Ziegelmeier
  * Fix bug #50206: UDP Netconn bind to IP6_ADDR_ANY fails

  2017-01-18: Dirk Ziegelmeier
  * Fix zero-copy RX, see bug bug #50064. PBUF_REFs were not supported as ARP  ↩
     requests.

  2017-01-15: Axel Lin, Dirk Ziegelmeier
  * minor bug fixes in mqtt

  2017-01-11: Knut Andre Tidemann
  * sockets/netconn: fix broken default ICMPv6 handling of checksums

(STABLE-2.0.1)

  ++ New features:

  2016-12-31: Simon Goldschmidt
  * tcp.h/.c: added function tcp_listen_with_backlog_and_err() to get the error
    reason when listening fails (bug #49861)

  2016-12-20: Erik Andersen
  * Add MQTT client

  2016-12-14: Jan Breuer:
```

```
* opt.h, ndc.h/.c: add support for RDNSS option (as per RFC 6106)

2016-12-14: David van Moolenbroek
* opt.h, nd6.c: Added LWIP_HOOK_ND6_GET_GW()

2016-12-09: Dirk Ziegelmeier
* ip6_frag.c: Implemented support for LWIP_NETIF_TX_SINGLE_PBUF

2016-12-09: Simon Goldschmidt
* dns.c: added one-shot multicast DNS queries

2016-11-24: Ambroz Bizjak, David van Moolenbroek
* tcp_out.c: Optimize passing contiguous nocopy buffers to tcp_write (bug  ↩
   #46290)

2016-11-16: Dirk Ziegelmeier
* sockets.c: added support for IPv6 mapped IPv4 addresses

++ Bugfixes:

2016-12-16: Thomas Mueller
* api_lib.c: fixed race condition in return value of netconn_gethostbyname()
  (and thus also lwip_gethostbyname/_r() and lwip_getaddrinfo())

2016-12-15: David van Moolenbroek
* opt.h, tcp: added LWIP_HOOK_TCP_ISN() to implement less predictable initial
  sequence numbers (see contrib/addons/tcp_isn for an example implementation)

2016-12-05: Dirk Ziegelmeier
* fixed compiling with IPv4 disabled (IPv6 only case)

2016-11-28: Simon Goldschmidt
* api_lib.c: fixed bug #49725 (send-timeout: netconn_write() can return
  ERR_OK without all bytes being written)

2016-11-28: Ambroz Bizjak
* tcpi_in.c: fixed bug #49717 (window size in received SYN and SYN-ACK
  assumed scaled)

2016-11-25: Simon Goldschmidt
* dhcp.c: fixed bug #49676 (Possible endless loop when parsing dhcp options)

2016-11-23: Dirk Ziegelmeier
* udp.c: fixed bug #49662: multicast traffic is now only received on a UDP PCB
 (and therefore on a UDP socket/netconn) when the PCB is bound to IP_ADDR_ANY

2016-11-16: Dirk Ziegelmeier
* *: Fixed dual-stack behaviour, IPv6 mapped IPv4 support in socket API

2016-11-14: Joel Cunningham
* tcp_out.c: fixed bug #49533 (start persist timer when unsent seg can't fit
  in window)

2016-11-16: Roberto Barbieri Carrera
* autoip.c: fixed bug #49610 (sometimes AutoIP fails to reuse the same address)

2016-11-11: Dirk Ziegelmeier
```

```
  * sockets.c: fixed bug #49578 (dropping multicast membership does not work
    with LWIP_SOCKET_OFFSET)

(STABLE-2.0.0)

  ++ New features:

  2016-07-27: Simon Goldschmidt
  * opt.h, timeouts.h/.c: added LWIP_TIMERS_CUSTOM to override the default
    implementation of timeouts

  2016-07-xx: Dirk Ziegelmeier
  * Large overhaul of doxygen documentation

  2016-04-05: Simon Goldschmidt
  * timers.h/.c: prepare for overriding current timeout implementation: all
    stack-internal caclic timers are avaliable in the lwip_cyclic_timers array

  2016-03-23: Simon Goldschmidt
  * tcp: call accept-callback with ERR_MEM when allocating a pcb fails on
    passive open to inform the application about this error
    ATTENTION: applications have to handle NULL pcb in accept callback!

  2016-02-22: Ivan Delamer
  * Initial 6LoWPAN support

  2016-02-XX to 2016-03-XX: Dirk Ziegelmeier
  * Cleanup TCPIP thread sync methods in a way that it is possibe to use them
    in arbitrary code that needs things to be done in TCPIP thread. Used to
    decouple netconn, netif, ppp and 6LoWPAN from LWIP core.

  2016-02-XX: Dirk Ziegelmeier
  * Implement dual-stack support in RAW, UDP and TCP. Add new IP address
    type IPADDR_ANY_TYPE for this. Netconn/Socket API: Dual-stack is
    automatically supported when an IPv6 netconn/socket is created.

  2015-12-26: Martin Hentschel and Dirk Ziegelmeier
  * Rewrite SNMP agent. SNMPv2c + MIB compiler.

  2015-11-12: Dirk Ziegelmeier
  * Decouple SNMP stack from lwIP core and move stack to apps/ directory.
    Breaking change: Users have to call snmp_init() now!

  2015-11-12: Dirk Ziegelmeier
  * Implement possibility to declare private memory pools. This is useful to
    decouple some apps from the core (SNMP stack) or make contrib app usage
    simpler (httpserver_raw)

  2015-10-09: Simon Goldschmidt
  * started to move "private" header files containing implementation details to
    "lwip/priv/" include directory to seperate the API from the implementation.

  2015-10-07: Simon Goldschmidt
  * added sntp client as first "supported" application layer protocol  ↩
     implementation
    added 'apps' folder
```

```
2015-09-30: Dirk Ziegelmeier
* snmp_structs.h, mib_structs.c, mib2.c: snmp: fixed ugly inheritance
  implementation by aggregating the "base class" (struct mib_node) in all
  derived node classes to get more type-safe code

2015-09-23: Simon Goldschmidt
* netif.h/.c, nd6.c: task #13729: Convert netif addresses (IPv4 & IPv6) to
  ip_addr_t (so they can be used without conversion/temporary storage)

2015-09-08: Dirk Ziegelmeier
* snmp: Separate mib2 counter/table callbacks from snmp agent. This both cleans
  up the code and should allow integration of a 3rd party agent/mib2. Simple
  counters are kept in MIB2_STATS, tree/table change function prototypes moved ↩
      to
  snmp_mib2.h.

2015-09-03: Simon Goldschmidt
* opt.h, dns.h/.c: DNS/IPv6: added support for AAAA records

2015-09-01: Simon Goldschmidt
* task #12178: hardware checksum capabilities can be configured per netif
 (use NETIF_SET_CHECKSUM_CTRL() in your netif's init function)

2015-08-30: Simon Goldschmidt
* PBUF_REF with "custom" pbufs is now supported for RX pbufs (see pcapif in
  contrib for an example, LWIP_SUPPORT_CUSTOM_PBUF is required)

2015-08-30: Simon Goldschmidt
* support IPv4 source based routing: define LWIP_HOOK_IP4_ROUTE_SRC to point
  to a routing function

2015-08-05: Simon Goldschmidt
* many files: allow multicast socket options IP_MULTICAST_TTL, IP_MULTICAST_IF
  and IP_MULTICAST_LOOP to be used without IGMP

2015-04-24: Simon Goldschmidt
* dhcp.h/c, autoip.h/.c: added functions dhcp/autoip_supplied_address() to
  check for the source of address assignment (replacement for NETIF_FLAG_DHCP)

2015-04-10: Simon Goldschmidt
* many files: task #13480: added LWIP_IPV4 define - IPv4 can be disabled,
  leaving an IPv6-only stack

2015-04-09: Simon Goldschmidt
* nearly all files: task #12722 (improve IPv4/v6 address handling): renamed
  ip_addr_t to ip4_addr_t, renamed ipX_addr_t to ip_addr_t and added IP
  version; ip_addr_t is used for all generic IP addresses for the API,
  ip(4/6)_addr_t are only used internally or when initializing netifs or when
  calling version-related functions

2015-03-24: Simon Goldschmidt
* opt.h, ip4_addr.h, ip4.c, ip6.c: loopif is not required for loopback traffic
  any more but passed through any netif (ENABLE_LOOPBACK has to be enabled)

2015-03-23: Simon Goldschmidt
* opt.h, etharp.c: with ETHARP_TABLE_MATCH_NETIF== 1, duplicate (Auto)-IP
  addresses on multiple netifs should now be working correctly (if correctly
  addressed by routing, that is)

2015-03-23: Simon Goldschmidt
* etharp.c: Stable etharp entries that are about to expire are now refreshed
  using unicast to prevent unnecessary broadcast. Only if no answer is received
  after 15 seconds, broadcast is used.

2015-03-06: Philip Gladstone
* netif.h/.c: patch #8359 (Provide utility function to add an IPv6 address to
  an interface)

2015-03-05: Simon Goldschmidt
* netif.c, ip4.c, dhcp.c, autoip.c: fixed bug #37068 (netif up/down handling
  is unclear): correclty separated administrative status of a netif (up/down)
  from 'valid address' status
  ATTENTION: netif_set_up() now always has to be called, even when dhcp/autoip
  is used!

2015-02-26: patch by TabascoEye
* netif.c, udp.h/.c: fixed bug #40753 (re-bind UDP pcbs on change of IP address)

2015-02-22: chrysn, Simon Goldschmidt
* *.*: Changed nearly all functions taking 'ip(X)_addr_t' pointer to take
  const pointers (changed user callbacks: raw_recv_fn, udp_recv_fn; changed
  port callbacks: netif_output_fn, netif_igmp_mac_filter_fn)

2015-02-19: Ivan Delamer
* netif.h, dhcp.c: Removed unused netif flag for DHCP. The preferred way to  ↩
    evaluate
  if DHCP is active is through netif->dhcp field.

2015-02-19: Ivan Delamer
* netif.h, slipif.c, ppp.c: Removed unused netif flag for point to point  ↩
    connections

2015-02-18: Simon Goldschmidt
* api_lib.c: fixed bug #37958 "netconn API doesn't handle correctly
  connections half-closed by peer"

2015-02-18: Simon Goldschmidt
* tcp.c: tcp_alloc() prefers killing CLOSING/LAST_ACK over active connections
  (see bug #39565)

2015-02-16: Claudius Zingerli, Sergio Caprile
* opt.h, dhcp.h/.c: patch #8361 "Add support for NTP option in DHCP"

2015-02-14: Simon Goldschmidt
* opt.h, snmp*: added support for write-access community and dedicated
  community for sending traps

2015-02-13: Simon Goldschmidt
* opt.h, memp.c: added hook LWIP_HOOK_MEMP_AVAILABLE() to get informed when
  a memp pool was empty and an item is now available

2015-02-13: Simon Goldschmidt
* opt.h, pbuf.h/.c, etharp.c: Added the option PBUF_LINK_ENCAPSULATION_HLEN to
  allocate additional header space for TX on netifs requiring additional headers
```

```
2015-02-12: chrysn
* timers.h/.c: introduce sys_timeouts_sleeptime (returns the time left before
  the next timeout is due, for NO_SYS==1)

2015-02-11: Nick van Ijzendoorn
* opt.h, sockets.h/c: patch #7702 "Include ability to increase the socket number
  with defined offset"

2015-02-11: Frederick Baksik
* opt.h, def.h, others: patch #8423 "arch/perf.h" should be made an optional ↩
   item

2015-02-11: Simon Goldschmidt
* api_msg.c, opt.h: started to implement fullduplex sockets/netconns
  (note that this is highly unstable yet!)

2015-01-17: Simon Goldschmidt
* api: allow enabling socket API without (public) netconn API - netconn API is
  still used by sockets, but keeping it private (static) should allow better
  compiler optimizations

2015-01-16: Simon Goldschmidt
* tcp_in.c: fixed bug #20506 "Initial congestion window is very small" again
  by implementing the calculation formula from RFC3390

2014-12-10: Simon Goldschmidt
* api: added option LWIP_NETCONN_SEM_PER_THREAD to use a semaphore per thread
  instead of using one per netconn and per select call

2014-12-08: Simon Goldschmidt
* ip6.h: fixed bug #43778: IPv6 header version not set on 16-bit platform
  (macro IP6H_VTCFL_SET())

2014-12-08: Simon Goldschmidt
* icmp.c, ip4.c, pbuf.c, udp.c, pbuf.h: task #11472 Support PBUF_REF for RX
  (IPv6 and IPv4/v6 reassembly might not work yet)

2014-11-06: Simon Goldschmidt
* sockets.c/.h, init.c: lwip_socket_init() is not needed any more
  -> compatibility define

2014-09-16: Simon Goldschmidt
* dns.c, opt.h: reduced ram usage by parsing DNS responses in place

2014-09-16: Simon Goldschmidt
* pbuf.h/.c: added pbuf_take_at() and pbuf_put_at()

2014-09-15: Simon Goldschmidt
* dns.c: added source port randomization to make the DNS client more robust
  (see bug #43144)

2013-09-02: Simon Goldschmidt
* arch.h and many other files: added optional macros PACK_STRUCT_FLD_8() and
  PACK_STRUCT_FLD_S() to prevent gcc 4 from warning about struct members that
  do not need packing
```

```
2013-08-19: Simon Goldschmidt
* netif.h: bug #42998: made NETIF_MAX_HWADDR_LEN overridable for some special
  networks

2013-03-17: Simon Goldschmidt (patch by Ghobad Emadi)
* opt.h, etharp.c: Added LWIP_HOOK_ETHARP_GET_GW to implement IPv4 routing with
  multiple gateways

2013-04-20: Fatih Asici
* opt.h, etharp.h/.c: patch #7993: Added support for transmitting packets
  with VLAN headers via hook function LWIP_HOOK_VLAN_SET and to check them
  via hook function LWIP_HOOK_VLAN_CHECK

2014-02-20: Simon Goldschmidt (based on patch by Artem Pisarenko)
* patch #7885: modification of api modules to support FreeRTOS-MPU
  (don’t pass stack-pointers to other threads)

2014-02-05: Simon Goldschmidt (patch by "xtian" and "alex_ab")
* patch #6537/#7858: TCP window scaling support

2014-01-17: Jiri Engelthaler
* icmp, icmp6, opt.h: patch #8027: Completed HW checksuming for IPv4 and
  IPv6 ICMP’s

2012-08-22: Sylvain Rochet
* New PPP stack for lwIP, developed in ppp-new branch.
  Based from pppd 2.4.5, released 2009-11-17, with huge changes to match
  code size and memory requirements for embedded devices, including:
  - Gluing together the previous low-level PPP code in lwIP to pppd 2.4.5, which
    is more or less what pppd sys-* files are, so that we get something working
    using the unix port.
  - Merged some patchs from lwIP Git repository which add interesting features
    or fix bugs.
  - Merged some patchs from Debian pppd package which add interesting features
    or fix bugs.
  - Ported PPP timeout handling to the lwIP timers system
  - Disabled all the PPP code using filesystem access, replaced in necessary ↩
     cases
    to configuration variables.
  - Disabled all the PPP code forking processes.
  - Removed IPX support, lwIP does not support IPX.
  - Ported and improved random module from the previous PPP port.
  - Removed samba TDB (file-driven database) usage, because it needs a ↩
     filesystem.
  - MS-CHAP required a DES implementation, we added the latest PolarSSL DES
    implementation which is under a BSD-ish license.
  - Also switched to PolarSSL MD4,MD5,SHA1 implementations, which are meant to ↩
     be
    used in embedded devices with reduced memory footprint.
  - Removed PPP configuration file parsing support.
  - Added macro definition EAP_SUPPORT to make EAP support optional.
  - Added macro definition CHAP_SUPPORT to make CHAP support optional.
  - Added macro definition MSCHAP_SUPPORT to make MSCHAP support optional.
  - Added macro definition PAP_SUPPORT to make PAP support optional.
  - Cleared all Linux syscall calls.
  - Disabled demand support using a macro, so that it can be ported later.
  - Disabled ECP support using a macro, so that it can be ported later.
```

```
    - Disabled CCP support using a macro, so that it can be ported later.
    - Disabled CBCP support using a macro, so that it can be ported later.
    - Disabled LQR support using a macro, so that it can be ported later.
    - Print packet debug feature optional, through PRINTPKT_SUPPORT
    - Removed POSIX signal usage.
    - Fully ported PPPoS code from the previous port.
    - Fully ported PPPoE code from the previous port.
    - Fully ported VJ compression protocol code from the previous port.
    - Removed all malloc()/free() use from PPP, replaced by stack usage or PBUF.
    - Disabled PPP server support using a macro, so that it can be ported later.
    - Switched all PPP debug to lwIP debug system.
    - Created PPP Control Block (PPP PCB), removed PPP unit integer everywhere,
      removed all global variables everywhere, did everything necessary for
      the PPP stack to support more than one PPP session (pppd only support
      one session per process).
    - Removed the statically allocated output buffer, now using PBUF.
    - Improved structure size of all PPP modules, deep analyze of code to reduce
      variables size to the bare minimum. Switched all boolean type (char type in
      most architecture) to compiler generated bitfields.
    - Added PPP IPv6 support, glued lwIP IPv6 support to PPP.
    - Now using a persistent netif interface which can then be used in lwIP
      functions requiring a netif.
    - Now initializing PPP in lwip_init() function.
    - Reworked completely the PPP state machine, so that we don't end up in
      anymore in inconsistent state, especially with PPPoE.
    - Improved the way we handle PPP reconnection after disconnect, cleaning
      everything required so that we start the PPP connection again from a
      clean state.
    - Added PPP holdoff support, allow the lwIP user to wait a little bit before
      reconnecting, prevents connection flood, especially when using PPPoL2TP.
    - Added PPPoL2TP LAC support (a.k.a. UDP tunnels), adding a VPN client
      feature to lwIP, L2TP being a widely used tunnel protocol.
    - Switched all used PPP types to lwIP types (u8t, u16t, u32t, ...)
    - Added PPP API "sequential" thread-safe API, based from NETIFAPI.

  2011-07-21: Simon Goldschmidt
  * sockets.c, opt.h: (bug #30185): added LWIP_FIONREAD_LINUXMODE that makes
    ioctl/FIONREAD return the size of the next pending datagram.

  2011-05-25: Simon Goldschmidt
  * again nearly the whole stack, renamed ip.c to ip4.c, ip_addr.c to ip4_addr.c,
    combined ipv4/ipv6 inet_chksum.c, added ip.h, ip_addr.h: Combined IPv4
    and IPv6 code where possible, added defines to access IPv4/IPv6 in non-IP
    code so that the code is more readable.

  2011-05-17: Patch by Ivan Delamer (only checked in by Simon Goldschmidt)
  * nearly the whole stack: Finally, we got decent IPv6 support, big thanks to
    Ivan! (this is work in progress: we're just post release anyway :-)


  ++ Bugfixes:

  2016-08-23: Simon Goldschmidt
  * etharp: removed ETHARP_TRUST_IP_MAC since it is insecure and we don't need
    it any more after implementing unicast ARP renewal towards arp entry timeout

  2016-07-20: Simon Goldschmidt
```

```
* memp.h/.c: fixed bug #48442 (memp stats don't work for MEMP_MEM_MALLOC)

2016-07-21: Simon Goldschmidt (patch by Ambroz Bizjak)
* tcp_in.c, tcp_out.c: fixed bug #48543 (TCP sent callback may prematurely
  report sent data when only part of a segment is acked) and don't include
  SYN/FIN in snd_buf counter

2016-07-19: Simon Goldschmidt
* etharp.c: fixed bug #48477 (ARP input packet might update static entry)

2016-07-11: Simon Goldschmidt
* tcp_in.c: fixed bug #48476 (TCP sent callback called wrongly due to picking
  up old pcb->acked

2016-06-30: Simon Goldschmidt (original patch by Fabian Koch)
* tcp_in.c: fixed bug #48170 (Vulnerable to TCP RST spoofing)

2016-05-20: Dirk Ziegelmeier
* sntp.h/.c: Fix return value of sntp_getserver() call to return a pointer

2016-04-05: Simon Goldschmidt (patch by Philip Gladstone)
* udp.c: patch #8358: allow more combinations of listening PCB for IPv6

2016-04-05: Simon Goldschmidt
* netconn/socket API: fixed bug# 43739 (Accept not reporting errors about
  aborted connections): netconn_accept() returns ERR_ABRT (sockets: ECONNABORTED ↩
      )
  for aborted connections, ERR_CLSD (sockets: EINVAL) if the listening netconn
  is closed, which better seems to follow the standard.

2016-03-23: Florent Matignon
* dhcp.c: fixed bug #38203: DHCP options are not recorded in all DHCP ack  ↩
   messages

2016-03-22: Simon Goldschmidt
* tcp: changed accept handling to be done internally: the application does not
  have to call tcp_accepted() any more. Instead, when delaying accept (e.g.  ↩
      sockets
  do), call tcp_backlog_delayed()/tcp_backlog_accepted() (fixes bug #46696)

2016-03-22: Simon Goldschmidt
* dns.c: ignore dns response parsing errors, only abort resolving for correct
  responses or error responses from correct server (bug #47459)

2016-03-17: Simon Goldschmidt
* api_msg.c: fixed bug #47448 (netconn/socket leak if RST is received during  ↩
   close)

2016-03-17: Joel Cunningham
* api_msg.c: don't fail closing a socket/netconn when failing to allocate the
  FIN segment; blocking the calling thread for a while is better than risking
  leaking a netconn/socket (see bug #46701)

2016-03-16: Joel Cunningham
* tcp_out.c: reset rto timer on fast retransmission

2016-03-16: Deomid Ryabkov
```

```
* tcp_out.c: fixed bug #46384 Segment size calculation bug with MSS != TCP_MSS

2016-03-05: Simon Goldschmidt
* err.h/.c, sockets.c: ERR_IF is not necessarily a fatal error

2015-11-19: fix by Kerem Hadimli
* sockets.c: fixed bug #46471: lwip_accept() leaks socket descriptors if new
  netconn was already closed because of peer behavior

2015-11-12: fix by Valery Ushakov
* tcp_in.c: fixed bug #46365 tcp_accept_null() should call tcp_abort()

2015-10-02: Dirk Ziegelmeier/Simon Goldschmidt
* snmp: cleaned up snmp structs API (fixed race conditions from bug #46089,
  reduce ram/rom usage of tables): incompatible change for private MIBs

2015-09-30: Simon Goldschmidt
* ip4_addr.c: fixed bug #46072: ip4addr_aton() does not check the number range
  of all address parts

2015-08-28: Simon Goldschmidt
* tcp.c, tcp_in.c: fixed bug #44023: TCP ssthresh value is unclear: ssthresh
  is set to the full send window for active open, too, and is updated once
  after SYN to ensure the correct send window is used

2015-08-28: Simon Goldschmidt
* tcp: fixed bug #45559: Window scaling casts u32_t to u16_t without checks

2015-08-26: Simon Goldschmidt
* ip6_frag.h/.c: fixed bug bug #41009: IPv6 reassembly broken on 64-bit  ↩
   platforms:
  define IPV6_FRAG_COPYHEADER==1 on these platforms to copy the IPv6 header
  instead of referencing it, which gives more room for struct ip6_reass_helper

2015-08-25: Simon Goldschmidt
* sockets.c: fixed bug #45827: recvfrom: TCP window is updated with MSG_PEEK

2015-08-20: Manoj Kumar
* snmp_msg.h, msg_in.c: fixed bug #43790: Sending octet string of Length >255
  from SNMP agent

2015-08-19: Jens Nielsen
* icmp.c, ip4.c, tcp_in.c, udp.c, raw.c: fixed bug #45120: Broadcast & multiple
  interfaces handling

2015-08-19: Simon Goldschmidt (patch by "Sandra")
* dns.c: fixed bug #45004: dns response without answer might be discarded

2015-08-18: Chrysn
* timers.c: patch #8704 fix sys_timeouts_sleeptime function

2015-07-01: Erik Ekman
* puf.c: fixed bug #45454 (pbuf_take_at() skips write and returns OK if offset
  is at start of pbuf in chain)

2015-05-19: Simon Goldschmidt
* dhcp.h/.c: fixed bugs #45140 and #45141 (dhcp was not stopped correctly after
  fixing bug #38204)

2015-03-21: Simon Goldschmidt (patch by Homyak)
* tcp_in.c: fixed bug #44766 (LWIP_WND_SCALE: tcphdr->wnd was not scaled in
  two places)

2015-03-21: Simon Goldschmidt
* tcp_impl.h, tcp.c, tcp_in.c: fixed bug #41318 (Bad memory ref in tcp_input()
  after tcp_close())

2015-03-21: Simon Goldschmidt
* tcp_in.c: fixed bug #38468 (tcp_sent() not called on half-open connection for
  data ACKed with the same ack as FIN)

2015-03-21: Simon Goldschmidt (patch by Christoffer Lind)
* dhcp.h/.c: fixed bug #38204 (DHCP lease time not handled correctly)

2015-03-20: Simon Goldschmidt
* dhcp.c: fixed bug #38714 (Missing option and client address in DHCPRELEASE  ↩
   message)

2015-03-19: Simon Goldschmidt
* api.h, tcpip.h, api_lib.c, api_msg.c: fixed race conditions in assigning
  netconn->last_err (fixed bugs #38121 and #37676)

2015-03-09: Simon Goldschmidt
* ip4.c: fixed the IPv4 part of bug #43904 (ip_route() must detect linkup status ↩
   )

2015-03-04: Simon Goldschmidt
* nd6.c: fixed bug #43784 (a host should send at least one Router Solicitation)

2015-03-04: Valery Ushakov
* ip6.c: fixed bug #41094 (Byte-order bug in IPv6 fragmentation header test)

2015-03-04: Zach Smith
* nd6.c: fixed bug #38153 (nd6_input() byte order issues)

2015-02-26: Simon Goldschmidt
* netif.c, tcp.h/.c: fixed bug #44378 (TCP connections are not aborted on netif
  remove)

2015-02-25: Simon Goldschmidt
* ip4.c, etharp.c: fixed bug #40177 (System hangs when dealing with corrupted
  packets), implemented task #12357 (Ensure that malicious packets don't
  assert-fail): improved some pbuf_header calls to not assert-fail.

2015-02-25: patch by Joel Cunningham
* udp.h/.c, sockets.c: fixed bug #43028 (IP_MULTICAST_TTL affects unicast
  datagrams)

2015-02-25: patch by Greg Renda
* ip4_frag.c: fixed bug #38210 (ip reassembly while remove oldest datagram)

2015-02-25: Simon Goldschmidt
* sockets.c: fixed bug #38165 (socket with mulicast): ensure igmp membership
  are dropped when socket (not netconn!) is closed.
```

```
2015-02-25: Simon Goldschmidt
* ip4.h/.c, udp.c: fixed bug #38061 (wrong multicast routing in IPv4) by
  adding an optional default netif for multicast routing

2015-02-25: Simon Goldschmidt
* netconn API: fixed that netconn_connect still used message passing for
  LWIP_TCPIP_CORE_LOCKING==1

2015-02-22: patch by Jens Nielsen
* icmp.c: fixed bug #38803 (Source address in broadcast ping reply)

2015-02-22: Simon Goldschmidt
* udp.h, sockets.c: added proper accessor functions for pcb->multicast_ip
  (previously used by get/setsockopt only)

2015-02-18: Simon Goldschmidt
* sockets.c: Fixed select not reporting received FIN as 'readable' in certain
  rare cases (bug #43779: select(), close(), and TCP retransmission error)

2015-02-17: Simon Goldschmidt
* err.h, sockets.c, api_msg.c: fixed bug #38853 "connect() use a wrong errno":
  return ERR_ALREADY/EALRADY during connect, ERR_ISCONN/EISCONN when already
  connected

2015-02-17: Simon Goldschmidt
* tcp_impl.h, tcp_out.c, tcp.c, api_msg.c: fixed bug #37614 "Errors from
  ipX_output are not processed". Now tcp_output(_segment) checks for the return
  value of ipX_output and does not try to send more on error. A netif driver
  can call tcp_txnow() (from tcpip_thread!) to try to send again if TX buffers
  are available again.

2015-02-14: patches by Freddie Chopin
* snmp*: made community writable, fixed some const pointers

2015-02-13: Simon Goldschmidt
* msg_in.c: fixed bug #22070 "MIB_OBJECT_WRITE_ONLY not implemented in SNMP"

2015-02-12: Simon Goldschmidt
* ip.h, ip4.c, ip6.c: fixed bug #36403 "ip4_input() and ip6_input() always pass
  inp to higher layers": now the accepting netif is passed up, but the input
  netif is available through ip_current_input_netif() if required.

2015-02-11: patch by hichard
* tcpip.c: fixed bug #43094 "The function tcpip_input() forget to handle IPv6"

2015-02-10: Simon Goldschmidt
* netconn API: fixed that netconn_close/netconn_delete still used message  ↩
   passing
  for LWIP_TCPIP_CORE_LOCKING==1

2015-02-10: Simon Goldschmidt
* netconn/socket api: fixed bug #44225 "closing TCP socket should time out
  eventually", implemented task #6930 "Implement SO_LINGER": closing TCP sockets
  times out after 20 seconds or after the configured SND_TIMEOUT or depending
  on the linger settings.
```

```
2015-01-27: Simon Goldschmidt
* api_msg.c: fixed that SHUT_RD followed by SHUT_WR was different to SHUT_RDWR,
  fixed return value of lwip_netconn_do_close on unconnected netconns

2015-01-17: Simon Goldschmidt
* sockets.c: fixed bug #43361 select() crashes with stale FDs

2015-01-17: Simon Goldschmidt
* sockets.c/.h, memp_std.h: fixed bug #40788 "lwip_setsockopt_internal() crashes ↩
   "
  by rewriting set/getsockopt functions to combine checks with the actual code
  and add more NULL checks; this also fixes that CORE_LOCKING used message
  passing for set/getsockopt.

2014-12-19: Simon Goldschmidt
* opt.h, dhcp.h/.c: prevent dhcp from starting when netif link is down (only
  when LWIP_DHCP_CHECK_LINK_UP==1, which is disabled by default for
  compatibility reasons)

2014-12-17: Simon Goldschmidt
* tcp_out.c: fixed bug #43840 Checksum error for TCP_CHECKSUM_ON_COPY==1 for
  no-copy data with odd length

2014-12-10: Simon Goldschmidt
* sockets.c, tcp.c, others: fixed bug #43797 set/getsockopt: SO_SNDTIMEO/ ↩
   SO_RCVTIMEO
  take int as option but should take timeval (LWIP_SO_SNDRCVTIMEO_STANDARD==0  ↩
     can
  be used to revert to the old 'winsock' style behaviour)
  Fixed implementation of SO_ACCEPTCONN to just look at the pcb state

2014-12-09: Simon Goldschmidt
* ip4.c: fixed bug #43596 IGMP queries from 0.0.0.0 are discarded

2014-10-21: Simon Goldschmidt (patch by Joel Cunningham and Albert Huitsing)
* sockts.c: fixed bugs #41495 Possible threading issue in select() and #43278
  event_callback() handle context switch when calling sys_sem_signal()

2014-10-21: Simon Goldschmidt
* api_msg.c: fixed bug #38219 Assert on TCP netconn_write with sndtimeout set

2014-09-16: Kevin Cernekee
* dns.c: patch #8480 Fix handling of dns_seqno wraparound

2014-09-16: Simon Goldschmidt
* tcp_out.c: fixed bug #43192 tcp_enqueue_flags() should not check  ↩
   TCP_SND_QUEUELEN
  when sending FIN

2014-09-03: Simon Goldschmidt
* msg_in.c: fixed bug #39355 SNMP Memory Leak in case of error

2014-09-02: Simon Goldschmidt
* err.h/.c, sockets.c, api_msg.c: fixed bug #43110 call getpeername() before
  listen() will cause a error

2014-09-02: Simon Goldschmidt
```

```
* sockets.c: fixed bug #42117 lwip_fcntl does not set errno

2014-09-02: Simon Goldschmidt
* tcp.c: fixed bug #42299 tcp_abort() leaves freed pcb on tcp_bound_pcbs list

2014-08-20: Simon Goldschmidt
* dns.c: fixed bug #42987 lwIP is vulnerable to DNS cache poisoning due to
  non-randomized TXIDs

2014-06-03: Simon Goldschmidt
* tcp_impl.h, tcp_in.c: fixed bug #37969 SYN packet dropped as short packet in
  tcp_input function

2014-05-20: Simon Goldschmidt
* tcp_out.c: fixed bug #37184 tcp_write problem for pcbs in the SYN_SENT state

2014-05-19: Simon Goldschmidt
* *.h: Fixed bug #35874 reserved identifier violation (removed leading ↩
   underscores
  from header include guards)

2014-04-08: Simon Goldschmidt
* tcp.c: Fixed bug #36167 tcp server crash when client closes (maximum window)

2014-04-06: Simon Goldschmidt
* tcp_in.c: Fixed bug #36210 lwIP does not elicit an empty ACK when received
  unacceptable ACK

2014-04-06: Simon Goldschmidt
* dhcp.c, ip4.c/.h, ip6.c/.h, udp.c/.h, ip.h: Fixed bug #41787 DHCP Discovery
  is invalid when an IP is set to thet netif.

2014-03-14: Simon Goldschmidt
* tcp_out.c: Fixed bug #36153 TCP Cheksum error if LWIP_CHECKSUM_ON_COPY=1

2014-03-11: Simon Goldschmidt (patch by Mason)
* opt.h, sockets.c: fixed bug #35928 BSD sockets functions must set errno for
  POSIX-compliance

2014-02-27: Simon Goldschmidt
* dhcp.c: fixed bug #40303 DHCP xid renewed when sending a DHCPREQUEST

2014-02-27: Simon Goldschmidt
* raw.c: fixed bug #41680 raw socket can not receive IPv6 packet when
  IP_SOF_BROADCAST_RECV==1

2014-02-27: Simon Goldschmidt
* api_msg.c, sockets.c: fixed bug #38404 getpeeraddr returns success on
  unconnected/listening TCP sockets

2014-02-27: Simon Goldschmidt
* sockets.c: fixed bug #41729 Some socket functions return Exyz instead of -1

2014-02-25: Simon Goldschmidt
* ip4.c: fixed bug #39514 ip_route() may return an IPv6-only interface

2014-02-25: Simon Goldschmidt, patch by Fatih Asici
```

```
* pbuf.c: fixed bug #39356 Wrong increment in pbuf_memfind()

2014-02-25: Simon Goldschmidt
* netif.c/.h, udp.c: fixed bug #39225 udp.c uses netif_matches_ip6_addr()  ↩
    incorrectly;
  renamed function netif_matches_ip6_addr() to netif_get_ip6_addr_match()

2014-02-25: Simon Goldschmidt
* igmp.c: fixed bug #39145 IGMP membership report for 224.0.0.1

2014-02-22: Simon Goldschmidt (patch by Amir Shalem)
* etharp.c, opt.h: fixed bug #34681 Limit ARP queue length by ARP_QUEUE_LEN (=3)

2014-02-22: Simon Goldschmidt (patch by Amir Shalem)
* etharp.h/.c: fixed bug #34682 Limit ARP request flood for unresolved entry

2014-02-20: Simon Goldschmidt
* tcp_out.c: fixed bug #39683 Assertion "seg->tcphdr not aligned" failed with
  MEM_ALIGNMENT = 8

2014-02-20: Simon Goldschmidt
* sockets.c: fixed bug #39882 No function shall set errno to 0

2014-02-20: Simon Goldschmidt
* mib_structs.c: fixed bug #40050 SNMP problem with MIB arrays > 255

2014-02-20: Simon Goldschmidt
* api.h, sockets.c: fixed bug #41499 netconn::recv_avail can overflow

2014-01-08: Stathis Voukelatos
* memp_std.h: patch #7928 Fixed size calculation in MALLOC memory pool
  creation macro

2014-01-18: Brian Fahs
* tcp_out.c: patch #8237: tcp_rexmit_rto fails to update pcb->unsent_oversize
  when necessary

2014-01-17: Grant Erickson, Jay Logue, Simon Goldschmidt
* ipv6.c, netif.c: patch #7913 Enable Support for IPv6 Loopback

2014-01-16: Stathis Voukelatos
* netif.c: patch #7902 Fixed netif_poll() operation when LWIP_LOOPBACK_MAX_PBUFS ↩
     > 0

2014-01-14: "Freddie Chopin"
* snmp.h, mib2.c: fixed constness and spelling of sysdescr

2014-01-14: Simon Goldschmidt (patch by Thomas Faber)
* tcpip.c: patch #8241: Fix implicit declaration of ip_input with
  LWIP_TCPIP_CORE_LOCKING_INPUT disabled

2014-01-14: chrysn
* timers.c: patch #8244 make timeouts usable reliably from outside of the
  timeout routine

2014-01-10: Simon Goldschmidt
* ip_frag.c, ip6_frag.c: fixed bug #41041 Potential use-after-free in IPv6  ↩
```

```
    reassembly

  2014-01-10: Simon Goldschmidt
  * memp.c: fixed bug #41188 Alignment error in memp_init() when  ↩
    MEMP_SEPARATE_POOLS==1

  2014-01-10: Simon Goldschmidt
  * tcp.c: fixed bug #39898 tcp_fasttmr() possible lock due to infinte queue  ↩
    process loop

  2013-06-29: Simon Goldschmidt
  * inet.h, sockets.h: partially fixed bug #37585: IPv6 compatibility (in socket  ↩
    structs)

  2013-06-29: Simon Goldschmidt
  * inet6.h: bug #37585/task #12600: fixed struct in6_addr.s6_addr to conform to  ↩
    spec

  2013-04-24: patch by Liam <morepork>
  * api_msg.c: patch #8008 Fix a potential null pointer dereference in assert

  2013-04-24: Simon Goldschmidt
  * igmp.c: fixed possible division by zero

  2013-04-24: Simon Goldschmidt
  * ip6.h, some ipv6 C files: fixed bug #38526 Coverity: Recursive Header  ↩
    Inclusion in ip6.h

  2013-04-24: Simon Goldschmidt (patch by Emil Ljungdahl):
  * netif.c: fixed bug #38586 netif_loop_output() "deadlocks"

  2013-01-15: Simon Goldschmidt
  * ip4.c: fixed bug #37665 ip_canforward operates on address in wrong byte order

  2013-01-15: Simon Goldschmidt
  * pbuf.h: fixed bug #38097 pbuf_free_ooseq() warning

  2013-01-14: Simon Goldschmidt
  * dns.c: fixed bug #37705 Possible memory corruption in DNS query

  2013-01-11: Simon Goldschmidt
  * raw.c: fixed bug #38066 Raw pcbs can alter packet without eating it

  2012-08-22: Simon Goldschmidt
  * memp.c: fixed bug #37166: memp_sanity check loops itself

  2012-08-13: Simon Goldschmidt
  * dhcp.c: fixed bug #36645: Calling dhcp_release before dhcp_start
    dereferences NULL

  2012-08-13: Simon Goldschmidt
  * msg_out.c: fixed bug #36840 snmp_send_trap() NULL de-reference if traps
    configured but no interfaces available

  2012-08-13: Simon Goldschmidt
  * dns.c: fixed bug #36899 DNS TTL 0 is cached for a long time
  2012-05-11: Simon Goldschmidt (patch by Marty)
  * memp.c: fixed bug #36412: memp.c does not compile when
    MEMP_OVERFLOW_CHECK > zero and MEMP_SEPARATE_POOLS == 1

  2012-05-03: Simon Goldschmidt (patch by Sylvain Rochet)
  * ppp.c: fixed bug #36283 (PPP struct used on header size computation and
    not packed)

  2012-05-03: Simon Goldschmidt (patch by David Empson)
  * ppp.c: fixed bug #36388 (PPP: checksum-only in last pbuf leads to pbuf with
    zero length)

  2012-03-25: Simon Goldschmidt
  * api_msg.c: Fixed bug #35817: do_connect() invalidly signals op_completed
    for UDP/RAW with LWIP_TCPIP_CORE_LOCKING==1

  2012-03-25: Simon Goldschmidt
  * api_msg.h, api_lib.c, api_msg.c, netifapi.c: fixed bug #35931: Name space
    pollution in api_msg.c and netifapi.c

  2011-08-24: Simon Goldschmidt
  * inet6.h: fixed bug #34124 struct in6_addr does not conform to the standard



(STABLE-1.4.1)

  ++ New features:

  2012-03-25: Simon Goldschmidt (idea by Mason)
  * posix/*: added posix-compatibility include files posix/netdb.h and posix/sys/ ↩
      socket.h
    which are a simple wrapper to the correct lwIP include files.

  2012-01-16: Simon Goldschmidt
  * opt.h, icmp.c: Added option CHECKSUM_GEN_ICMP

  2011-12-17: Simon Goldschmidt
  * ip.h: implemented API functions to access so_options of IP pcbs (UDP, TCP, RAW ↩
      )
    (fixes bug #35061)

  2011-09-27: Simon Goldschmidt
  * opt.h, tcp.c, tcp_in.c: Implemented limiting data on ooseq queue (task #9989)
    (define TCP_OOSEQ_MAX_BYTES / TCP_OOSEQ_MAX_PBUFS in lwipopts.h)

  2011-09-21: Simon Goldschmidt
  * opt.h, api.h, api_lib.c, api_msg.h/.c, sockets.c: Implemented timeout on
    send (TCP only, bug #33820)

  2011-09-21: Simon Goldschmidt
  * init.c: Converted runtime-sanity-checks into compile-time checks that can
    be disabled (since runtime checks can often not be seen on embedded targets)

  2011-09-11: Simon Goldschmidt
  * ppp.h, ppp_impl.h: splitted ppp.h to an internal and external header file
    to get a clear separation of which functions an application or port may use
    (task #11281)

 2011-09-11: Simon Goldschmidt
  * opt.h, tcp_impl.h, tcp.c, udp.h/.c: Added a config option to randomize
    initial local TCP/UDP ports (so that different port ranges are used after
    a reboot; bug #33818; this one added tcp_init/udp_init functions again)

  2011-09-03: Simon Goldschmidt
  * dhcp.c: DHCP uses LWIP_RAND() for xid's (bug #30302)

  2011-08-24: Simon Goldschmidt
  * opt.h, netif.h/.c: added netif remove callback (bug #32397)

  2011-07-26: Simon Goldschmidt
  * etharp.c: ETHARP_SUPPORT_VLAN: add support for an external VLAN filter
    function instead of only checking for one VLAN (define ETHARP_VLAN_CHECK_FN)

  2011-07-21: Simon Goldschmidt (patch by hanhui)
  * ip4.c, etharp.c, pbuf.h: bug #33634 ip_forward() have a faulty behaviour:
    Added pbuf flags to mark incoming packets as link-layer broadcast/multicast.
    Also added code to allow ip_forward() to forward non-broadcast packets to
    the input netif (set IP_FORWARD_ALLOW_TX_ON_RX_NETIF==1).

  2011-06-26: Simon Goldschmidt (patch by Cameron Gutman)
  * tcp.c, tcp_out.c: bug #33604: added some more asserts to check that
    pcb->state != LISTEN

   2011-05-14: Simon Goldschmidt (patch by Stéphane Lesage)
  * tcpip.c/.h: patch #7449 allow tcpip callback from interrupt with static
    memory message


  ++ Bugfixes:

  2012-09-26: Simon Goldschmidt
  * api_msg.c: fixed bug #37405 'err_tcp()' uses already freed 'netconn' object

  2012-09-26: patch by Henrik Persson
  * dhcp.c: patch #7843 Fix corner case with dhcp timeouts

  2012-09-26: patch by Henrik Persson
  * dhcp.c: patch #7840 Segfault in dhcp_parse_reply if no end marker in dhcp  ↩
     packet

  2012-08-22: Simon Goldschmidt
  * memp.c: fixed bug #37166: memp_sanity check loops itself

  2012-05-08: Simon Goldschmidt
  * tcp_out.c: fixed bug: #36380 unsent_oversize mismatch in 1.4.1RC1 (this was
    a debug-check issue only)

  2012-03-27: Simon Goldschmidt
  * vj.c: fixed bug #35756 header length calculation problem in ppp/vj.c

  2012-03-27: Simon Goldschmidt (patch by Mason)
  * tcp_out.c: fixed bug #35945: SYN packet should provide the recv MSS not the
    send MSS
  2012-03-22: Simon Goldschmidt
  * ip4.c: fixed bug #35927: missing refragmentaion in ip_forward

  2012-03-20: Simon Goldschmidt (patch by Mason)
  * netdb.c: fixed bug #35907: lwip_gethostbyname_r returns an invalid h_addr_list

  2012-03-12: Simon Goldschmidt (patch by Bostjan Meglic)
  * ppp.c: fixed bug #35809: PPP GetMask(): Compiler warning on big endian,
    possible bug on little endian system

  2012-02-23: Simon Goldschmidt
  * etharp.c: fixed bug #35595: Impossible to send broadcast without a gateway
    (introduced when fixing bug# 33551)

  2012-02-16: Simon Goldschmidt
  * ppp.c: fixed pbuf leak when PPP session is aborted through pppSigHUP()
    (bug #35541: PPP Memory Leak)

  2012-02-16: Simon Goldschmidt
  * etharp.c: fixed bug #35531: Impossible to send multicast without a gateway
    (introduced when fixing bug# 33551)

  2012-02-16: Simon Goldschmidt (patch by Stéphane Lesage)
  * msg_in.c, msg_out.c: fixed bug #35536 SNMP: error too big response is  ↩
     malformed

  2012-02-15: Simon Goldschmidt
  * init.c: fixed bug #35537: MEMP_NUM_* sanity checks should be disabled with
    MEMP_MEM_MALLOC==1

  2012-02-12: Simon Goldschmidt
  * tcp.h, tcp_in.c, tcp_out.c: partly fixed bug #25882: TCP hangs on
    MSS > pcb->snd_wnd (by not creating segments bigger than half the window)

  2012-02-11: Simon Goldschmidt
  * tcp.c: fixed bug #35435: No pcb state check before adding it to time-wait
    queue while closing

  2012-01-22: Simon Goldschmidt
  * tcp.c, tcp_in.c: fixed bug #35305: pcb may be freed too early on shutdown(WR)

  2012-01-21: Simon Goldschmidt
  * tcp.c: fixed bug #34636: FIN_WAIT_2 - Incorrect shutdown of TCP pcb

  2012-01-20: Simon Goldschmidt
  * dhcp.c: fixed bug #35151: DHCP asserts on incoming option lengths

 2012-01-20: Simon Goldschmidt
  * pbuf.c: fixed bug #35291: NULL pointer in pbuf_copy

  2011-11-25: Simon Goldschmidt
  * tcp.h/.c, tcp_impl.h, tcp_in.c: fixed bug #31177: tcp timers can corrupt
    tcp_active_pcbs in some cases

  2011-11-23: Simon Goldschmidt
  * sys.c: fixed bug #34884: sys_msleep() body needs to be surrounded with
```

```
  '#ifndef sys_msleep'

2011-11-22: Simon Goldschmidt
* netif.c, etharp.h/.c: fixed bug #34684: Clear the arp table cache when
  netif is brought down

2011-10-28: Simon Goldschmidt
* tcp_in.c: fixed bug #34638: Dead code in tcp_receive - pcb->dupacks

2011-10-23: Simon Goldschmidt
* mem.c: fixed bug #34429: possible memory corruption with
  LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT set to 1

2011-10-18: Simon Goldschmidt
* arch.h, netdb.c: fixed bug #34592: lwip_gethostbyname_r uses nonstandard
  error value

2011-10-18: Simon Goldschmidt
* opt.h: fixed default values of TCP_SNDLOWAT and TCP_SNDQUEUELOWAT for small
  windows (bug #34176 select after non-blocking send times out)

2011-10-18: Simon Goldschmidt
* tcp_impl.h, tcp_out.c: fixed bug #34587: TCP_BUILD_MSS_OPTION doesn't
  consider netif->mtu, causes slow network

2011-10-18: Simon Goldschmidt
* sockets.c: fixed bug #34581 missing parentheses in udplite sockets code

2011-10-18: Simon Goldschmidt
* sockets.h: fixed bug #34580 fcntl() is missing in LWIP_COMPAT_SOCKETS

2011-10-17: Simon Goldschmidt
* api_msg.c: fixed bug #34569: shutdown(SHUT_WR) crashes netconn/socket api

2011-10-13: Simon Goldschmidt
* tcp_in.c, tcp_out.c: fixed bug #34517 (persist timer is started although no
  zero window is received) by starting the persist timer when a zero window is
  received, not when we have more data queued for sending than fits into the
  window

2011-10-13: Simon Goldschmidt
* def.h, timers.c: fixed bug #34541: LWIP_U32_DIFF is unnecessarily complex

2011-10-13: Simon Goldschmidt
* sockets.c, api_lib.c: fixed bug #34540: compiler error when CORE_LOCKING is
  used and not all protocols are enabled

2011-10-12: Simon Goldschmidt
* pbuf.c: fixed bug #34534: Error in sending fragmented IP if MEM_ALIGNMENT > 4

2011-10-09: Simon Goldschmidt
* tcp_out.c: fixed bug #34426: tcp_zero_window_probe() transmits incorrect
  byte value when pcb->unacked != NULL

2011-10-09: Simon Goldschmidt
* ip4.c: fixed bug #34447 LWIP_IP_ACCEPT_UDP_PORT(dst_port) wrong
```

```
2011-09-27: Simon Goldschmidt
* tcp_in.c, tcp_out.c: Reset pcb->unsent_oversize in 2 more places...

2011-09-27: Simon Goldschmidt
* tcp_in.c: fixed bug #28288: Data after FIN in oos queue

2011-09-27: Simon Goldschmidt
* dhcp.c: fixed bug #34406 dhcp_option_hostname() can overflow the pbuf

2011-09-24: Simon Goldschmidt
* mem.h: fixed bug #34377 MEM_SIZE_F is not defined if MEM_LIBC_MALLOC==1

2011-09-23: Simon Goldschmidt
* pbuf.h, tcp.c, tcp_in.c: fixed bug #33871: rejecting TCP_EVENT_RECV() for
  the last packet including FIN can lose data

2011-09-22: Simon Goldschmidt
* tcp_impl.h: fixed bug #34355: nagle does not take snd_buf/snd_queuelen into
  account

2011-09-21: Simon Goldschmidt
* opt.h: fixed default value of TCP_SND_BUF to not violate the sanity checks
  in init.c

2011-09-20: Simon Goldschmidt
* timers.c: fixed bug #34337 (possible NULL pointer in sys_check_timeouts)

2011-09-11: Simon Goldschmidt
* tcp_out.c: use pcb->mss instead of TCP_MSS for preallocate mss-sized pbufs
  (bug #34019)

2011-09-09: Simon Goldschmidt
* udp.c: fixed bug #34072: UDP broadcast is received from wrong UDP pcb if
  udp port matches

2011-09-03: Simon Goldschmidt
* tcp_in.c: fixed bug #33952 PUSH flag in incoming packet is lost when packet
  is aggregated and sent to application

2011-09-01: Simon Goldschmidt
* opt.h: fixed bug #31809 LWIP_EVENT_API in opts.h is inconsistent compared
  to other options

2011-09-01: Simon Goldschmidt
* tcp_in.c: fixed bug #34111 RST for ACK to listening pcb has wrong seqno

2011-08-24: Simon Goldschmidt
* api_msg.c, sockets.c: fixed bug #33956 Wrong error returned when calling
  accept() on UDP connections

2011-08-24: Simon Goldschmidt
* sockets.h: fixed bug #34057 socklen_t should be a typedef

2011-08-24: Simon Goldschmidt
* pbuf.c: fixed bug #34112 Odd check in pbuf_alloced_custom (typo)

2011-08-24: Simon Goldschmidt
```

```
  * dhcp.c: fixed bug #34122 dhcp: hostname can overflow

  2011-08-24: Simon Goldschmidt
  * netif.c: fixed bug #34121 netif_add/netif_set_ipaddr fail on NULL ipaddr

  2011-08-22: Simon Goldschmidt
  * tcp_out.c: fixed bug #33962 TF_FIN not always set after FIN is sent. (This
    merely prevents nagle from not transmitting fast after closing.)

  2011-07-22: Simon Goldschmidt
  * api_lib.c, api_msg.c, sockets.c, api.h: fixed bug #31084 (socket API returns
    always EMSGSIZE on non-blocking sockets if data size > send buffers) -> now
    lwip_send() sends as much as possible for non-blocking sockets

  2011-07-22: Simon Goldschmidt
  * pbuf.c/.h, timers.c: freeing ooseq pbufs when the pbuf pool is empty  ↩
     implemented
    for NO_SYS==1: when not using sys_check_timeouts(), call PBUF_CHECK_FREE_OOSEQ ↩
       ()
    at regular intervals from main level.

  2011-07-21: Simon Goldschmidt
  * etharp.c: fixed bug #33551 (ARP entries may time out although in use) by
    sending an ARP request when an ARP entry is used in the last minute before
    it would time out.

  2011-07-04: Simon Goldschmidt
  * sys_arch.txt: Fixed documentation after changing sys arch prototypes for  ↩
     1.4.0.

  2011-06-26: Simon Goldschmidt
  * tcp.c: fixed bug #31723 (tcp_kill_prio() kills pcbs with the same prio) by
    updating its documentation only.

 2011-06-26: Simon Goldschmidt
  * mem.c: fixed bug #33545: With MEM_USE_POOLS==1, mem_malloc can return an
    unaligned pointer.

  2011-06-26: Simon Goldschmidt
  * mem.c: fixed bug #33544 "warning in mem.c in lwip 1.4.0 with NO_SYS=1"

   2011-05-25: Simon Goldschmidt
  * tcp.c: fixed bug #33398 (pointless conversion when checking TCP port range)



(STABLE-1.4.0)

  ++ New features:

  2011-03-27: Simon Goldschmidt
  * tcp_impl.h, tcp_in.c, tcp_out.c: Removed 'dataptr' from 'struct tcp_seg' and
    calculate it in tcp_zero_window_probe (the only place where it was used).

  2010-11-21: Simon Goldschmidt
  * dhcp.c/.h: Added a function to deallocate the struct dhcp from a netif
    (fixes bug #31525).
```

```
2010-07-12: Simon Goldschmidt (patch by Stephane Lesage)
* ip.c, udp.c/.h, pbuf.h, sockets.c: task #10495: Added support for
  IP_MULTICAST_LOOP at socket- and raw-API level.

2010-06-16: Simon Goldschmidt
* ip.c: Added an optional define (LWIP_IP_ACCEPT_UDP_PORT) that can allow
  link-layer-addressed UDP traffic to be received while a netif is down (just
  like DHCP during configuration)

2010-05-22: Simon Goldschmidt
* many many files: bug #27352: removed packing from ip_addr_t, the packed
  version is now only used in protocol headers. Added global storage for
  current src/dest IP address while in input functions.

2010-05-16: Simon Goldschmidt
* def.h: task #10391: Add preprocessor-macros for compile-time htonl
  calculation (and use them throughout the stack where applicable)

2010-05-16: Simon Goldschmidt
* opt.h, memp_std.h, memp.c, ppp_oe.h/.c: PPPoE now uses its own MEMP pool
  instead of the heap (moved struct pppoe_softc from ppp_oe.c to ppp_oe.h)

2010-05-16: Simon Goldschmidt
* opt.h, memp_std.h, dns.h/.c: DNS_LOCAL_HOSTLIST_IS_DYNAMIC uses its own
  MEMP pool instead of the heap

2010-05-13: Simon Goldschmidt
* tcp.c, udp.c: task #6995: Implement SO_REUSEADDR (correctly), added
  new option SO_REUSE_RXTOALL to pass received UDP broadcast/multicast
  packets to more than one pcb.

2010-05-02: Simon Goldschmidt
* netbuf.h/.c, sockets.c, api_msg.c: use checksum-on-copy for sending
  UDP data for LWIP_NETIF_TX_SINGLE_PBUF==1

2010-04-30: Simon Goldschmidt
* udp.h/.c, pbuf.h/.c: task #6849: added udp_send(_to/_if) functions that
  take a precalculated checksum, added pbuf_fill_chksum() to copy data
  into a pbuf and at the same time calculating the checksum for that data

2010-04-29: Simon Goldschmidt
* ip_addr.h, etharp.h/.c, autoip.c: Create overridable macros for copying
  2-byte-aligned IP addresses and MAC addresses

2010-04-28: Patch by Bill Auerbach
* ip.c: Inline generating IP checksum to save a function call

2010-04-14: Simon Goldschmidt
* tcpip.h/.c, timers.c: Added an overridable define to get informed when the
  tcpip_thread processes messages or timeouts to implement a watchdog.

2010-03-28: Simon Goldschmidt
* ip_frag.c: create a new (contiguous) PBUF_RAM for every outgoing
  fragment if LWIP_NETIF_TX_SINGLE_PBUF==1

2010-03-27: Simon Goldschmidt
```

```
* etharp.c: Speedup TX by moving code from find_entry to etharp_output/
  etharp_query to prevent unnecessary function calls (inspired by
  patch #7135).

2010-03-20: Simon Goldschmidt
* opt.h, tcpip.c/.h: Added an option to disable tcpip_(un)timeout code
  since the linker cannot do this automatically to save space.

2010-03-20: Simon Goldschmidt
* opt.h, etharp.c/.h: Added support for static ARP table entries

2010-03-14: Simon Goldschmidt
* tcp_impl.h, tcp_out.c, inet_chksum.h/.c: task #6849: Calculate checksum
  when creating TCP segments, not when (re-)transmitting them.

2010-03-07: Simon Goldschmidt
* sockets.c: bug #28775 (select/event_callback: only check select_cb_list
  on change) plus use SYS_LIGHTWEIGHT_PROT to protect the select code.
  This should speed up receiving data on sockets as the select code in
  event_callback is only executed when select is waiting.

2010-03-06: Simon Goldschmidt
* tcp_out.c: task #7013 (Create option to have all packets delivered to
  netif->output in one piece): Always copy to try to create single pbufs
  in tcp_write.

2010-03-06: Simon Goldschmidt
* api.h, api_lib.c, sockets.c: task #10167 (sockets: speed up TCP recv
  by not allocating a netbuf): added function netconn_recv_tcp_pbuf()
  for tcp netconns to receive pbufs, not netbufs; use that function
  for tcp sockets.

2010-03-05: Jakob Ole Stoklundsen / Simon Goldschmidt
* opt.h, tcp.h, tcp_impl.h, tcp.c, tcp_in.c, tcp_out.c: task #7040:
  Work on tcp_enqueue: Don't waste memory when chaining segments,
  added option TCP_OVERSIZE to prevent creating many small pbufs when
  calling tcp_write with many small blocks of data. Instead, pbufs are
  allocated larger than needed and the space is used for later calls to
  tcp_write.

2010-02-21: Simon Goldschmidt
* stats.c/.h: Added const char* name to mem- and memp-stats for easier
  debugging.

2010-02-21: Simon Goldschmidt
* tcp.h (and usages), added tcp_impl.h: Splitted API and internal
  implementation of tcp to make API usage cleare to application programmers

2010-02-14: Simon Goldschmidt/Stephane Lesage
* ip_addr.h: Improved some defines working on ip addresses, added faster
  macro to copy addresses that cannot be NULL

2010-02-13: Simon Goldschmidt
* api.h, api_lib.c, api_msg.c, sockets.c: task #7865 (implement non-
  blocking send operation)

2010-02-12: Simon Goldschmidt
```

```
* sockets.c/.h: Added a minimal version of posix fctl() to have a
  standardised way to set O_NONBLOCK for nonblocking sockets.

2010-02-12: Simon Goldschmidt
* dhcp.c/.h, autoip.c/.h: task #10139 (Prefer statically allocated
  memory): added autoip_set_struct() and dhcp_set_struct() to let autoip
  and dhcp work with user-allocated structs instead of callin mem_malloc

2010-02-12: Simon Goldschmidt/Jeff Barber
* tcp.c/h: patch #6865 (SO_REUSEADDR for TCP): if pcb.so_options has
  SOF_REUSEADDR set, allow binding to endpoint in TIME_WAIT

2010-02-12: Simon Goldschmidt
* sys layer: task #10139 (Prefer statically allocated memory): converted
  mbox and semaphore functions to take pointers to sys_mbox_t/sys_sem_t;
  converted sys_mbox_new/sys_sem_new to take pointers and return err_t;
  task #7212: Add Mutex concept in sys_arch (define LWIP_COMPAT_MUTEX
  to let sys.h use binary semaphores instead of mutexes - as before)

2010-02-09: Simon Goldschmidt (Simon Kallweit)
* timers.c/.h: Added function sys_restart_timeouts() from patch #7085
  (Restart system timeout handling)

2010-02-09: Simon Goldschmidt
* netif.c/.h, removed loopif.c/.h: task #10153 (Integrate loopif into
  netif.c) - loopif does not have to be created by the port any more,
  just define LWIP_HAVE_LOOPIF to 1.

2010-02-08: Simon Goldschmidt
* inet.h, ip_addr.c/.h: Added reentrant versions of inet_ntoa/ipaddr_ntoa
  inet_ntoa_r/ipaddr_ntoa_r

2010-02-08: Simon Goldschmidt
* netif.h: Added netif_s/get_igmp_mac_filter() macros

2010-02-05: Simon Goldschmidt
* netif.h: Added function-like macros to get/set the hostname on a netif

2010-02-04: Simon Goldschmidt
* nearly every file: Replaced struct ip_addr by typedef ip_addr_t to
  make changing the actual implementation behind the typedef easier.

2010-02-01: Simon Goldschmidt
* opt.h, memp_std.h, dns.h, netdb.c, memp.c: Let netdb use a memp pool
  for allocating memory when getaddrinfo() is called.

2010-01-31: Simon Goldschmidt
* dhcp.h, dhcp.c: Reworked the code that parses DHCP options: parse
  them once instead of parsing for every option. This also removes
  the need for mem_malloc from dhcp_recv and makes it possible to
  correctly retrieve the BOOTP file.

2010-01-30: simon Goldschmidt
* sockets.c: Use SYS_LIGHTWEIGHT_PROT instead of a semaphore to protect
  the sockets array.

2010-01-29: Simon Goldschmidt (patch by Laura Garrett)
```

```
* api.h, api_msg.c, sockets.c: Added except set support in select
  (patch #6860)

2010-01-29: Simon Goldschmidt (patch by Laura Garrett)
* api.h, sockets.h, err.h, api_lib.c, api_msg.c, sockets.c, err.c:
  Add non-blocking support for connect (partly from patch #6860),
  plus many cleanups in socket & netconn API.

2010-01-27: Simon Goldschmidt
* opt.h, tcp.h, init.c, api_msg.c: Added TCP_SNDQUEUELOWAT corresponding
  to TCP_SNDLOWAT and added tcp_sndqueuelen() - this fixes bug #28605

2010-01-26: Simon Goldschmidt
* snmp: Use memp pools for snmp instead of the heap; added 4 new pools.

2010-01-14: Simon Goldschmidt
* ppp.c/.h: Fixed bug #27856: PPP: Set netif link- and status-callback
  by adding ppp_set_netif_statuscallback()/ppp_set_netif_linkcallback()

2010-01-13: Simon Goldschmidt
* mem.c: The heap now may be moved to user-defined memory by defining
  LWIP_RAM_HEAP_POINTER as a void pointer to that memory's address
  (patch #6966 and bug #26133)

2010-01-10: Simon Goldschmidt (Bill Auerbach)
* opt.h, memp.c: patch #6822 (Add option to place memory pools in
  separate arrays)

2010-01-10: Simon Goldschmidt
* init.c, igmp.c: patch #6463 (IGMP - Adding Random Delay): added define
  LWIP_RAND() for lwip-wide randomization (to be defined in cc.h)

2009-12-31: Simon Goldschmidt
* tcpip.c, init.c, memp.c, sys.c, memp_std.h, sys.h, tcpip.h
  added timers.c/.h: Separated timer implementation from semaphore/mbox
  implementation, moved timer implementation to timers.c/.h, timers are
  now only called from tcpip_thread or by explicitly checking them.
  (TASK#7235)

2009-12-27: Simon Goldschmidt
* opt.h, etharp.h/.c, init.c, tcpip.c: Added an additional option
  LWIP_ETHERNET to support ethernet without ARP (necessary for pure PPPoE)


++ Bugfixes:

2011-04-20: Simon Goldschmidt
* sys_arch.txt: sys_arch_timeouts() is not needed any more.

2011-04-13: Simon Goldschmidt
* tcp.c, udp.c: Fixed bug #33048 (Bad range for IP source port numbers) by
  using ports in the IANA private/dynamic range (49152 through 65535).

2011-03-29: Simon Goldschmidt, patch by Emil Lhungdahl:
* etharp.h/.c: Fixed broken VLAN support.

2011-03-27: Simon Goldschmidt
```

```
* tcp.c: Fixed bug #32926 (TCP_RMV(&tcp_bound_pcbs) is called on unbound tcp
  pcbs) by checking if the pcb was bound (local_port != 0).

2011-03-27: Simon Goldschmidt
* ppp.c: Fixed bug #32280 (ppp: a pbuf is freed twice)

2011-03-27: Simon Goldschmidt
* sockets.c: Fixed bug #32906: lwip_connect+lwip_send did not work for udp and
  raw pcbs with LWIP_TCPIP_CORE_LOCKING==1.

2011-03-27: Simon Goldschmidt
* tcp_out.c: Fixed bug #32820 (Outgoing TCP connections created before route
  is present never times out) by starting retransmission timer before checking
  route.

2011-03-22: Simon Goldschmidt
* ppp.c: Fixed bug #32648 (PPP code crashes when terminating a link) by only
  calling sio_read_abort() if the file descriptor is valid.

2011-03-14: Simon Goldschmidt
* err.h/.c, sockets.c, api_msg.c: fixed bug #31748 (Calling non-blocking connect
  more than once can render a socket useless) since it mainly involves changing
  "FATAL" classification of error codes: ERR_USE and ERR_ISCONN just aren't  ↩
     fatal.

2011-03-13: Simon Goldschmidt
* sockets.c: fixed bug #32769 (ESHUTDOWN is linux-specific) by fixing
  err_to_errno_table (ERR_CLSD: ENOTCONN instead of ESHUTDOWN), ERR_ISCONN:
  use EALRADY instead of -1

2011-03-13: Simon Goldschmidt
* api_lib.c: netconn_accept: return ERR_ABRT instead of ERR_CLSD if the
  connection has been aborted by err_tcp (since this is not a normal closing
  procedure).

2011-03-13: Simon Goldschmidt
* tcp.c: tcp_bind: return ERR_VAL instead of ERR_ISCONN when trying to bind
  with pcb->state != CLOSED

2011-02-17: Simon Goldschmidt
* rawapi.txt: Fixed bug #32561 tcp_poll argument definition out-of-order in
  documentation

2011-02-17: Simon Goldschmidt
* many files: Added missing U/UL modifiers to fix 16-bit-arch portability.

2011-01-24: Simon Goldschmidt
* sockets.c: Fixed bug #31741: lwip_select seems to have threading problems

2010-12-02: Simon Goldschmidt
* err.h: Fixed ERR_IS_FATAL so that ERR_WOULDBLOCK is not fatal.

2010-11-23: Simon Goldschmidt
* api.h, api_lib.c, api_msg.c, sockets.c: netconn.recv_avail is only used for
  LWIP_SO_RCVBUF and ioctl/FIONREAD.

2010-11-23: Simon Goldschmidt
```

```
* etharp.c: Fixed bug #31720: ARP-queueing: RFC 1122 recommends to queue at
  least 1 packet -> ARP_QUEUEING==0 now queues the most recent packet.

2010-11-23: Simon Goldschmidt
* tcp_in.c: Fixed bug #30577: tcp_input: don't discard ACK-only packets after
  refusing 'refused_data' again.

2010-11-22: Simon Goldschmidt
* sockets.c: Fixed bug #31590: getsockopt(... SO_ERROR ...) gives EINPROGRESS
  after a successful nonblocking connection.

2010-11-22: Simon Goldschmidt
* etharp.c: Fixed bug #31722: IP packets sent with an AutoIP source addr
  must be sent link-local

2010-11-22: Simon Goldschmidt
* timers.c: patch #7329: tcp_timer_needed prototype was ifdef'ed out for
  LWIP_TIMERS==0

2010-11-20: Simon Goldschmidt
* sockets.c: Fixed bug #31170: lwip_setsockopt() does not set socket number

2010-11-20: Simon Goldschmidt
* sockets.h: Fixed bug #31304: Changed SHUT_RD, SHUT_WR and SHUT_RDWR to
  resemble other stacks.

2010-11-20: Simon Goldschmidt
* dns.c: Fixed bug #31535: TCP_SND_QUEUELEN must be at least 2 or else
  no-copy TCP writes will never succeed.

2010-11-20: Simon Goldschmidt
* dns.c: Fixed bug #31701: Error return value from dns_gethostbyname() does
  not match documentation: return ERR_ARG instead of ERR_VAL if not
  initialized or wrong argument.

2010-10-20: Simon Goldschmidt
* sockets.h: Fixed bug #31385: sizeof(struct sockaddr) is 30 but should be 16

2010-10-05: Simon Goldschmidt
* dhcp.c: Once again fixed #30038: DHCP/AutoIP cooperation failed when
  replugging the network cable after an AutoIP address was assigned.

2010-08-10: Simon Goldschmidt
* tcp.c: Fixed bug #30728: tcp_new_port() did not check listen pcbs

2010-08-03: Simon Goldschmidt
* udp.c, raw.c: Don't chain empty pbufs when sending them (fixes bug #30625)

2010-08-01: Simon Goldschmidt (patch by Greg Renda)
* ppp.c: Applied patch #7264 (PPP protocols are rejected incorrectly on big
  endian architectures)

2010-07-28: Simon Goldschmidt
* api_lib.c, api_msg.c, sockets.c, mib2.c: Fixed compilation with TCP or UDP
  disabled.

2010-07-27: Simon Goldschmidt
```

```
* tcp.c: Fixed bug #30565 (tcp_connect() check bound list): that check did no
  harm but never did anything

2010-07-21: Simon Goldschmidt
* ip.c: Fixed invalid fix for bug #30402 (CHECKSUM_GEN_IP_INLINE does not
  add IP options)

2010-07-16: Kieran Mansley
* msg_in.c: Fixed SNMP ASN constant defines to not use ! operator

2010-07-10: Simon Goldschmidt
* ip.c: Fixed bug #30402: CHECKSUM_GEN_IP_INLINE does not add IP options

2010-06-30: Simon Goldschmidt
* api_msg.c: fixed bug #30300 (shutdown parameter was not initialized in
  netconn_delete)

2010-06-28: Kieran Mansley
* timers.c remove unportable printing of C function pointers

2010-06-24: Simon Goldschmidt
* init.c, timers.c/.h, opt.h, memp_std.h: From patch #7221: added flag
  NO_SYS_NO_TIMERS to drop timer support for NO_SYS==1 for easier upgrading

2010-06-24: Simon Goldschmidt
* api(_lib).c/.h, api_msg.c/.h, sockets.c/.h: Fixed bug #10088: Correctly
  implemented shutdown at socket level.

2010-06-21: Simon Goldschmidt
* pbuf.c/.h, ip_frag.c/.h, opt.h, memp_std.h: Fixed bug #29361 (ip_frag has
  problems with zero-copy DMA MACs) by adding custom pbufs and implementing
  custom pbufs that reference other (original) pbufs. Additionally set
  IP_FRAG_USES_STATIC_BUF=0 as default to be on the safe side.

2010-06-15: Simon Goldschmidt
* dhcp.c: Fixed bug #29970: DHCP endian issue parsing option responses

2010-06-14: Simon Goldschmidt
* autoip.c: Fixed bug #30039: AutoIP does not reuse previous addresses

2010-06-12: Simon Goldschmidt
* dhcp.c: Fixed bug #30038: dhcp_network_changed doesn't reset AUTOIP coop
  state

2010-05-17: Simon Goldschmidt
* netdb.c: Correctly NULL-terminate h_addr_list

2010-05-16: Simon Goldschmidt
* def.h/.c: changed the semantics of LWIP_PREFIX_BYTEORDER_FUNCS to prevent
  "symbol already defined" i.e. when linking to winsock

2010-05-05: Simon Goldschmidt
* def.h, timers.c: Fixed bug #29769 (sys_check_timeouts: sys_now() may
  overflow)

2010-04-21: Simon Goldschmidt
* api_msg.c: Fixed bug #29617 (sometime cause stall on delete listening
  connection)

2010-03-28: Luca Ceresoli
* ip_addr.c/.h: patch #7143: Add a few missing const qualifiers

2010-03-27: Luca Ceresoli
* mib2.c: patch #7130: remove meaningless const qualifiers

2010-03-26: Simon Goldschmidt
* tcp_out.c: Make LWIP_NETIF_TX_SINGLE_PBUF work for TCP, too

2010-03-26: Simon Goldschmidt
* various files: Fixed compiling with different options disabled (TCP/UDP),
  triggered by bug #29345; don't allocate acceptmbox if LWIP_TCP is disabled

2010-03-25: Simon Goldschmidt
* sockets.c: Fixed bug #29332: lwip_select() processes readset incorrectly

2010-03-25: Simon Goldschmidt
* tcp_in.c, test_tcp_oos.c: Fixed bug #29080: Correctly handle remote side
  overrunning our rcv_wnd in ooseq case.

2010-03-22: Simon Goldschmidt
* tcp.c: tcp_listen() did not copy the pcb's prio.

2010-03-19: Simon Goldschmidt
* snmp_msg.c: Fixed bug #29256: SNMP Trap address was not correctly set

2010-03-14: Simon Goldschmidt
* opt.h, etharp.h: Fixed bug #29148 (Incorrect PBUF_POOL_BUFSIZE for ports
  where ETH_PAD_SIZE > 0) by moving definition of ETH_PAD_SIZE to opt.h
  and basing PBUF_LINK_HLEN on it.

2010-03-08: Simon Goldschmidt
* netif.c, ipv4/ip.c: task #10241 (AutoIP: don't break existing connections
  when assiging routable address): when checking incoming packets and
  aborting existing connection on address change, filter out link-local
  addresses.

2010-03-06: Simon Goldschmidt
* sockets.c: Fixed LWIP_NETIF_TX_SINGLE_PBUF for LWIP_TCPIP_CORE_LOCKING

2010-03-06: Simon Goldschmidt
* ipv4/ip.c: Don't try to forward link-local addresses

2010-03-06: Simon Goldschmidt
* etharp.c: Fixed bug #29087: etharp: don't send packets for LinkLocal-
  addresses to gw

2010-03-05: Simon Goldschmidt
* dhcp.c: Fixed bug #29072: Correctly set ciaddr based on message-type
  and state.

2010-03-05: Simon Goldschmidt
* api_msg.c: Correctly set TCP_WRITE_FLAG_MORE when netconn_write is split
  into multiple calls to tcp_write.
```

```
2010-02-21: Simon Goldschmidt
* opt.h, mem.h, dns.c: task #10140: Remove DNS_USES_STATIC_BUF (keep
  the implementation of DNS_USES_STATIC_BUF==1)

2010-02-20: Simon Goldschmidt
* tcp.h, tcp.c, tcp_in.c, tcp_out.c: Task #10088: Correctly implement
  close() vs. shutdown(). Now the application does not get any more
  recv callbacks after calling tcp_close(). Added tcp_shutdown().

2010-02-19: Simon Goldschmidt
* mem.c/.h, pbuf.c: Renamed mem_realloc() to mem_trim() to prevent
  confusion with realloc()

2010-02-15: Simon Goldschmidt/Stephane Lesage
* netif.c/.h: Link status does not depend on LWIP_NETIF_LINK_CALLBACK
  (fixes bug #28899)

2010-02-14: Simon Goldschmidt
* netif.c: Fixed bug #28877 (Duplicate ARP gratuitous packet with
  LWIP_NETIF_LINK_CALLBACK set on) by only sending if both link- and
  admin-status of a netif are up

2010-02-14: Simon Goldschmidt
* opt.h: Disable ETHARP_TRUST_IP_MAC by default since it slows down packet
  reception and is not really necessary

2010-02-14: Simon Goldschmidt
* etharp.c/.h: Fixed ARP input processing: only add a new entry if a
  request was directed as us (RFC 826, Packet Reception), otherwise
  only update existing entries; internalized some functions

2010-02-14: Simon Goldschmidt
* netif.h, etharp.c, tcpip.c: Fixed bug #28183 (ARP and TCP/IP cannot be
  disabled on netif used for PPPoE) by adding a new netif flag
  (NETIF_FLAG_ETHERNET) that tells the stack the device is an ethernet
  device but prevents usage of ARP (so that ethernet_input can be used
  for PPPoE).

2010-02-12: Simon Goldschmidt
* netif.c: netif_set_link_up/down: only do something if the link state
  actually changes

2010-02-12: Simon Goldschmidt/Stephane Lesage
* api_msg.c: Fixed bug #28865 (Cannot close socket/netconn in non-blocking
  connect)

2010-02-12: Simon Goldschmidt
* mem.h: Fixed bug #28866 (mem_realloc function defined in mem.h)

2010-02-09: Simon Goldschmidt
* api_lib.c, api_msg.c, sockets.c, api.h, api_msg.h: Fixed bug #22110
 (recv() makes receive window update for data that wasn't received by
  application)

2010-02-09: Simon Goldschmidt/Stephane Lesage
* sockets.c: Fixed bug #28853 (lwip_recvfrom() returns 0 on receive time-out
  or any netconn_recv() error)
```

```
2010-02-09: Simon Goldschmidt
* ppp.c: task #10154 (PPP: Update snmp in/out counters for tx/rx packets)

2010-02-09: Simon Goldschmidt
* netif.c: For loopback packets, adjust the stats- and snmp-counters
  for the loopback netif.

2010-02-08: Simon Goldschmidt
* igmp.c/.h, ip.h: Moved most defines from igmp.h to igmp.c for clarity
  since they are not used anywhere else.

2010-02-08: Simon Goldschmidt (Stéphane Lesage)
* igmp.c, igmp.h, stats.c, stats.h: Improved IGMP stats
  (patch from bug #28798)

2010-02-08: Simon Goldschmidt (Stéphane Lesage)
* igmp.c: Fixed bug #28798 (Error in "Max Response Time" processing) and
  another bug when LWIP_RAND() returns zero.

2010-02-04: Simon Goldschmidt
* nearly every file: Use macros defined in ip_addr.h (some of them new)
  to work with IP addresses (preparation for bug #27352 - Change ip_addr
  from struct to typedef (u32_t) - and better code).

2010-01-31: Simon Goldschmidt
* netif.c: Don’t call the link-callback from netif_set_up/down() since
  this invalidly retriggers DHCP.

2010-01-29: Simon Goldschmidt
* ip_addr.h, inet.h, def.h, inet.c, def.c, more: Cleanly separate the
  portability file inet.h and its contents from the stack: moved htonX-
  functions to def.h (and the new def.c - they are not ipv4 dependent),
  let inet.h depend on ip_addr.h and not the other way round.
  This fixes bug #28732.

2010-01-28: Kieran Mansley
* tcp.c: Ensure ssthresh >= 2*MSS

2010-01-27: Simon Goldschmidt
* tcp.h, tcp.c, tcp_in.c: Fixed bug #27871: Calling tcp_abort() in recv
  callback can lead to accessing unallocated memory. As a consequence,
  ERR_ABRT means the application has called tcp_abort()!

2010-01-25: Simon Goldschmidt
* snmp_structs.h, msg_in.c: Partly fixed bug #22070 (MIB_OBJECT_WRITE_ONLY
  not implemented in SNMP): write-only or not-accessible are still
  returned by getnext (though not by get)

2010-01-24: Simon Goldschmidt
* snmp: Renamed the private mib node from ’private’ to ’mib_private’ to
  not use reserved C/C++ keywords

2010-01-23: Simon Goldschmidt
* sockets.c: Fixed bug #28716: select() returns 0 after waiting for less
  than 1 ms
```

```
2010-01-21: Simon Goldschmidt
* tcp.c, api_msg.c: Fixed bug #28651 (tcp_connect: no callbacks called
  if tcp_enqueue fails) both in raw- and netconn-API

2010-01-19: Simon Goldschmidt
* api_msg.c: Fixed bug #27316: netconn: Possible deadlock in err_tcp

2010-01-18: Iordan Neshev/Simon Goldschmidt
* src/netif/ppp: reorganised PPP sourcecode to 2.3.11 including some
  bugfix backports from 2.4.x.

2010-01-18: Simon Goldschmidt
* mem.c: Fixed bug #28679: mem_realloc calculates mem_stats wrong

2010-01-17: Simon Goldschmidt
* api_lib.c, api_msg.c, (api_msg.h, api.h, sockets.c, tcpip.c):
  task #10102: "netconn: clean up conn->err threading issues" by adding
  error return value to struct api_msg_msg

2010-01-17: Simon Goldschmidt
* api.h, api_lib.c, sockets.c: Changed netconn_recv() and netconn_accept()
  to return err_t (bugs #27709 and #28087)

2010-01-14: Simon Goldschmidt
* ...: Use typedef for function prototypes throughout the stack.

2010-01-13: Simon Goldschmidt
* api_msg.h/.c, api_lib.c: Fixed bug #26672 (close connection when receive
  window = 0) by correctly draining recvmbox/acceptmbox

2010-01-11: Simon Goldschmidt
* pap.c: Fixed bug #13315 (PPP PAP authentication can result in
  erroneous callbacks) by copying the code from recent pppd

2010-01-10: Simon Goldschmidt
* raw.c: Fixed bug #28506 (raw_bind should filter received packets)

2010-01-10: Simon Goldschmidt
* tcp.h/.c: bug #28127 (remove call to tcp_output() from tcp_ack(_now)())

2010-01-08: Simon Goldschmidt
* sockets.c: Fixed bug #28519 (lwip_recvfrom bug with len > 65535)

2010-01-08: Simon Goldschmidt
* dns.c: Copy hostname for DNS_LOCAL_HOSTLIST_IS_DYNAMIC==1 since string
  passed to dns_local_addhost() might be volatile

2010-01-07: Simon Goldschmidt
* timers.c, tcp.h: Call tcp_timer_needed() with NO_SYS==1, too

2010-01-06: Simon Goldschmidt
* netdb.h: Fixed bug #28496: missing include guards in netdb.h

2009-12-31: Simon Goldschmidt
* many ppp files: Reorganised PPP source code from ucip structure to pppd
  structure to easily compare our code against the pppd code (around v2.3.1)
  2009-12-27: Simon Goldschmidt
  * tcp_in.c: Another fix for bug #28241 (ooseq processing) and adapted
    unit test


(STABLE-1.3.2)

  ++ New features:

  2009-10-27 Simon Goldschmidt/Stephan Lesage
  * netifapi.c/.h: Added netifapi_netif_set_addr()

  2009-10-07 Simon Goldschmidt/Fabian Koch
  * api_msg.c, netbuf.c/.h, opt.h: patch #6888: Patch for UDP Netbufs to
    support dest-addr and dest-port (optional: LWIP_NETBUF_RECVINFO)

  2009-08-26 Simon Goldschmidt/Simon Kallweit
  * slipif.c/.h: bug #26397: SLIP polling support

  2009-08-25 Simon Goldschmidt
  * opt.h, etharp.h/.c: task #9033: Support IEEE 802.1q tagged frame (VLAN),
    New configuration options ETHARP_SUPPORT_VLAN and ETHARP_VLAN_CHECK.

  2009-08-25 Simon Goldschmidt
  * ip_addr.h, netdb.c: patch #6900: added define ip_ntoa(struct ip_addr*)

  2009-08-24 Jakob Stoklund Olesen
  * autoip.c, dhcp.c, netif.c: patch #6725: Teach AutoIP and DHCP to respond
    to netif_set_link_up().

  2009-08-23 Simon Goldschmidt
  * tcp.h/.c: Added function tcp_debug_state_str() to convert a tcp state
    to a human-readable string.

  ++ Bugfixes:

  2009-12-24: Kieran Mansley
  * tcp_in.c Apply patches from Oleg Tyshev to improve OOS processing
    (BUG#28241)

  2009-12-06: Simon Goldschmidt
  * ppp.h/.c: Fixed bug #27079 (Yet another leak in PPP): outpacket_buf can
    be statically allocated (like in ucip)

  2009-12-04: Simon Goldschmidt (patch by Ioardan Neshev)
  * pap.c: patch #6969: PPP: missing PAP authentication UNTIMEOUT

  2009-12-03: Simon Goldschmidt
  * tcp.h, tcp_in.c, tcp_out.c: Fixed bug #28106: dup ack for fast retransmit
    could have non-zero length

  2009-12-02: Simon Goldschmidt
  * tcp_in.c: Fixed bug #27904: TCP sends too many ACKs: delay resetting
    tcp_input_pcb until after calling the pcb's callbacks

  2009-11-29: Simon Goldschmidt
  * tcp_in.c: Fixed bug #28054: Two segments with FIN flag on the out-of-
```

```
    sequence queue, also fixed PBUF_POOL leak in the out-of-sequence code

  2009-11-29: Simon Goldschmidt
  * pbuf.c: Fixed bug #28064: pbuf_alloc(PBUF_POOL) is not thread-safe by
    queueing a call into tcpip_thread to free ooseq-bufs if the pool is empty

  2009-11-26: Simon Goldschmidt
  * tcp.h: Fixed bug #28098: Nagle can prevent fast retransmit from sending
    segment

  2009-11-26: Simon Goldschmidt
  * tcp.h, sockets.c: Fixed bug #28099: API required to disable Nagle
    algorithm at PCB level

  2009-11-22: Simon Goldschmidt
  * tcp_out.c: Fixed bug #27905: FIN isn't combined with data on unsent

  2009-11-22: Simon Goldschmidt (suggested by Bill Auerbach)
  * tcp.c: tcp_alloc: prevent increasing stats.err for MEMP_TCP_PCB when
    reusing time-wait pcb

  2009-11-20: Simon Goldschmidt (patch by Albert Bartel)
  * sockets.c: Fixed bug #28062: Data received directly after accepting
    does not wake up select

  2009-11-11: Simon Goldschmidt
  * netdb.h: Fixed bug #27994: incorrect define for freeaddrinfo(addrinfo)

  2009-10-30: Simon Goldschmidt
  * opt.h: Increased default value for TCP_MSS to 536, updated default
    value for TCP_WND to 4*TCP_MSS to keep delayed ACK working.

  2009-10-28: Kieran Mansley
  * tcp_in.c, tcp_out.c, tcp.h: re-work the fast retransmission code
    to follow algorithm from TCP/IP Illustrated

  2009-10-27: Kieran Mansley
  * tcp_in.c: fix BUG#27445: grow cwnd with every duplicate ACK

  2009-10-25: Simon Goldschmidt
  * tcp.h: bug-fix in the TCP_EVENT_RECV macro (has to call tcp_recved if
    pcb->recv is NULL to keep rcv_wnd correct)

  2009-10-25: Simon Goldschmidt
  * tcp_in.c: Fixed bug #26251: RST process in TIME_WAIT TCP state

  2009-10-23: Simon Goldschmidt (David Empson)
  * tcp.c: Fixed bug #27783: Silly window avoidance for small window sizes

  2009-10-21: Simon Goldschmidt
  * tcp_in.c: Fixed bug #27215: TCP sent() callback gives leading and
    trailing 1 byte len (SYN/FIN)

  2009-10-21: Simon Goldschmidt
  * tcp_out.c: Fixed bug #27315: zero window probe and FIN

  2009-10-19: Simon Goldschmidt
```

```
* dhcp.c/.h: Minor code simplification (don't store received pbuf, change
  conditional code to assert where applicable), check pbuf length before
  testing for valid reply

2009-10-19: Simon Goldschmidt
* dhcp.c: Removed most calls to udp_connect since they aren't necessary
  when using udp_sendto_if() - always stay connected to IP_ADDR_ANY.

2009-10-16: Simon Goldschmidt
* ip.c: Fixed bug #27390: Source IP check in ip_input() causes it to drop
  valid DHCP packets -> allow 0.0.0.0 as source address when LWIP_DHCP is
  enabled

2009-10-15: Simon Goldschmidt (Oleg Tyshev)
* tcp_in.c: Fixed bug #27329: dupacks by unidirectional data transmit

2009-10-15: Simon Goldschmidt
* api_lib.c: Fixed bug #27709: conn->err race condition on netconn_recv()
  timeout

2009-10-15: Simon Goldschmidt
* autoip.c: Fixed bug #27704: autoip starts with wrong address
  LWIP_AUTOIP_CREATE_SEED_ADDR() returned address in host byte order instead
  of network byte order

2009-10-11 Simon Goldschmidt (Jörg Kesten)
* tcp_out.c: Fixed bug #27504: tcp_enqueue wrongly concatenates segments
  which are not consecutive when retransmitting unacked segments

2009-10-09 Simon Goldschmidt
* opt.h: Fixed default values of some stats to only be enabled if used
  Fixes bug #27338: sys_stats is defined when NO_SYS = 1

2009-08-30 Simon Goldschmidt
* ip.c: Fixed bug bug #27345: "ip_frag() does not use the LWIP_NETIF_LOOPBACK
  function" by checking for loopback before calling ip_frag

2009-08-25 Simon Goldschmidt
* dhcp.c: fixed invalid dependency to etharp_query if DHCP_DOES_ARP_CHECK==0

2009-08-23 Simon Goldschmidt
* ppp.c: bug #27078: Possible memory leak in pppInit()

2009-08-23 Simon Goldschmidt
* netdb.c, dns.c: bug #26657: DNS, if host name is "localhost", result
  is error.

2009-08-23 Simon Goldschmidt
* opt.h, init.c: bug #26649: TCP fails when TCP_MSS > TCP_SND_BUF
  Fixed wrong parenthesis, added check in init.c

2009-08-23 Simon Goldschmidt
* ppp.c: bug #27266: wait-state debug message in pppMain occurs every ms

2009-08-23 Simon Goldschmidt
* many ppp files: bug #27267: Added include to string.h where needed
```

```
  2009-08-23 Simon Goldschmidt
  * tcp.h: patch #6843: tcp.h macro optimization patch (for little endian)


(STABLE-1.3.1)

  ++ New features:

  2009-05-10 Simon Goldschmidt
  * opt.h, sockets.c, pbuf.c, netbuf.h, pbuf.h: task #7013: Added option
    LWIP_NETIF_TX_SINGLE_PBUF to try to create transmit packets from only
    one pbuf to help MACs that don't support scatter-gather DMA.

  2009-05-09 Simon Goldschmidt
  * icmp.h, icmp.c: Shrinked ICMP code, added option to NOT check icoming
    ECHO pbuf for size (just use it): LWIP_ICMP_ECHO_CHECK_INPUT_PBUF_LEN

  2009-05-05 Simon Goldschmidt, Jakob Stoklund Olesen
  * ip.h, ip.c: Added ip_current_netif() & ip_current_header() to receive
    extended info about the currently received packet.

  2009-04-27 Simon Goldschmidt
  * sys.h: Made SYS_LIGHTWEIGHT_PROT and sys_now() work with NO_SYS=1

  2009-04-25 Simon Goldschmidt
  * mem.c, opt.h: Added option MEM_USE_POOLS_TRY_BIGGER_POOL to try the next
    bigger malloc pool if one is empty (only usable with MEM_USE_POOLS).

  2009-04-21 Simon Goldschmidt
  * dns.c, init.c, dns.h, opt.h: task #7507, patch #6786: DNS supports static
    hosts table. New configuration options DNS_LOCAL_HOSTLIST and
    DNS_LOCAL_HOSTLIST_IS_DYNAMIC. Also, DNS_LOOKUP_LOCAL_EXTERN() can be defined
    as an external function for lookup.

  2009-04-15 Simon Goldschmidt
  * dhcp.c: patch #6763: Global DHCP XID can be redefined to something more unique

  2009-03-31 Kieran Mansley
  * tcp.c, tcp_out.c, tcp_in.c, sys.h, tcp.h, opts.h: add support for
    TCP timestamp options, off by default.  Rework tcp_enqueue() to
    take option flags rather than specified option data

  2009-02-18 Simon Goldschmidt
  * cc.h: Added printf formatter for size_t: SZT_F

  2009-02-16 Simon Goldschmidt (patch by Rishi Khan)
  * icmp.c, opt.h: patch #6539: (configurable) response to broadcast- and  ↩
     multicast
    pings

  2009-02-12 Simon Goldschmidt
  * init.h: Added LWIP_VERSION to get the current version of the stack

  2009-02-11 Simon Goldschmidt (suggested by Gottfried Spitaler)
  * opt.h, memp.h/.c: added MEMP_MEM_MALLOC to use mem_malloc/mem_free instead
    of the pool allocator (can save code size with MEM_LIBC_MALLOC if libc-malloc
    is otherwise used)
```

```
2009-01-28 Jonathan Larmour (suggested by Bill Bauerbach)
* ipv4/inet_chksum.c, ipv4/lwip/inet_chksum.h: inet_chksum_pseudo_partial()
is only used by UDPLITE at present, so conditionalise it.

2008-12-03 Simon Goldschmidt (base on patch from Luca Ceresoli)
* autoip.c: checked in (slightly modified) patch #6683: Customizable AUTOIP
  "seed" address. This should reduce AUTOIP conflicts if
  LWIP_AUTOIP_CREATE_SEED_ADDR is overridden.

2008-10-02 Jonathan Larmour and Rishi Khan
* sockets.c (lwip_accept): Return EWOULDBLOCK if would block on non-blocking
  socket.

2008-06-30 Simon Goldschmidt
* mem.c, opt.h, stats.h: fixed bug #21433: Calling mem_free/pbuf_free from
  interrupt context isn't safe: LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT allows
  mem_free to run between mem_malloc iterations. Added illegal counter for
  mem stats.

2008-06-27 Simon Goldschmidt
* stats.h/.c, some other files: patch #6483: stats module improvement:
  Added defines to display each module's statistic individually, added stats
  defines for MEM, MEMP and SYS modules, removed (unused) rexmit counter.

2008-06-17 Simon Goldschmidt
* err.h: patch #6459: Made err_t overridable to use a more efficient type
  (define LWIP_ERR_T in cc.h)

2008-06-17 Simon Goldschmidt
* slipif.c: patch #6480: Added a configuration option for slipif for symmetry
  to loopif

2008-06-17 Simon Goldschmidt (patch by Luca Ceresoli)
* netif.c, loopif.c, ip.c, netif.h, loopif.h, opt.h: Checked in slightly
  modified version of patch # 6370: Moved loopif code to netif.c so that
  loopback traffic is supported on all netifs (all local IPs).
  Added option to limit loopback packets for each netifs.


++ Bugfixes:
2009-08-12 Kieran Mansley
* tcp_in.c, tcp.c: Fix bug #27209: handle trimming of segments when
  out of window or out of order properly

2009-08-12 Kieran Mansley
* tcp_in.c: Fix bug #27199: use snd_wl2 instead of snd_wl1

2009-07-28 Simon Goldschmidt
* mem.h: Fixed bug #27105: "realloc() cannot replace mem_realloc()"s

2009-07-27 Kieran Mansley
* api.h api_msg.h netdb.h sockets.h: add missing #include directives

2009-07-09 Kieran Mansley
* api_msg.c, sockets.c, api.h: BUG23240 use signed counters for
  recv_avail and don't increment counters until message successfully
```

```
    sent to mbox

  2009-06-25 Kieran Mansley
  * api_msg.c api.h: BUG26722: initialise netconn write variables
    in netconn_alloc

  2009-06-25 Kieran Mansley
  * tcp.h: BUG26879: set ret value in TCP_EVENT macros when function is not set

  2009-06-25 Kieran Mansley
  * tcp.c, tcp_in.c, tcp_out.c, tcp.h: BUG26301 and BUG26267: correct
    simultaneous close behaviour, and make snd_nxt have the same meaning
    as in the RFCs.

  2009-05-12 Simon Goldschmidt
  * etharp.h, etharp.c, netif.c: fixed bug #26507: "Gratuitous ARP depends on
    arp_table / uses etharp_query" by adding etharp_gratuitous()

  2009-05-12 Simon Goldschmidt
  * ip.h, ip.c, igmp.c: bug #26487: Added ip_output_if_opt that can add IP options
    to the IP header (used by igmp_ip_output_if)

  2009-05-06 Simon Goldschmidt
  * inet_chksum.c: On little endian architectures, use LWIP_PLATFORM_HTONS (if
    defined) for SWAP_BYTES_IN_WORD to speed up checksumming.

  2009-05-05 Simon Goldschmidt
  * sockets.c: bug #26405: Prematurely released semaphore causes lwip_select()
    to crash

  2009-05-04 Simon Goldschmidt
  * init.c: snmp was not initialized in lwip_init()

  2009-05-04 Frédéric Bernon
  * dhcp.c, netbios.c: Changes if IP_SOF_BROADCAST is enabled.

  2009-05-03 Simon Goldschmidt
  * tcp.h: bug #26349: Nagle algorithm doesn’t send although segment is full
    (and unsent->next == NULL)

  2009-05-02 Simon Goldschmidt
  * tcpip.h, tcpip.c: fixed tcpip_untimeout (does not need the time, broken after
    1.3.0 in CVS only) - fixes compilation of ppp_oe.c

  2009-05-02 Simon Goldschmidt
  * msg_in.c: fixed bug #25636: SNMPSET value is ignored for integer fields

  2009-05-01 Simon Goldschmidt
  * pap.c: bug #21680: PPP upap_rauthnak() drops legal NAK packets

  2009-05-01 Simon Goldschmidt
  * ppp.c: bug #24228: Memory corruption with PPP and DHCP

  2009-04-29 Frédéric Bernon
  * raw.c, udp.c, init.c, opt.h, ip.h, sockets.h: bug #26309: Implement the
    SO(F)_BROADCAST filter for all API layers. Avoid the unindented reception
    of broadcast packets even when this option wasn’t set. Port maintainers
    which want to enable this filter have to set IP_SOF_BROADCAST=1 in opt.h.
    If you want this option also filter broadcast on recv operations, you also
    have to set IP_SOF_BROADCAST_RECV=1 in opt.h.

  2009-04-28 Simon Goldschmidt, Jakob Stoklund Olesen
  * dhcp.c: patch #6721, bugs #25575, #25576: Some small fixes to DHCP and
    DHCP/AUTOIP cooperation

  2009-04-25 Simon Goldschmidt, Oleg Tyshev
  * tcp_out.c: bug #24212: Deadlocked tcp_retransmit due to exceeded pcb->cwnd
    Fixed by sorting the unsent and unacked queues (segments are inserted at the
    right place in tcp_output and tcp_rexmit).

  2009-04-25 Simon Goldschmidt
  * memp.c, mem.c, memp.h, mem_std.h: bug #26213 "Problem with memory allocation
    when debugging": memp_sizes contained the wrong sizes (including sanity
    regions); memp pools for MEM_USE_POOLS were too small

  2009-04-24 Simon Goldschmidt, Frédéric Bernon
  * inet.c: patch #6765: Fix a small problem with the last changes (incorrect
    behavior, with with ip address string not ended by a '\0', a space or a
    end of line)

  2009-04-19 Simon Goldschmidt
  * rawapi.txt: Fixed bug #26069: Corrected documentation: if tcp_connect fails,
    pcb->err is called, not pcb->connected (with an error code).

  2009-04-19 Simon Goldschmidt
  * tcp_out.c: Fixed bug #26236: "TCP options (timestamp) don't work with
    no-copy-tcpwrite": deallocate option data, only concat segments with same  ↩
        flags

  2009-04-19 Simon Goldschmidt
  * tcp_out.c: Fixed bug #25094: "Zero-length pbuf" (options are now allocated
    in the header pbuf, not the data pbuf)

  2009-04-18 Simon Goldschmidt
  * api_msg.c: fixed bug #25695: Segmentation fault in do_writemore()

  2009-04-15 Simon Goldschmidt
  * sockets.c: tried to fix bug #23559: lwip_recvfrom problem with tcp

  2009-04-15 Simon Goldschmidt
  * dhcp.c: task #9192: mem_free of dhcp->options_in and dhcp->msg_in

  2009-04-15 Simon Goldschmidt
  * ip.c, ip6.c, tcp_out.c, ip.h: patch #6808: Add a utility function
    ip_hinted_output() (for smaller code mainly)

  2009-04-15 Simon Goldschmidt
  * inet.c: patch #6765: Supporting new line characters in inet_aton()

  2009-04-15 Simon Goldschmidt
  * dhcp.c: patch #6764: DHCP rebind and renew did not send hostnam option;
    Converted constant OPTION_MAX_MSG_SIZE to netif->mtu, check if netif->mtu
    is big enough in dhcp_start
```

```
2009-04-15 Simon Goldschmidt
* netbuf.c: bug #26027: netbuf_chain resulted in pbuf memory leak

2009-04-15 Simon Goldschmidt
* sockets.c, ppp.c: bug #25763: corrected 4 occurrences of SMEMCPY to MEMCPY

2009-04-15 Simon Goldschmidt
* sockets.c: bug #26121: set_errno can be overridden

2009-04-09 Kieran Mansley (patch from Luca Ceresoli <lucaceresoli>)
* init.c, opt.h: Patch#6774 TCP_QUEUE_OOSEQ breaks compilation when
  LWIP_TCP==0

2009-04-09 Kieran Mansley (patch from Roy Lee <roylee17>)
* tcp.h: Patch#6802 Add do-while-clauses to those function like
  macros in tcp.h

2009-03-31 Kieran Mansley
* tcp.c, tcp_in.c, tcp_out.c, tcp.h, opt.h: Rework the way window
  updates are calculated and sent (BUG20515)

* tcp_in.c: cope with SYN packets received during established states,
  and retransmission of initial SYN.

* tcp_out.c: set push bit correctly when tcp segments are merged

2009-03-27 Kieran Mansley
* tcp_out.c set window correctly on probes (correcting change made
  yesterday)

2009-03-26 Kieran Mansley
* tcp.c, tcp_in.c, tcp.h: add tcp_abandon() to cope with dropping
  connections where no reset required (bug #25622)

* tcp_out.c: set TCP_ACK flag on keepalive and zero window probes
  (bug #20779)

2009-02-18 Simon Goldschmidt (Jonathan Larmour and Bill Auerbach)
* ip_frag.c: patch #6528: the buffer used for IP_FRAG_USES_STATIC_BUF could be
  too small depending on MEM_ALIGNMENT

2009-02-16 Simon Goldschmidt
* sockets.h/.c, api_*.h/.c: fixed arguments of socket functions to match the ↩
   standard;
  converted size argument of netconn_write to 'size_t'

2009-02-16 Simon Goldschmidt
* tcp.h, tcp.c: fixed bug #24440: TCP connection close problem on 64-bit host
  by moving accept callback function pointer to TCP_PCB_COMMON

2009-02-12 Simon Goldschmidt
* dhcp.c: fixed bug #25345 (DHCPDECLINE is sent with "Maximum message size"
  option)

2009-02-11 Simon Goldschmidt
* dhcp.c: fixed bug #24480 (releasing old udp_pdb and pbuf in dhcp_start)
```

```
2009-02-11 Simon Goldschmidt
* opt.h, api_msg.c: added configurable default valud for netconn->recv_bufsize:
  RECV_BUFSIZE_DEFAULT (fixes bug #23726: pbuf pool exhaustion on slow recv())

2009-02-10 Simon Goldschmidt
* tcp.c: fixed bug #25467: Listen backlog is not reset on timeout in SYN_RCVD:
  Accepts_pending is decrease on a corresponding listen pcb when a connection
  in state SYN_RCVD is close.

2009-01-28 Jonathan Larmour
* pbuf.c: reclaim pbufs from TCP out-of-sequence segments if we run
  out of pool pbufs.

2008-12-19 Simon Goldschmidt
* many files: patch #6699: fixed some warnings on platform where sizeof(int) == ↩
   2

2008-12-10 Tamas Somogyi, Frédéric Bernon
* sockets.c: fixed bug #25051: lwip_recvfrom problem with udp: fromaddr and
  port uses deleted netbuf.

2008-10-18 Simon Goldschmidt
* tcp_in.c: fixed bug ##24596: Vulnerability on faulty TCP options length
  in tcp_parseopt

2008-10-15 Simon Goldschmidt
* ip_frag.c: fixed bug #24517: IP reassembly crashes on unaligned IP headers
  by packing the struct ip_reass_helper.

2008-10-03 David Woodhouse, Jonathan Larmour
* etharp.c (etharp_arp_input): Fix type aliasing problem copying ip address.

2008-10-02 Jonathan Larmour
* dns.c: Hard-code structure sizes, to avoid issues on some compilers where
  padding is included.

2008-09-30 Jonathan Larmour
* sockets.c (lwip_accept): check addr isn't NULL. If it's valid, do an
  assertion check that addrlen isn't NULL.

2008-09-30 Jonathan Larmour
* tcp.c: Fix bug #24227, wrong error message in tcp_bind.

2008-08-26 Simon Goldschmidt
* inet.h, ip_addr.h: fixed bug #24132: Cross-dependency between ip_addr.h and
  inet.h -> moved declaration of struct in_addr from ip_addr.h to inet.h

2008-08-14 Simon Goldschmidt
* api_msg.c: fixed bug #23847: do_close_internal references freed memory (when
  tcp_close returns != ERR_OK)

2008-07-08 Frédéric Bernon
* stats.h: Fix some build bugs introduced with patch #6483 (missing some ↩
   parameters
  in macros, mainly if MEM_STATS=0 and MEMP_STATS=0).

2008-06-24 Jonathan Larmour
```

```
  * tcp_in.c: Fix for bug #23693 as suggested by Art R. Ensure cseg is unused
    if tcp_seg_copy fails.

  2008-06-17 Simon Goldschmidt
  * inet_chksum.c: Checked in some ideas of patch #6460 (loop optimizations)
    and created defines for swapping bytes and folding u32 to u16.

  2008-05-30 Kieran Mansley
  * tcp_in.c Remove redundant "if" statement, and use real rcv_wnd
    rather than rcv_ann_wnd when deciding if packets are in-window.
    Contributed by <arasmussen@consultant.datasys.swri.edu>

  2008-05-30 Kieran Mansley
  * mem.h: Fix BUG#23254.  Change macro definition of mem_* to allow
    passing as function pointers when MEM_LIBC_MALLOC is defined.

  2008-05-09 Jonathan Larmour
  * err.h, err.c, sockets.c: Fix bug #23119: Reorder timeout error code to
    stop it being treated as a fatal error.

  2008-04-15 Simon Goldschmidt
  * dhcp.c: fixed bug #22804: dhcp_stop doesn't clear NETIF_FLAG_DHCP
    (flag now cleared)

  2008-03-27 Simon Goldschmidt
  * mem.c, tcpip.c, tcpip.h, opt.h: fixed bug #21433 (Calling mem_free/pbuf_free
    from interrupt context isn't safe): set LWIP_USE_HEAP_FROM_INTERRUPT to 1
    in lwipopts.h or use pbuf_free_callback(p)/mem_free_callback(m) to free pbufs
    or heap memory from interrupt context

  2008-03-26 Simon Goldschmidt
  * tcp_in.c, tcp.c: fixed bug #22249: division by zero could occur if a remote
    host sent a zero mss as TCP option.


(STABLE-1.3.0)

  ++ New features:

  2008-03-10 Jonathan Larmour
  * inet_chksum.c: Allow choice of one of the sample algorithms to be
    made from lwipopts.h. Fix comment on how to override LWIP_CHKSUM.

  2008-01-22 Frédéric Bernon
  * tcp.c, tcp_in.c, tcp.h, opt.h: Rename LWIP_CALCULATE_EFF_SEND_MSS in
    TCP_CALCULATE_EFF_SEND_MSS to have coherent TCP options names.

  2008-01-14 Frédéric Bernon
  * rawapi.txt, api_msg.c, tcp.c, tcp_in.c, tcp.h: changes for task #7675 "Enable
    to refuse data on a TCP_EVENT_RECV call". Important, behavior changes for the
    tcp_recv callback (see rawapi.txt).

  2008-01-14 Frédéric Bernon, Marc Chaland
  * ip.c: Integrate patch #6369" ip_input : checking before realloc".

  2008-01-12 Frédéric Bernon
  * tcpip.h, tcpip.c, api.h, api_lib.c, api_msg.c, sockets.c: replace the field
    netconn::sem per netconn::op_completed like suggested for the task #7490
    "Add return value to sys_mbox_post".

  2008-01-12 Frédéric Bernon
  * api_msg.c, opt.h: replace DEFAULT_RECVMBOX_SIZE per DEFAULT_TCP_RECVMBOX_SIZE,
    DEFAULT_UDP_RECVMBOX_SIZE and DEFAULT_RAW_RECVMBOX_SIZE (to optimize queues
    sizes), like suggested for the task #7490 "Add return value to sys_mbox_post".

  2008-01-10 Frédéric Bernon
  * tcpip.h, tcpip.c: add tcpip_callback_with_block function for the task #7490
    "Add return value to sys_mbox_post". tcpip_callback is always defined as
    "blocking" ("block" parameter = 1).

  2008-01-10 Frédéric Bernon
  * tcpip.h, tcpip.c, api.h, api_lib.c, api_msg.c, sockets.c: replace the field
    netconn::mbox (sys_mbox_t) per netconn::sem (sys_sem_t) for the task #7490
    "Add return value to sys_mbox_post".

  2008-01-05 Frédéric Bernon
  * sys_arch.txt, api.h, api_lib.c, api_msg.h, api_msg.c, tcpip.c, sys.h, opt.h:
    Introduce changes for task #7490 "Add return value to sys_mbox_post" with some
    modifications in the sys_mbox api: sys_mbox_new take a "size" parameters which
    indicate the number of pointers query by the mailbox. There is three defines
    in opt.h to indicate sizes for tcpip::mbox, netconn::recvmbox, and for the
    netconn::acceptmbox. Port maintainers, you can decide to just add this new
    parameter in your implementation, but to ignore it to keep the previous ↩
       behavior.
    The new sys_mbox_trypost function return a value to know if the mailbox is
    full or if the message is posted. Take a look to sys_arch.txt for more details ↩
       .
    This new function is used in tcpip_input (so, can be called in an interrupt
    context since the function is not blocking), and in recv_udp and recv_raw.

  2008-01-04 Frédéric Bernon, Simon Goldschmidt, Jonathan Larmour
  * rawapi.txt, api.h, api_lib.c, api_msg.h, api_msg.c, sockets.c, tcp.h, tcp.c,
    tcp_in.c, init.c, opt.h: rename backlog options with TCP_ prefix, limit the
    "backlog" parameter in an u8_t, 0 is interpreted as "smallest queue", add
    documentation in the rawapi.txt file.

  2007-12-31 Kieran Mansley (based on patch from Per-Henrik Lundbolm)
  * tcp.c, tcp_in.c, tcp_out.c, tcp.h: Add TCP persist timer

  2007-12-31 Frédéric Bernon, Luca Ceresoli
  * autoip.c, etharp.c: ip_addr.h: Integrate patch #6348: "Broadcast ARP packets
    in autoip". The change in etharp_raw could be removed, since all calls to
    etharp_raw use ethbroadcast for the "ethdst_addr" parameter. But it could be
    wrong in the future.

  2007-12-30 Frédéric Bernon, Tom Evans
  * ip.c: Fix bug #21846 "LwIP doesn't appear to perform any IP Source Address
    Filtering" reported by Tom Evans.

  2007-12-21 Frédéric Bernon, Simon Goldschmidt, Jonathan Larmour
  * tcp.h, opt.h, api.h, api_msg.h, tcp.c, tcp_in.c, api_lib.c, api_msg.c,
    sockets.c, init.c: task #7252: Implement TCP listen backlog: Warning: raw API
    applications have to call 'tcp_accepted(pcb)' in their accept callback to
    keep accepting new connections.
```

```
2007-12-13 Frédéric Bernon
* api_msg.c, err.h, err.c, sockets.c, dns.c, dns.h: replace "enum dns_result"
  by err_t type. Add a new err_t code "ERR_INPROGRESS".

2007-12-12 Frédéric Bernon
* dns.h, dns.c, opt.h: move DNS options to the "right" place. Most visibles
  are the one which have ram usage.

2007-12-05 Frédéric Bernon
* netdb.c: add a LWIP_DNS_API_HOSTENT_STORAGE option to decide to use a static
  set of variables (=0) or a local one (=1). In this last case, your port should
  provide a function "struct hostent* sys_thread_hostent( struct hostent* h)"
  which have to do a copy of "h" and return a pointer ont the "per-thread" copy.

2007-12-03 Simon Goldschmidt
* ip.c: ip_input: check if a packet is for inp first before checking all other
  netifs on netif_list (speeds up packet receiving in most cases)

2007-11-30 Simon Goldschmidt
* udp.c, raw.c: task #7497: Sort lists (pcb, netif, ...) for faster access
  UDP: move a (connected) pcb selected for input to the front of the list of
  pcbs so that it is found faster next time. Same for RAW pcbs that have eaten
  a packet.

2007-11-28 Simon Goldschmidt
* etharp.c, stats.c, stats.h, opt.h: Introduced ETHARP_STATS

2007-11-25 Simon Goldschmidt
* dhcp.c: dhcp_unfold_reply() uses pbuf_copy_partial instead of its own copy
  algorithm.

2007-11-24 Simon Goldschmidt
* netdb.h, netdb.c, sockets.h/.c: Moved lwip_gethostbyname from sockets.c
  to the new file netdb.c; included lwip_getaddrinfo.

2007-11-21 Simon Goldschmidt
* tcp.h, opt.h, tcp.c, tcp_in.c: implemented calculating the effective send-mss
  based on the MTU of the netif used to send. Enabled by default. Disable by
  setting LWIP_CALCULATE_EFF_SEND_MSS to 0. This fixes bug #21492.

2007-11-19 Frédéric Bernon
* api_msg.c, dns.h, dns.c: Implement DNS_DOES_NAME_CHECK option (check if name
  received match the name query), implement DNS_USES_STATIC_BUF (the place where
  copy dns payload to parse the response), return an error if there is no place
  for a new query, and fix some minor problems.

2007-11-16 Simon Goldschmidt
* new files: ipv4/inet.c, ipv4/inet_chksum.c, ipv6/inet6.c
  removed files: core/inet.c, core/inet6.c
  Moved inet files into ipv4/ipv6 directory; splitted inet.c/inet.h into
  inet and chksum part; changed includes in all lwIP files as appropriate

2007-11-16 Simon Goldschmidt
* api.h, api_msg.h, api_lib.c, api_msg.c, socket.h, socket.c: Added sequential
  dns resolver function for netconn api (netconn_gethostbyname) and socket api
  (gethostbyname/gethostbyname_r).
```

```
2007-11-15 Jim Pettinato, Frédéric Bernon
* opt.h, init.c, tcpip.c, dhcp.c, dns.h, dns.c: add DNS client for simple name
  requests with RAW api interface. Initialization is done in lwip_init() with
  build time options. DNS timer is added in tcpip_thread context. DHCP can set
  DNS server ip addresses when options are received. You need to set LWIP_DNS=1
  in your lwipopts.h file (LWIP_DNS=0 in opt.h). DNS_DEBUG can be set to get
  some traces with LWIP_DEBUGF. Sanity check have been added. There is a "todo"
  list with points to improve.

2007-11-06 Simon Goldschmidt
* opt.h, mib2.c: Patch #6215: added ifAdminStatus write support (if explicitly
  enabled by defining SNMP_SAFE_REQUESTS to 0); added code to check link status
  for ifOperStatus if LWIP_NETIF_LINK_CALLBACK is defined.

2007-11-06 Simon Goldschmidt
* api.h, api_msg.h and dependent files: Task #7410: Removed the need to include
  core header files in api.h (ip/tcp/udp/raw.h) to hide the internal
  implementation from netconn api applications.

2007-11-03 Frédéric Bernon
* api.h, api_lib.c, api_msg.c, sockets.c, opt.h: add SO_RCVBUF option for UDP &
  RAW netconn. You need to set LWIP_SO_RCVBUF=1 in your lwipopts.h (it’s  ↩
     disabled
  by default). Netconn API users can use the netconn_recv_bufsize macro to  ↩
     access
  it. This is a first release which have to be improve for TCP. Note it used the
  netconn::recv_avail which need to be more "thread-safe" (note there is already
  the problem for FIONREAD with lwip_ioctl/ioctlsocket).

2007-11-01 Frédéric Bernon, Marc Chaland
* sockets.h, sockets.c, api.h, api_lib.c, api_msg.h, api_msg.c, tcp.h, tcp_out.c ↩
   :
  Integrate "patch #6250 : MSG_MORE flag for send". MSG_MORE is used at socket  ↩
     api
  layer, NETCONN_MORE at netconn api layer, and TCP_WRITE_FLAG_MORE at raw api
  layer. This option enable to delayed TCP PUSH flag on multiple "write" calls.
  Note that previous "copy" parameter for "write" APIs is now called "apiflags".

2007-10-24 Frédéric Bernon
* api.h, api_lib.c, api_msg.c: Add macro API_EVENT in the same spirit than
  TCP_EVENT_xxx macros to get a code more readable. It could also help to remove
  some code (like we have talk in "patch #5919 : Create compile switch to remove
  select code"), but it could be done later.

2007-10-08 Simon Goldschmidt
* many files: Changed initialization: many init functions are not needed any
  more since we now rely on the compiler initializing global and static
  variables to zero!

2007-10-06 Simon Goldschmidt
* ip_frag.c, memp.c, mib2.c, ip_frag.h, memp_std.h, opt.h: Changed IP_REASSEMBLY
  to enqueue the received pbufs so that multiple packets can be reassembled
  simultaneously and no static reassembly buffer is needed.

2007-10-05 Simon Goldschmidt
* tcpip.c, etharp.h, etharp.c: moved ethernet_input from tcpip.c to etharp.c so
```

```
  all netifs (or ports) can use it.

2007-10-05 Frédéric Bernon
* netifapi.h, netifapi.c: add function netifapi_netif_set_default. Change the
  common function to reduce a little bit the footprint (for all functions using
  only the "netif" parameter).

2007-10-03 Frédéric Bernon
* netifapi.h, netifapi.c: add functions netifapi_netif_set_up,  ↩
   netifapi_netif_set_down,
  netifapi_autoip_start and netifapi_autoip_stop. Use a common function to  ↩
     reduce
  a little bit the footprint (for all functions using only the "netif" parameter ↩
     ).

2007-09-15 Frédéric Bernon
* udp.h, udp.c, sockets.c: Changes for "#20503 IGMP Improvement". Add  ↩
   IP_MULTICAST_IF
  option in socket API, and a new field "multicast_ip" in "struct udp_pcb" (for
  netconn and raw API users), only if LWIP_IGMP=1. Add getsockopt processing for
  IP_MULTICAST_TTL and IP_MULTICAST_IF.

2007-09-10 Frédéric Bernon
* snmp.h, mib2.c: enable to remove SNMP timer (which consumne several cycles
  even when it's not necessary). snmp_agent.txt tell to call snmp_inc_sysuptime ↩
     ()
  each 10ms (but, it's intrusive if you use sys_timeout feature). Now, you can
  decide to call snmp_add_sysuptime(100) each 1000ms (which is bigger "step",  ↩
     but
  call to a lower frequency). Or, you can decide to not call snmp_inc_sysuptime ↩
     ()
  or snmp_add_sysuptime(), and to define the SNMP_GET_SYSUPTIME(sysuptime) macro ↩
     .
  This one is undefined by default in mib2.c. SNMP_GET_SYSUPTIME is called  ↩
     inside
  snmp_get_sysuptime(u32_t *value), and enable to change "sysuptime" value only
  when it's queried (any direct call to "sysuptime" is changed by a call to
  snmp_get_sysuptime).

2007-09-09 Frédéric Bernon, Bill Florac
* igmp.h, igmp.c, netif.h, netif.c, ip.c: To enable to have interfaces with IGMP ↩
   ,
  and others without it, there is a new NETIF_FLAG_IGMP flag to set in netif-> ↩
     flags
  if you want IGMP on an interface. igmp_stop() is now called inside  ↩
     netif_remove().
  igmp_report_groups() is now called inside netif_set_link_up() (need to have
  LWIP_NETIF_LINK_CALLBACK=1) to resend reports once the link is up (avoid to  ↩
     wait
  the next query message to receive the matching multicast streams).

2007-09-08 Frédéric Bernon
* sockets.c, ip.h, api.h, tcp.h: declare a "struct ip_pcb" which only contains
  IP_PCB. Add in the netconn's "pcb" union a "struct ip_pcb *ip;" (no size  ↩
     change).
  Use this new field to access to common pcb fields (ttl, tos, so_options, etc ↩
     ...).
  Enable to access to these fields with LWIP_TCP=0.

2007-09-05 Frédéric Bernon
* udp.c, ipv4/icmp.c, ipv4/ip.c, ipv6/icmp.c, ipv6/ip6.c, ipv4/icmp.h,
  ipv6/icmp.h, opt.h: Integrate "task #7272 : LWIP_ICMP option". The new option
  LWIP_ICMP enable/disable ICMP module inside the IP stack (enable per default).
  Be careful, disabling ICMP make your product non-compliant to RFC1122, but
  help to reduce footprint, and to reduce "visibility" on the Internet.

2007-09-05 Frédéric Bernon, Bill Florac
* opt.h, sys.h, tcpip.c, slipif.c, ppp.c, sys_arch.txt: Change parameters list
  for sys_thread_new (see "task #7252 : Create sys_thread_new_ex()"). Two new
  parameters have to be provided: a task name, and a task stack size. For this
  one, since it's platform dependant, you could define the best one for you in
  your lwipopts.h. For port maintainers, you can just add these new parameters
  in your sys_arch.c file, and but it's not mandatory, use them in your OS
  specific functions.

2007-09-05 Frédéric Bernon
* inet.c, autoip.c, msg_in.c, msg_out.c, init.c: Move some build time checkings
  inside init.c for task #7142 "Sanity check user-configurable values".

2007-09-04 Frédéric Bernon, Bill Florac
* igmp.h, igmp.c, memp_std.h, memp.c, init.c, opt.h: Replace mem_malloc call by
  memp_malloc, and use a new MEMP_NUM_IGMP_GROUP option (see opt.h to define the
  value). It will avoid potential fragmentation problems, use a counter to know
  how many times a group is used on an netif, and free it when all applications
  leave it. MEMP_NUM_IGMP_GROUP got 8 as default value (and init.c got a sanity
  check if LWIP_IGMP!=0).

2007-09-03 Frédéric Bernon
* igmp.h, igmp.c, sockets.c, api_msg.c: Changes for "#20503 IGMP Improvement".
  Initialize igmp_mac_filter to NULL in netif_add (this field should be set in
  the netif's "init" function). Use the "imr_interface" field (for socket layer)
  and/or the "interface" field (for netconn layer), for join/leave operations.
  The igmp_join/leavegroup first parameter change from a netif to an ipaddr.
  This field could be a netif's ipaddr, or "any" (same meaning than ↩
      ip_addr_isany).

2007-08-30 Frédéric Bernon
* Add netbuf.h, netbuf.c, Change api.h, api_lib.c: #7249 "Split netbuf functions
  from api/api_lib". Now netbuf API is independant of netconn, and can be used
  with other API (application based on raw API, or future "socket2" API). Ports
  maintainers just have to add src/api/netbuf.c in their makefile/projects.

2007-08-30 Frédéric Bernon, Jonathan Larmour
* init.c: Add first version of lwip_sanity_check for task #7142 "Sanity check
  user-configurable values".

2007-08-29 Frédéric Bernon
* igmp.h, igmp.c, tcpip.c, init.c, netif.c: change igmp_init and add igmp_start.
  igmp_start is call inside netif_add. Now, igmp initialization is in the same
  spirit than the others modules. Modify some IGMP debug traces.

2007-08-29 Frédéric Bernon
* Add init.h, init.c, Change opt.h, tcpip.c: Task  #7213 "Add a lwip_init ↩
   function"
  Add lwip_init function to regroup all modules initializations, and to provide
  a place to add code for task #7142 "Sanity check user-configurable values".
  Ports maintainers should remove direct initializations calls from their code,
  and add init.c in their makefiles. Note that lwip_init() function is called
  inside tcpip_init, but can also be used by raw api users since all calls are
  disabled when matching options are disabled. Also note that their is new ↩
     options
  in opt.h, you should configure in your lwipopts.h (they are enabled per ↩
     default).

2007-08-26 Marc Boucher
* api_msg.c: do_close_internal(): Reset the callbacks and arg (conn) to NULL
  since they can under certain circumstances be called with an invalid conn
  pointer after the connection has been closed (and conn has been freed).

2007-08-25 Frédéric Bernon (Artem Migaev’s Patch)
* netif.h, netif.c: Integrate "patch #6163 : Function to check if link layer is ↩
   up".
  Add a netif_is_link_up() function if LWIP_NETIF_LINK_CALLBACK option is set.

2007-08-22 Frédéric Bernon
* netif.h, netif.c, opt.h: Rename LWIP_NETIF_CALLBACK in ↩
   LWIP_NETIF_STATUS_CALLBACK
  to be coherent with new LWIP_NETIF_LINK_CALLBACK option before next release.

2007-08-22 Frédéric Bernon
* tcpip.h, tcpip.c, ethernetif.c, opt.h: remove options ETHARP_TCPIP_INPUT &
  ETHARP_TCPIP_ETHINPUT, now, only "ethinput" code is supported, even if the
  name is tcpip_input (we keep the name of 1.2.0 function).

2007-08-17 Jared Grubb
* memp_std.h, memp.h, memp.c, mem.c, stats.c: (Task #7136) Centralize mempool
  settings into new memp_std.h and optional user file lwippools.h. This adds
  more dynamic mempools, and allows the user to create an arbitrary number of
  mempools for mem_malloc.

2007-08-16 Marc Boucher
* api_msg.c: Initialize newconn->state to NETCONN_NONE in accept_function;
  otherwise it was left to NETCONN_CLOSE and sent_tcp() could prematurely
  close the connection.

2007-08-16 Marc Boucher
* sockets.c: lwip_accept(): check netconn_peer() error return.

2007-08-16 Marc Boucher
* mem.c, mem.h: Added mem_calloc().

2007-08-16 Marc Boucher
* tcpip.c, tcpip.h memp.c, memp.h: Added distinct memp (MEMP_TCPIP_MSG_INPKT)
  for input packets to prevent floods from consuming all of MEMP_TCPIP_MSG
  and starving other message types.
  Renamed MEMP_TCPIP_MSG to MEMP_TCPIP_MSG_API

2007-08-16 Marc Boucher
* pbuf.c, pbuf.h, etharp.c, tcp_in.c, sockets.c: Split pbuf flags in pbuf
  type and flgs (later renamed to flags).
  Use enum pbuf_flag as pbuf_type.  Renumber PBUF_FLAG_*.
```

```
    Improved lwip_recvfrom().  TCP push now propagated.

  2007-08-16 Marc Boucher
  * ethernetif.c, contrib/ports/various: ethbroadcast now a shared global
    provided by etharp.

  2007-08-16 Marc Boucher
  * ppp_oe.c ppp_oe.h, auth.c chap.c fsm.c lcp.c ppp.c ppp.h,
    etharp.c ethernetif.c, etharp.h, opt.h tcpip.h, tcpip.c:
    Added PPPoE support and various PPP improvements.

  2007-07-25 Simon Goldschmidt
  * api_lib.c, ip_frag.c, pbuf.c, api.h, pbuf.h: Introduced pbuf_copy_partial,
    making netbuf_copy_partial use this function.

  2007-07-25 Simon Goldschmidt
  * tcp_in.c: Fix bug #20506: Slow start / initial congestion window starts with
    2 * mss (instead of 1 * mss previously) to comply with some newer RFCs and
    other stacks.

  2007-07-13 Jared Grubb (integrated by Frédéric Bernon)
  * opt.h, netif.h, netif.c, ethernetif.c: Add new configuration option to add
    a link callback in the netif struct, and functions to handle it. Be carefull
    for port maintainers to add the NETIF_FLAG_LINK_UP flag (like in ethernetif.c)
    if you want to be sure to be compatible with future changes...

  2007-06-30 Frédéric Bernon
  * sockets.h, sockets.c: Implement MSG_PEEK flag for recv/recvfrom functions.

  2007-06-21 Simon Goldschmidt
  * etharp.h, etharp.c: Combined etharp_request with etharp_raw for both
    LWIP_AUTOIP =0 and =1 to remove redundant code.

  2007-06-21 Simon Goldschmidt
  * mem.c, memp.c, mem.h, memp.h, opt.h: task #6863: Introduced the option
    MEM_USE_POOLS to use 4 pools with different sized elements instead of a
    heap. This both prevents memory fragmentation and gives a higher speed
    at the cost of more memory consumption. Turned off by default.

  2007-06-21 Simon Goldschmidt
  * api_lib.c, api_msg.c, api.h, api_msg.h: Converted the length argument of
    netconn_write (and therefore also api_msg_msg.msg.w.len) from u16_t into
    int to be able to send a bigger buffer than 64K with one time (mainly
    used from lwip_send).

  2007-06-21 Simon Goldschmidt
  * tcp.h, api_msg.c: Moved the nagle algorithm from netconn_write/do_write
    into a define (tcp_output_nagle) in tcp.h to provide it to raw api users, too.

  2007-06-21 Simon Goldschmidt
  * api.h, api_lib.c, api_msg.c: Fixed bug #20021: Moved sendbuf-processing in
    netconn_write from api_lib.c to api_msg.c to also prevent multiple context-
    changes on low memory or empty send-buffer.

  2007-06-18 Simon Goldschmidt
  * etharp.c, etharp.h: Changed etharp to use a defined hardware address length
    of 6 to avoid loading netif->hwaddr_len every time (since this file is only
    used for ethernet and struct eth_addr already had a defined length of 6).

  2007-06-17 Simon Goldschmidt
  * sockets.c, sockets.h: Implemented socket options SO_NO_CHECK for UDP sockets
    to disable UDP checksum generation on transmit.

  2007-06-13 Frédéric Bernon, Simon Goldschmidt
  * debug.h, api_msg.c: change LWIP_ERROR to use it to check errors like invalid
    pointers or parameters, and let the possibility to redefined it in cc.h. Use
    this macro to check "conn" parameter in api_msg.c functions.

  2007-06-11 Simon Goldschmidt
  * sockets.c, sockets.h: Added UDP lite support for sockets

  2007-06-10 Simon Goldschmidt
  * udp.h, opt.h, api_msg.c, ip.c, udp.c: Included switch LWIP_UDPLITE (enabled
    by default) to switch off UDP-Lite support if not needed (reduces udp.c code
    size)

  2007-06-09 Dominik Spies (integrated by Frédéric Bernon)
  * autoip.h, autoip.c, dhcp.h, dhcp.c, netif.h, netif.c, etharp.h, etharp.c, opt. ↩
     h:
    AutoIP implementation available for IPv4, with new options LWIP_AUTOIP and
    LWIP_DHCP_AUTOIP_COOP if you want to cooperate with DHCP. Some tips to adapt
    (see TODO mark in the source code).

  2007-06-09 Simon Goldschmidt
  * etharp.h, etharp.c, ethernetif.c: Modified order of parameters for
    etharp_output() to match netif->output so etharp_output() can be used
    directly as netif->output to save one function call.

  2007-06-08 Simon Goldschmidt
  * netif.h, ethernetif.c, slipif.c, loopif.c: Added define
    NETIF_INIT_SNMP(netif, type, speed) to initialize per-netif snmp variables,
    added initialization of those to ethernetif, slipif and loopif.

  2007-05-18 Simon Goldschmidt
  * opt.h, ip_frag.c, ip_frag.h, ip.c: Added option IP_FRAG_USES_STATIC_BUF
    (defaulting to off for now) that can be set to 0 to send fragmented
    packets by passing PBUF_REFs down the stack.

  2007-05-23 Frédéric Bernon
  * api_lib.c: Implement SO_RCVTIMEO for accept and recv on TCP
    connections, such present in patch #5959.

  2007-05-23 Frédéric Bernon
  * api.h, api_lib.c, api_msg.c, sockets.c: group the different NETCONN_UDPxxx
    code in only one part...

  2007-05-18 Simon Goldschmidt
  * opt.h, memp.h, memp.c: Added option MEMP_OVERFLOW_CHECK to check for memp
    elements to overflow. This is achieved by adding some bytes before and after
    each pool element (increasing their size, of course), filling them with a
    prominent value and checking them on freeing the element.
    Set it to 2 to also check every element in every pool each time memp_malloc()
    or memp_free() is called (slower but more helpful).
```

```
2007-05-10 Simon Goldschmidt
* opt.h, memp.h, memp.c, pbuf.c (see task #6831): use a new memp pool for
  PBUF_POOL pbufs instead of the old pool implementation in pbuf.c to reduce
  code size.

2007-05-11 Frédéric Bernon
* sockets.c, api_lib.c, api_msg.h, api_msg.c, netifapi.h, netifapi.c, tcpip.c:
  Include a function pointer instead of a table index in the message to reduce
  footprint. Disable some part of lwip_send and lwip_sendto if some options are
  not set (LWIP_TCP, LWIP_UDP, LWIP_RAW).

2007-05-10 Simon Goldschmidt
* *.h (except netif/ppp/*.h): Included patch #5448: include '#ifdef __cplusplus
  \ extern "C" {' in all header files. Now you can write your application using
  the lwIP stack in C++ and simply #include the core files. Note I have left
  out the netif/ppp/*h header files for now, since I don't know which files are
  included by applications and which are for internal use only.

2007-05-09 Simon Goldschmidt
* opt.h, *.c/*.h: Included patch #5920: Create define to override C-library
  memcpy. 2 Defines are created: MEMCPY() for normal memcpy, SMEMCPY() for
  situations where some compilers might inline the copy and save a function
  call. Also replaced all calls to memcpy() with calls to (S)MEMCPY().

2007-05-08 Simon Goldschmidt
* mem.h: If MEM_LIBC_MALLOC==1, allow the defines (e.g. mem_malloc() -> malloc() ↩
   )
  to be overriden in case the C-library malloc implementation is not protected
  against concurrent access.

2007-05-04 Simon Goldschmidt (Atte Kojo)
* etharp.c: Introduced fast one-entry-cache to speed up ARP lookup when sending
  multiple packets to the same host.

2007-05-04 Frédéric Bernon, Jonathan Larmour
* sockets.c, api.h, api_lib.c, api_msg.h, api_msg.c: Fix bug #19162 "lwip_sento: ↩
     a possible
  to corrupt remote addr/port connection state". Reduce problems "not enought  ↩
     memory" with
  netbuf (if we receive lot of datagrams). Improve lwip_sendto (only one  ↩
     exchange between
  sockets api and api_msg which run in tcpip_thread context). Add netconn_sento  ↩
     function.
  Warning, if you directly access to "fromaddr" & "fromport" field from netbuf  ↩
     struct,
  these fields are now renamed "addr" & "port".

2007-04-11 Jonathan Larmour
* sys.h, api_lib.c: Provide new sys_mbox_tryfetch function. Require ports to  ↩
    provide new
  sys_arch_mbox_tryfetch function to get a message if one is there, otherwise  ↩
     return
  with SYS_MBOX_EMPTY. sys_arch_mbox_tryfetch can be implemented as a function- ↩
     like macro
  by the port in sys_arch.h if desired.

2007-04-06 Frédéric Bernon, Simon Goldschmidt
```

```
* opt.h, tcpip.h, tcpip.c, netifapi.h, netifapi.c: New configuration option ↩
   LWIP_NETIF_API
  allow to use thread-safe functions to add/remove netif in list, and to start/ ↩
     stop dhcp
  clients, using new functions from netifapi.h. Disable as default (no port ↩
     change to do).

2007-04-05 Frédéric Bernon
* sockets.c: remplace ENOBUFS errors on alloc_socket by ENFILE to be more BSD ↩
   compliant.

2007-04-04 Simon Goldschmidt
* arch.h, api_msg.c, dhcp.c, msg_in.c, sockets.c: Introduced #define ↩
   LWIP_UNUSED_ARG(x)
  use this for and architecture-independent form to tell the compiler you ↩
     intentionally
  are not using this variable. Can be overriden in cc.h.

2007-03-28 Frédéric Bernon
* opt.h, netif.h, dhcp.h, dhcp.c: New configuration option LWIP_NETIF_HOSTNAME ↩
   allow to
  define a hostname in netif struct (this is just a pointer, so, you can use a ↩
     hardcoded
  string, point on one of your's ethernetif field, or alloc a string you will ↩
     free yourself).
  It will be used by DHCP to register a client hostname, but can also be use ↩
     when you call
  snmp_set_sysname.

2007-03-28 Frédéric Bernon
* netif.h, netif.c: A new NETIF_FLAG_ETHARP flag is defined in netif.h, to allow ↩
    to
  initialize a network interface's flag with. It tell this interface is an ↩
     ethernet
  device, and we can use ARP with it to do a "gratuitous ARP" (RFC 3220 "IP ↩
     Mobility
  Support for IPv4" section 4.6) when interface is "up" with netif_set_up().

2007-03-26 Frédéric Bernon, Jonathan Larmour
* opt.h, tcpip.c: New configuration option LWIP_ARP allow to disable ARP init at ↩
    build
  time if you only use PPP or SLIP. The default is enable. Note we don't have to ↩
      call
  etharp_init in your port's initilization sequence if you use tcpip.c, because ↩
     this call
  is done in tcpip_init function.

2007-03-22 Frédéric Bernon
* stats.h, stats.c, msg_in.c: Stats counters can be change to u32_t if necessary ↩
    with the
  new option LWIP_STATS_LARGE. If you need this option, define LWIP_STATS_LARGE ↩
     to 1 in
  your lwipopts.h. More, unused counters are not defined in the stats structs, ↩
     and not
  display by stats_display(). Note that some options (SYS_STATS and RAW_STATS) ↩
     are defined
  but never used. Fix msg_in.c with the correct #if test for a stat display.
```

```
2007-03-21 Kieran Mansley
* netif.c, netif.h: Apply patch#4197 with some changes (originator: ↩
    rireland@hmgsl.com).
  Provides callback on netif up/down state change.

2007-03-11 Frédéric Bernon, Mace Gael, Steve Reynolds
* sockets.h, sockets.c, api.h, api_lib.c, api_msg.h, api_msg.c, igmp.h, igmp.c,
  ip.c, netif.h, tcpip.c, opt.h:
  New configuration option LWIP_IGMP to enable IGMP processing. Based on only ↩
      one
  filter per all network interfaces. Declare a new function in netif to enable ↩
      to
  control the MAC filter (to reduce lwIP traffic processing).

2007-03-11 Frédéric Bernon
* tcp.h, tcp.c, sockets.c, tcp_out.c, tcp_in.c, opt.h: Keepalive values can
  be configured at run time with LWIP_TCP_KEEPALIVE, but don't change this
  unless you know what you're doing (default are RFC1122 compliant). Note
  that TCP_KEEPIDLE and TCP_KEEPINTVL have to be set in seconds.

2007-03-08 Frédéric Bernon
* tcp.h: Keepalive values can be configured at compile time, but don't change
  this unless you know what you're doing (default are RFC1122 compliant).

2007-03-08 Frédéric Bernon
* sockets.c, api.h, api_lib.c, tcpip.c, sys.h, sys.c, err.c, opt.h:
  Implement LWIP_SO_RCVTIMEO configuration option to enable/disable SO_RCVTIMEO
  on UDP sockets/netconn.

2007-03-08 Simon Goldschmidt
* snmp_msg.h, msg_in.c: SNMP UDP ports can be configured at compile time.

2007-03-06 Frédéric Bernon
* api.h, api_lib.c, sockets.h, sockets.c, tcpip.c, sys.h, sys.c, err.h:
  Implement SO_RCVTIMEO on UDP sockets/netconn.

2007-02-28 Kieran Mansley (based on patch from Simon Goldschmidt)
* api_lib.c, tcpip.c, memp.c, memp.h: make API msg structs allocated
  on the stack and remove the API msg type from memp

2007-02-26 Jonathan Larmour (based on patch from Simon Goldschmidt)
* sockets.h, sockets.c: Move socket initialization to new
  lwip_socket_init() function.
  NOTE: this changes the API with ports. Ports will have to be
  updated to call lwip_socket_init() now.

2007-02-26 Jonathan Larmour (based on patch from Simon Goldschmidt)
* api_lib.c: Use memcpy in netbuf_copy_partial.


++ Bug fixes:

2008-03-17 Frédéric Bernon, Ed Kerekes
* igmp.h, igmp.c: Fix bug #22613 "IGMP iphdr problem" (could have
  some problems to fill the IP header on some targets, use now the
  ip.h macros to do it).
```

```
2008-03-13 Frédéric Bernon
* sockets.c: Fix bug #22435 "lwip_recvfrom with TCP break;". Using
  (lwip_)recvfrom with valid "from" and "fromlen" parameters, on a
  TCP connection caused a crash. Note that using (lwip_)recvfrom
  like this is a bit slow and that using (lwip)getpeername is the
  good lwip way to do it (so, using recv is faster on tcp sockets).

2008-03-12 Frédéric Bernon, Jonathan Larmour
* api_msg.c, contrib/apps/ping.c: Fix bug #22530 "api_msg.c's
  recv_raw() does not consume data", and the ping sample (with
  LWIP_SOCKET=1, the code did the wrong supposition that lwip_recvfrom
  returned the IP payload, without the IP header).

2008-03-04 Jonathan Larmour
* mem.c, stats.c, mem.h: apply patch #6414 to avoid compiler errors
and/or warnings on some systems where mem_size_t and size_t differ.
* pbuf.c, ppp.c: Fix warnings on some systems with mem_malloc.

2008-03-04 Kieran Mansley (contributions by others)
* Numerous small compiler error/warning fixes from contributions to
  mailing list after 1.3.0 release candidate made.

2008-01-25 Cui hengbin (integrated by Frédéric Bernon)
* dns.c: Fix bug #22108 "DNS problem" caused by unaligned structures.

2008-01-15 Kieran Mansley
* tcp_out.c: BUG20511.  Modify persist timer to start when we are
  prevented from sending by a small send window, not just a zero
  send window.

2008-01-09 Jonathan Larmour
* opt.h, ip.c: Rename IP_OPTIONS define to IP_OPTIONS_ALLOWED to avoid
  conflict with Linux system headers.

2008-01-06 Jonathan Larmour
* dhcp.c: fix bug #19927: "DHCP NACK problem" by clearing any existing set IP
  address entirely on receiving a DHCPNAK, and restarting discovery.

2007-12-21 Simon Goldschmidt
* sys.h, api_lib.c, api_msg.c, sockets.c: fix bug #21698: "netconn->recv_avail
  is not protected" by using new macros for interlocked access to modify/test
  netconn->recv_avail.

2007-12-20 Kieran Mansley (based on patch from Oleg Tyshev)
* tcp_in.c: fix bug# 21535 (nrtx not reset correctly in SYN_SENT state)

2007-12-20 Kieran Mansley (based on patch from Per-Henrik Lundbolm)
* tcp.c, tcp_in.c, tcp_out.c, tcp.h: fix bug #20199 (better handling
  of silly window avoidance and prevent lwIP from shrinking the window)

2007-12-04 Simon Goldschmidt
* tcp.c, tcp_in.c: fix bug #21699 (segment leak in ooseq processing when last
  data packet was lost): add assert that all segment lists are empty in
  tcp_pcb_remove before setting pcb to CLOSED state; don't directly set CLOSED
  state from LAST_ACK in tcp_process
```

```
2007-12-02 Simon Goldschmidt
* sockets.h: fix bug #21654: exclude definition of struct timeval from #ifndef ↩
   FD_SET
  If including <sys/time.h> for system-struct timeval, LWIP_TIMEVAL_PRIVATE now
  has to be set to 0 in lwipopts.h

2007-12-02 Simon Goldschmidt
* api_msg.c, api_lib.c: fix bug #21656 (recvmbox problem in netconn API): always
  allocate a recvmbox in netconn_new_with_proto_and_callback. For a tcp-listen
  netconn, this recvmbox is later freed and a new mbox is allocated for ↩
     acceptmbox.
  This is a fix for thread-safety and allocates all items needed for a netconn
  when the netconn is created.

2007-11-30 Simon Goldschmidt
* udp.c: first attempt to fix bug #21655 (DHCP doesn't work reliably with ↩
   multiple
  netifs): if LWIP_DHCP is enabled, UDP packets to DHCP_CLIENT_PORT are passed
  to netif->dhcp->pcb only (if that exists) and not to any other pcb for the ↩
     same
  port (only solution to let UDP pcbs 'bind' to a netif instead of an IP address ↩
     )

2007-11-27 Simon Goldschmidt
* ip.c: fixed bug #21643 (udp_send/raw_send don't fail if netif is down) by
  letting ip_route only use netifs that are up.

2007-11-27 Simon Goldschmidt
* err.h, api_lib.c, api_msg.c, sockets.c: Changed error handling: ERR_MEM, ↩
   ERR_BUF
  and ERR_RTE are seen as non-fatal, all other errors are fatal. netconns and
  sockets block most operations once they have seen a fatal error.

2007-11-27 Simon Goldschmidt
* udp.h, udp.c, dhcp.c: Implemented new function udp_sendto_if which takes the
  netif to send as an argument (to be able to send on netifs that are down).

2007-11-26 Simon Goldschmidt
* tcp_in.c: Fixed bug #21582: pcb->acked accounting can be wrong when ACKs
  arrive out-of-order

2007-11-21 Simon Goldschmidt
* tcp.h, tcp_out.c, api_msg.c: Fixed bug #20287: tcp_output_nagle sends too ↩
   early
  Fixed the nagle algorithm; nagle now also works for all raw API applications
  and has to be explicitly disabled with 'tcp_pcb->flags |= TF_NODELAY'

2007-11-12 Frédéric Bernon
* sockets.c, api.h, api_lib.c, api_msg.h, api_msg.c: Fixed bug #20900. Now, most
  of the netconn_peer and netconn_addr processing is done inside tcpip_thread
  context in do_getaddr.

2007-11-10 Simon Goldschmidt
* etharp.c: Fixed bug: assert fired when MEMP_ARP_QUEUE was empty (which can
  happen any time). Now the packet simply isn't enqueued when out of memory.

2007-11-01 Simon Goldschmidt
```

```
* tcp.c, tcp_in.c: Fixed bug #21494: The send mss (pcb->mss) is set to 536 (or
  TCP_MSS if that is smaller) as long as no MSS option is received from the
  remote host.

2007-11-01 Simon Goldschmidt
* tcp.h, tcp.c, tcp_in.c: Fixed bug #21491: The MSS option sent (with SYN)
  is now based on TCP_MSS instead of pcb->mss (on passive open now effectively
  sending our configured TCP_MSS instead of the one received).

2007-11-01 Simon Goldschmidt
* tcp_in.c: Fixed bug #21181: On active open, the initial congestion window was
  calculated based on the configured TCP_MSS, not on the MSS option received
  with SYN+ACK.

2007-10-09 Simon Goldschmidt
* udp.c, inet.c, inet.h: Fixed UDPLite: send: Checksum was always generated too
  short and also was generated wrong if checksum coverage != tot_len;
  receive: checksum was calculated wrong if checksum coverage != tot_len

2007-10-08 Simon Goldschmidt
* mem.c: lfree was not updated in mem_realloc!

2007-10-07 Frédéric Bernon
* sockets.c, api.h, api_lib.c: First step to fix "bug #20900 : Potential
  crash error problem with netconn_peer & netconn_addr". VERY IMPORTANT:
  this change cause an API breakage for netconn_addr, since a parameter
  type change. Any compiler should cause an error without any changes in
  yours netconn_peer calls (so, it can’t be a "silent change"). It also
  reduce a little bit the footprint for socket layer (lwip_getpeername &
  lwip_getsockname use now a common lwip_getaddrname function since
  netconn_peer & netconn_addr have the same parameters).

2007-09-20 Simon Goldschmidt
* tcp.c: Fixed bug #21080 (tcp_bind without check pcbs in TIME_WAIT state)
  by checking  tcp_tw_pcbs also

2007-09-19 Simon Goldschmidt
* icmp.c: Fixed bug #21107 (didn’t reset IP TTL in ICMP echo replies)

2007-09-15 Mike Kleshov
* mem.c: Fixed bug #21077 (inaccuracy in calculation of lwip_stat.mem.used)

2007-09-06 Frédéric Bernon
* several-files: replace some #include "arch/cc.h" by "lwip/arch.h", or simply  ↩
   remove
  it as long as "lwip/opt.h" is included before (this one include "lwip/debug.h" ↩
      which
  already include "lwip/arch.h"). Like that, default defines are provided by " ↩
     lwip/arch.h"
  if they are not defined in cc.h, in the same spirit than "lwip/opt.h" for  ↩
     lwipopts.h.

2007-08-30 Frédéric Bernon
* igmp.h, igmp.c: Some changes to remove some redundant code, add some traces,
  and fix some coding style.

2007-08-28 Frédéric Bernon
```

```
* tcpip.c: Fix TCPIP_MSG_INPKT processing: now, tcpip_input can be used for any
  kind of packets. These packets are considered like Ethernet packets (payload
  pointing to ethhdr) if the netif got the NETIF_FLAG_ETHARP flag. Else, packets
  are considered like IP packets (payload pointing to iphdr).

2007-08-27 Frédéric Bernon
* api.h, api_lib.c, api_msg.c: First fix for "bug #20900 : Potential crash error
  problem with netconn_peer & netconn_addr". Introduce NETCONN_LISTEN  ↩
      netconn_state
  and remove obsolete ones (NETCONN_RECV & NETCONN_ACCEPT).

2007-08-24 Kieran Mansley
* inet.c Modify (acc >> 16) test to ((acc >> 16) != 0) to help buggy
  compiler (Paradigm C++)

2007-08-09 Frédéric Bernon, Bill Florac
* stats.h, stats.c, igmp.h, igmp.c, opt.h: Fix for bug #20503 : IGMP Improvement ↩
   .
  Introduce IGMP_STATS to centralize statistics management.

2007-08-09 Frédéric Bernon, Bill Florac
* udp.c: Fix for bug #20503 : IGMP Improvement. Enable to receive a multicast
  packet on a udp pcb binded on an netif's IP address, and not on "any".

2007-08-09 Frédéric Bernon, Bill Florac
* igmp.h, igmp.c, ip.c: Fix minor changes from bug #20503 : IGMP Improvement.
  This is mainly on using lookup/lookfor, and some coding styles...

2007-07-26 Frédéric Bernon (and "thedoctor")
* igmp.c: Fix bug #20595 to accept IGMPv3 "Query" messages.

2007-07-25 Simon Goldschmidt
* api_msg.c, tcp.c: Another fix for bug #20021: by not returning an error if
  tcp_output fails in tcp_close, the code in do_close_internal gets simpler
  (tcp_output is called again later from tcp timers).

2007-07-25 Simon Goldschmidt
* ip_frag.c: Fixed bug #20429: use the new pbuf_copy_partial instead of the old
  copy_from_pbuf, which illegally modified the given pbuf.

2007-07-25 Simon Goldschmidt
* tcp_out.c: tcp_enqueue: pcb->snd_queuelen didn't work for chaine PBUF_RAMs:
  changed snd_queuelen++ to snd_queuelen += pbuf_clen(p).

2007-07-24 Simon Goldschmidt
* api_msg.c, tcp.c: Fix bug #20480: Check the pcb passed to tcp_listen() for the
  correct state (must be CLOSED).

2007-07-13 Thomas Taranowski (commited by Jared Grubb)
* memp.c: Fix bug #20478: memp_malloc returned NULL+MEMP_SIZE on failed
  allocation. It now returns NULL.

2007-07-13 Frédéric Bernon
* api_msg.c: Fix bug #20318: api_msg "recv" callbacks don't call pbuf_free in
  all error cases.

2007-07-13 Frédéric Bernon
```

```
* api_msg.c: Fix bug #20315: possible memory leak problem if tcp_listen failed,
  because current code doesn't follow rawapi.txt documentation.

2007-07-13 Kieran Mansley
* src/core/tcp_in.c Apply patch#5741 from Oleg Tyshev to fix bug in
  out of sequence processing of received packets

2007-07-03 Simon Goldschmidt
* nearly-all-files: Added assertions where PBUF_RAM pbufs are used and an
  assumption is made that this pbuf is in one piece (i.e. not chained). These
  assumptions clash with the possibility of converting to fully pool-based
  pbuf implementations, where PBUF_RAM pbufs might be chained.

2007-07-03 Simon Goldschmidt
* api.h, api_lib.c, api_msg.c: Final fix for bug #20021 and some other problems
  when closing tcp netconns: removed conn->sem, less context switches when
  closing, both netconn_close and netconn_delete should safely close tcp
  connections.

2007-07-02 Simon Goldschmidt
* ipv4/ip.h, ipv6/ip.h, opt.h, netif.h, etharp.h, ipv4/ip.c, netif.c, raw.c,
  tcp_out.c, udp.c, etharp.c: Added option LWIP_NETIF_HWADDRHINT (default=off)
  to cache ARP table indices with each pcb instead of single-entry cache for
  the complete stack.

2007-07-02 Simon Goldschmidt
* tcp.h, tcp.c, tcp_in.c, tcp_out.c: Added some ASSERTS and casts to prevent
  warnings when assigning to smaller types.

2007-06-28 Simon Goldschmidt
* tcp_out.c: Added check to prevent tcp_pcb->snd_queuelen from overflowing.

2007-06-28 Simon Goldschmidt
* tcp.h: Fixed bug #20287: Fixed nagle algorithm (sending was done too early if
  a segment contained chained pbufs)

2007-06-28 Frédéric Bernon
* autoip.c: replace most of rand() calls by a macro LWIP_AUTOIP_RAND which  ↩
   compute
  a "pseudo-random" value based on netif's MAC and some autoip fields. It's  ↩
     always
  possible to define this macro in your own lwipopts.h to always use C library's
  rand(). Note that autoip_create_rand_addr doesn't use this macro.

2007-06-28 Frédéric Bernon
* netifapi.h, netifapi.c, tcpip.h, tcpip.c: Update code to handle the option
  LWIP_TCPIP_CORE_LOCKING, and do some changes to be coherent with last  ↩
     modifications
  in api_lib/api_msg (use pointers and not type with table, etc...)

2007-06-26 Simon Goldschmidt
* udp.h: Fixed bug #20259: struct udp_hdr was lacking the packin defines.

2007-06-25 Simon Goldschmidt
* udp.c: Fixed bug #20253: icmp_dest_unreach was called with a wrong p->payload
  for udp packets with no matching pcb.
```

```
2007-06-25 Simon Goldschmidt
* udp.c: Fixed bug #20220: UDP PCB search in udp_input(): a non-local match
  could get udp input packets if the remote side matched.

2007-06-13 Simon Goldschmidt
* netif.c: Fixed bug #20180 (TCP pcbs listening on IP_ADDR_ANY could get
  changed in netif_set_ipaddr if previous netif->ip_addr.addr was 0.

2007-06-13 Simon Goldschmidt
* api_msg.c: pcb_new sets conn->err if protocol is not implemented
  -> netconn_new_..() does not allocate a new connection for unsupported
  protocols.

2007-06-13 Frédéric Bernon, Simon Goldschmidt
* api_lib.c: change return expression in netconn_addr and netconn_peer, because
  conn->err was reset to ERR_OK without any reasons (and error was lost)...

2007-06-13 Frédéric Bernon, Matthias Weisser
* opt.h, mem.h, mem.c, memp.c, pbuf.c, ip_frag.c, vj.c: Fix bug #20162. Rename
  MEM_ALIGN in LWIP_MEM_ALIGN and MEM_ALIGN_SIZE in LWIP_MEM_ALIGN_SIZE to avoid
  some macro names collision with some OS macros.

2007-06-11 Simon Goldschmidt
* udp.c: UDP Lite: corrected the use of chksum_len (based on RFC3828: if it's 0,
  create checksum over the complete packet. On RX, if it's < 8 (and not 0),
  discard the packet. Also removed the duplicate 'udphdr->chksum = 0' for both
  UDP & UDP Lite.

2007-06-11 Srinivas Gollakota & Oleg Tyshev
* tcp_out.c: Fix for bug #20075 : "A problem with keep-alive timer and TCP flags ↩
    "
  where TCP flags wasn't initialized in tcp_keepalive.

2007-06-03 Simon Goldschmidt
* udp.c: udp_input(): Input pbuf was not freed if pcb had no recv function
  registered, p->payload was modified without modifying p->len if sending
  icmp_dest_unreach() (had no negative effect but was definitively wrong).

2007-06-03 Simon Goldschmidt
* icmp.c: Corrected bug #19937: For responding to an icmp echo request, icmp
  re-used the input pbuf even if that didn't have enough space to include the
  link headers. Now the space is tested and a new pbuf is allocated for the
  echo response packet if the echo request pbuf isn't big enough.

2007-06-01 Simon Goldschmidt
* sockets.c: Checked in patch #5914: Moved sockopt processing into tcpip_thread.

2007-05-23 Frédéric Bernon
* api_lib.c, sockets.c: Fixed bug #5958 for netconn_listen (acceptmbox only
  allocated by do_listen if success) and netconn_accept errors handling. In
  most of api_lib functions, we replace some errors checkings like "if (conn== ↩
      NULL)"
  by ASSERT, except for netconn_delete.

2007-05-23 Frédéric Bernon
* api_lib.c: Fixed bug #5957 "Safe-thread problem inside netconn_recv" to return
  an error code if it's impossible to fetch a pbuf on a TCP connection (and not
  directly close the recvmbox).

2007-05-22 Simon Goldschmidt
* tcp.c: Fixed bug #1895 (tcp_bind not correct) by introducing a list of
  bound but unconnected (and non-listening) tcp_pcbs.

2007-05-22 Frédéric Bernon
* sys.h, sys.c, api_lib.c, tcpip.c: remove sys_mbox_fetch_timeout() (was only
  used for LWIP_SO_RCVTIMEO option) and use sys_arch_mbox_fetch() instead of
  sys_mbox_fetch() in api files. Now, users SHOULD NOT use internal lwIP  ↩
      features
  like "sys_timeout" in their application threads.

2007-05-22 Frédéric Bernon
* api.h, api_lib.c, api_msg.h, api_msg.c: change the struct api_msg_msg to see
  which parameters are used by which do_xxx function, and to avoid "misusing"
  parameters (patch #5938).

2007-05-22 Simon Goldschmidt
* api_lib.c, api_msg.c, raw.c, api.h, api_msg.h, raw.h: Included patch #5938:
  changed raw_pcb.protocol from u16_t to u8_t since for IPv4 and IPv6, proto
  is only 8 bits wide. This affects the api, as there, the protocol was
  u16_t, too.

2007-05-18 Simon Goldschmidt
* memp.c: addition to patch #5913: smaller pointer was returned but
  memp_memory was the same size -> did not save memory.

2007-05-16 Simon Goldschmidt
* loopif.c, slipif.c: Fix bug #19729: free pbuf if netif->input() returns
  != ERR_OK.

2007-05-16 Simon Goldschmidt
* api_msg.c, udp.c: If a udp_pcb has a local_ip set, check if it is the same
  as the one of the netif used for sending to prevent sending from old
  addresses after a netif address gets changed (partly fixes bug #3168).

2007-05-16 Frédéric Bernon
* tcpip.c, igmp.h, igmp.c: Fixed bug "#19800 : IGMP: igmp_tick() will not work
  with NO_SYS=1". Note that igmp_init is always in tcpip_thread (and not in
  tcpip_init) because we have to be sure that network interfaces are already
  added (mac filter is updated only in igmp_init for the moment).

2007-05-16 Simon Goldschmidt
* mem.c, memp.c: Removed semaphores from memp, changed sys_sem_wait calls
  into sys_arch_sem_wait calls to prevent timers from running while waiting
  for the heap. This fixes bug #19167.

2007-05-13 Simon Goldschmidt
* tcp.h, sockets.h, sockets.c: Fixed bug from patch #5865 by moving the defines
  for socket options (lwip_set/-getsockopt) used with level IPPROTO_TCP from
  tcp.h to sockets.h.

2007-05-07 Simon Goldschmidt
* mem.c: Another attempt to fix bug #17922.

2007-05-04 Simon Goldschmidt
```

```
* pbuf.c, pbuf.h, etharp.c: Further update to ARP queueing: Changed pbuf_copy()
  implementation so that it can be reused (don't allocate the target
  pbuf inside pbuf_copy()).

2007-05-04 Simon Goldschmidt
* memp.c: checked in patch #5913: in memp_malloc() we can return memp as mem
  to save a little RAM (next pointer of memp is not used while not in pool).

2007-05-03 "maq"
* sockets.c: Fix ioctl FIONREAD when some data remains from last recv.
  (patch #3574).

2007-04-23 Simon Goldschmidt
* loopif.c, loopif.h, opt.h, src/netif/FILES: fix bug #2595: "loopif results
  in NULL reference for incoming TCP packets". Loopif has to be configured
  (using LWIP_LOOPIF_MULTITHREADING) to directly call netif->input()
  (multithreading environments, e.g. netif->input() = tcpip_input()) or
  putting packets on a list that is fed to the stack by calling loopif_poll()
  (single-thread / NO_SYS / polling environment where e.g.
  netif->input() = ip_input).

2007-04-17 Jonathan Larmour
* pbuf.c: Use s32_t in pbuf_realloc(), as an s16_t can't reliably hold
  the difference between two u16_t's.
* sockets.h: FD_SETSIZE needs to match number of sockets, which is
  MEMP_NUM_NETCONN in sockets.c right now.

2007-04-12 Jonathan Larmour
* icmp.c: Reset IP header TTL in ICMP ECHO responses (bug #19580).

2007-04-12 Kieran Mansley
* tcp.c, tcp_in.c, tcp_out.c, tcp.h: Modify way the retransmission
  timer is reset to fix bug#19434, with help from Oleg Tyshev.

2007-04-11 Simon Goldschmidt
* etharp.c, pbuf.c, pbuf.h: 3rd fix for bug #11400 (arp-queuing): More pbufs  ↩
   than
  previously thought need to be copied (everything but PBUF_ROM!). Cleaned up
  pbuf.c: removed functions no needed any more (by etharp).

2007-04-11 Kieran Mansley
* inet.c, ip_addr.h, sockets.h, sys.h, tcp.h: Apply patch #5745: Fix
  "Constant is long" warnings with 16bit compilers.  Contributed by
  avatar@mmlab.cse.yzu.edu.tw

2007-04-05 Frédéric Bernon, Jonathan Larmour
* api_msg.c: Fix bug #16830: "err_tcp() posts to connection mailbox when no pend ↩
    on
  the mailbox is active". Now, the post is only done during a connect, and  ↩
     do_send,
  do_write and do_join_leave_group don't do anything if a previous error was  ↩
     signaled.

2007-04-03 Frédéric Bernon
* ip.c: Don't set the IP_DF ("Don't fragment") flag in the IP header in IP  ↩
   output
  packets. See patch #5834.
```

```
2007-03-30 Frédéric Bernon
* api_msg.c: add a "pcb_new" helper function to avoid redundant code, and to add
  missing  pcb allocations checking (in do_bind, and for each raw_new). Fix  ↩
     style.

2007-03-30 Frédéric Bernon
* most of files: prefix all debug.h define with "LWIP_" to avoid any conflict  ↩
   with
  others environment defines (these were too "generic").

2007-03-28 Frédéric Bernon
* api.h, api_lib.c, sockets.c: netbuf_ref doesn't check its internal pbuf_alloc  ↩
   call
  result and can cause a crash. lwip_send now check netbuf_ref result.

2007-03-28 Simon Goldschmidt
* sockets.c Remove "#include <errno.h>" from sockets.c to avoid multiple
  definition of macros (in errno.h and lwip/arch.h) if LWIP_PROVIDE_ERRNO is
  defined. This is the way it should have been already (looking at
  doc/sys_arch.txt)

2007-03-28 Kieran Mansley
* opt.h Change default PBUF_POOL_BUFSIZE (again) to accomodate default MSS +
  IP and TCP headers *and* physical link headers

2007-03-26 Frédéric Bernon (based on patch from Dmitry Potapov)
* api_lib.c: patch for netconn_write(), fixes a possible race condition which  ↩
   cause
  to send some garbage. It is not a definitive solution, but the patch does  ↩
     solve
  the problem for most cases.

2007-03-22 Frédéric Bernon
* api_msg.h, api_msg.c: Remove obsolete API_MSG_ACCEPT and do_accept (never used ↩
   ).

2007-03-22 Frédéric Bernon
* api_lib.c: somes resources couldn't be freed if there was errors during
  netconn_new_with_proto_and_callback.

2007-03-22 Frédéric Bernon
* ethernetif.c: update netif->input calls to check return value. In older ports,
  it's a good idea to upgrade them, even if before, there could be another  ↩
     problem
  (access to an uninitialized mailbox).

2007-03-21 Simon Goldschmidt
* sockets.c: fixed bug #5067 (essentialy a signed/unsigned warning fixed
  by casting to unsigned).

2007-03-21 Frédéric Bernon
* api_lib.c, api_msg.c, tcpip.c: integrate sys_mbox_fetch(conn->mbox, NULL)  ↩
   calls from
  api_lib.c to tcpip.c's tcpip_apimsg(). Now, use a local variable and not a
  dynamic one from memp to send tcpip_msg to tcpip_thread in a synchrone call.
  Free tcpip_msg from tcpip_apimsg is not done in tcpip_thread. This give a
```

```
    faster and more reliable communication between api_lib and tcpip.

  2007-03-21 Frédéric Bernon
  * opt.h: Add LWIP_NETIF_CALLBACK (to avoid compiler warning) and set it to 0.

  2007-03-21 Frédéric Bernon
  * api_msg.c, igmp.c, igmp.h: Fix C++ style comments

  2007-03-21 Kieran Mansley
  * opt.h Change default PBUF_POOL_BUFSIZE to accomodate default MSS +
    IP and TCP headers

  2007-03-21 Kieran Mansley
  * Fix all uses of pbuf_header to check the return value.  In some
    cases just assert if it fails as I'm not sure how to fix them, but
    this is no worse than before when they would carry on regardless
    of the failure.

  2007-03-21 Kieran Mansley
  * sockets.c, igmp.c, igmp.h, memp.h: Fix C++ style comments and
    comment out missing header include in icmp.c

  2007-03-20 Frédéric Bernon
  * memp.h, stats.c: Fix stats_display function where memp_names table wasn't
    synchronized with memp.h.

  2007-03-20 Frédéric Bernon
  * tcpip.c: Initialize tcpip's mbox, and verify if initialized in tcpip_input,
    tcpip_ethinput, tcpip_callback, tcpip_apimsg, to fix a init problem with
    network interfaces. Also fix a compiler warning.

  2007-03-20 Kieran Mansley
  * udp.c: Only try and use pbuf_header() to make space for headers if
    not a ROM or REF pbuf.

  2007-03-19 Frédéric Bernon
  * api_msg.h, api_msg.c, tcpip.h, tcpip.c: Add return types to tcpip_apimsg()
    and api_msg_post().

  2007-03-19 Frédéric Bernon
  * Remove unimplemented "memp_realloc" function from memp.h.

  2007-03-11 Simon Goldschmidt
  * pbuf.c: checked in patch #5796: pbuf_alloc: len field claculation caused
    memory corruption.

  2007-03-11 Simon Goldschmidt (based on patch from Dmitry Potapov)
  * api_lib.c, sockets.c, api.h, api_msg.h, sockets.h: Fixed bug #19251
    (missing `const' qualifier in socket functions), to get more compatible to
    standard POSIX sockets.

  2007-03-11 Frédéric Bernon (based on patch from Dmitry Potapov)
  * sockets.c: Add asserts inside bind, connect and sendto to check input
    parameters. Remove excessive set_errno() calls after get_socket(), because
    errno is set inside of get_socket(). Move last sock_set_errno() inside
    lwip_close.
```

```
2007-03-09 Simon Goldschmidt
* memp.c: Fixed bug #11400: New etharp queueing introduced bug: memp_memory
  was allocated too small.

2007-03-06 Simon Goldschmidt
* tcpip.c: Initialize dhcp timers in tcpip_thread (if LWIP_DHCP) to protect
  the stack from concurrent access.

2007-03-06 Frédéric Bernon, Dmitry Potapov
* tcpip.c, ip_frag.c, ethernetif.c: Fix some build problems, and a redundancy
  call to "lwip_stats.link.recv++;" in low_level_input() & ethernetif_input().

2007-03-06 Simon Goldschmidt
* ip_frag.c, ip_frag.h: Reduce code size: don't include code in those files
  if IP_FRAG == 0 and IP_REASSEMBLY == 0

2007-03-06 Frédéric Bernon, Simon Goldschmidt
* opt.h, ip_frag.h, tcpip.h, tcpip.c, ethernetif.c: add new configuration
  option named ETHARP_TCPIP_ETHINPUT, which enable the new tcpip_ethinput.
  Allow to do ARP processing for incoming packets inside tcpip_thread
  (protecting ARP layer against concurrent access). You can also disable
  old code using tcp_input with new define ETHARP_TCPIP_INPUT set to 0.
  Older ports have to use tcpip_ethinput.

2007-03-06 Simon Goldschmidt (based on patch from Dmitry Potapov)
* err.h, err.c: fixed compiler warning "initialization dircards qualifiers
  from pointer target type"

2007-03-05 Frédéric Bernon
* opt.h, sockets.h: add new configuration options (LWIP_POSIX_SOCKETS_IO_NAMES,
  ETHARP_TRUST_IP_MAC, review SO_REUSE)

2007-03-04 Frédéric Bernon
* api_msg.c: Remove some compiler warnings : parameter "pcb" was never
  referenced.

2007-03-04 Frédéric Bernon
* api_lib.c: Fix "[patch #5764] api_lib.c cleanup: after patch #5687" (from
  Dmitry Potapov).
  The api_msg struct stay on the stack (not moved to netconn struct).

2007-03-04 Simon Goldschmidt (based on patch from Dmitry Potapov)
* pbuf.c: Fix BUG#19168 - pbuf_free can cause deadlock (if
  SYS_LIGHTWEIGHT_PROT=1 & freeing PBUF_RAM when mem_sem is not available)
  Also fixed cast warning in pbuf_alloc()

2007-03-04 Simon Goldschmidt
* etharp.c, etharp.h, memp.c, memp.h, opt.h: Fix BUG#11400 - don't corrupt
  existing pbuf chain when enqueuing multiple pbufs to a pending ARP request

2007-03-03 Frédéric Bernon
* udp.c: remove obsolete line "static struct udp_pcb *pcb_cache = NULL;"
  It is static, and never used in udp.c except udp_init().

2007-03-02 Simon Goldschmidt
* tcpip.c: Moved call to ip_init(), udp_init() and tcp_init() from
  tcpip_thread() to tcpip_init(). This way, raw API connections can be
```

```
    initialized before tcpip_thread is running (e.g. before OS is started)

  2007-03-02 Frédéric Bernon
  * rawapi.txt: Fix documentation mismatch with etharp.h about etharp_tmr's call
    interval.

  2007-02-28 Kieran Mansley
  * pbuf.c: Fix BUG#17645 - ensure pbuf payload pointer is not moved
    outside the region of the pbuf by pbuf_header()

  2007-02-28 Kieran Mansley
  * sockets.c: Fix BUG#19161 - ensure milliseconds timeout is non-zero
    when supplied timeout is also non-zero

(STABLE-1.2.0)

  2006-12-05 Leon Woestenberg
  * CHANGELOG: Mention STABLE-1.2.0 release.

  ++ New features:

  2006-12-01 Christiaan Simons
  * mem.h, opt.h: Added MEM_LIBC_MALLOC option.
    Note this is a workaround. Currently I have no other options left.

  2006-10-26 Christiaan Simons (accepted patch by Jonathan Larmour)
  * ipv4/ip_frag.c: rename MAX_MTU to IP_FRAG_MAX_MTU and move define
    to include/lwip/opt.h.
  * ipv4/lwip/ip_frag.h: Remove unused IP_REASS_INTERVAL.
    Move IP_REASS_MAXAGE and IP_REASS_BUFSIZE to include/lwip/opt.h.
  * opt.h: Add above new options.

  2006-08-18 Christiaan Simons
  * tcp_{in,out}.c: added SNMP counters.
  * ipv4/ip.c: added SNMP counters.
  * ipv4/ip_frag.c: added SNMP counters.

  2006-08-08 Christiaan Simons
  * etharp.{c,h}: added etharp_find_addr() to read
    (stable) ethernet/IP address pair from ARP table

  2006-07-14 Christiaan Simons
  * mib_structs.c: added
  * include/lwip/snmp_structs.h: added
  * netif.{c,h}, netif/ethernetif.c: added SNMP statistics to netif struct

  2006-07-06 Christiaan Simons
  * snmp/asn1_{enc,dec}.c added
  * snmp/mib2.c added
  * snmp/msg_{in,out}.c added
  * include/lwip/snmp_asn1.h added
  * include/lwip/snmp_msg.h added
  * doc/snmp_agent.txt added

  2006-03-29 Christiaan Simons
  * inet.c, inet.h: Added platform byteswap support.
    Added LWIP_PLATFORM_BYTESWAP define (defaults to 0) and
    optional LWIP_PLATFORM_HTONS(), LWIP_PLATFORM_HTONL() macros.

  ++ Bug fixes:

  2006-11-30 Christiaan Simons
  * dhcp.c: Fixed false triggers of request_timeout.

  2006-11-28 Christiaan Simons
  * netif.c: In netif_add() fixed missing clear of ip_addr, netmask, gw and flags.

  2006-10-11 Christiaan Simons
  * api_lib.c etharp.c, ip.c, memp.c, stats.c, sys.{c,h} tcp.h:
    Partially accepted patch #5449 for ANSI C compatibility / build fixes.
  * ipv4/lwip/ip.h ipv6/lwip/ip.h: Corrected UDP-Lite protocol
    identifier from 170 to 136 (bug #17574).

  2006-10-10 Christiaan Simons
  * api_msg.c: Fixed Nagle algorithm as reported by Bob Grice.

  2006-08-17 Christiaan Simons
  * udp.c: Fixed bug #17200, added check for broadcast
    destinations for PCBs bound to a unicast address.

  2006-08-07 Christiaan Simons
  * api_msg.c: Flushing TCP output in do_close() (bug #15926).

  2006-06-27 Christiaan Simons
  * api_msg.c: Applied patch for cold case (bug #11135).
    In accept_function() ensure newconn->callback is always initialized.

  2006-06-15 Christiaan Simons
  * mem.h: added MEM_SIZE_F alias to fix an ancient cold case (bug #1748),
    facilitate printing of mem_size_t and u16_t statistics.

  2006-06-14 Christiaan Simons
  * api_msg.c: Applied patch #5146 to handle allocation failures
    in accept() by Kevin Lawson.

  2006-05-26 Christiaan Simons
  * api_lib.c: Removed conn->sem creation and destruction
    from netconn_write() and added sys_sem_new to netconn_new_*.

(STABLE-1_1_1)

  2006-03-03  Christiaan Simons
  * ipv4/ip_frag.c: Added bound-checking assertions on ip_reassbitmap
    access and added pbuf_alloc() return value checks.

  2006-01-01  Leon Woestenberg <leon.woestenberg@gmx.net>
  * tcp_{in,out}.c, tcp_out.c: Removed 'even sndbuf' fix in TCP, which is
    now handled by the checksum routine properly.

  2006-02-27  Leon Woestenberg <leon.woestenberg@gmx.net>
   * pbuf.c: Fix alignment; pbuf_init() would not work unless
     pbuf_pool_memory[] was properly aligned. (Patch by Curt McDowell.)

  2005-12-20  Leon Woestenberg <leon.woestenberg@gmx.net>
  * tcp.c: Remove PCBs which stay in LAST_ACK state too long. Patch
    submitted by Mitrani Hiroshi.

  2005-12-15  Christiaan Simons
  * inet.c: Disabled the added summing routine to preserve code space.

  2005-12-14  Leon Woestenberg <leon.woestenberg@gmx.net>
  * tcp_in.c: Duplicate FIN ACK race condition fix by Kelvin Lawson.
    Added Curt McDowell's optimized checksumming routine for future
    inclusion. Need to create test case for unaliged, aligned, odd,
    even length combination of cases on various endianess machines.

  2005-12-09  Christiaan Simons
  * inet.c: Rewrote standard checksum routine in proper portable C.

  2005-11-25  Christiaan Simons
  * udp.c tcp.c: Removed SO_REUSE hack. Should reside in socket code only.
  * *.c: introduced cc.h LWIP_DEBUG formatters matching the u16_t, s16_t,
    u32_t, s32_t typedefs. This solves most debug word-length assumes.

  2005-07-17 Leon Woestenberg <leon.woestenberg@gmx.net>
  * inet.c: Fixed unaligned 16-bit access in the standard checksum
    routine by Peter Jolasson.
  * slipif.c: Fixed implementation assumption of single-pbuf datagrams.

  2005-02-04 Leon Woestenberg <leon.woestenberg@gmx.net>
  * tcp_out.c: Fixed uninitialized 'queue' referenced in memerr branch.
  * tcp_{out|in}.c: Applied patch fixing unaligned access.

  2005-01-04 Leon Woestenberg <leon.woestenberg@gmx.net>
  * pbuf.c: Fixed missing semicolon after LWIP_DEBUG statement.

  2005-01-03 Leon Woestenberg <leon.woestenberg@gmx.net>
  * udp.c: UDP pcb->recv() was called even when it was NULL.

(STABLE-1_1_0)

  2004-12-28 Leon Woestenberg <leon.woestenberg@gmx.net>
  * etharp.*: Disabled multiple packets on the ARP queue.
    This clashes with TCP queueing.

  2004-11-28 Leon Woestenberg <leon.woestenberg@gmx.net>
  * etharp.*: Fixed race condition from ARP request to ARP timeout.
    Halved the ARP period, doubled the period counts.
    ETHARP_MAX_PENDING now should be at least 2. This prevents
    the counter from reaching 0 right away (which would allow
    too little time for ARP responses to be received).

  2004-11-25 Leon Woestenberg <leon.woestenberg@gmx.net>
  * dhcp.c: Decline messages were not multicast but unicast.
  * etharp.c: ETHARP_CREATE is renamed to ETHARP_TRY_HARD.
    Do not try hard to insert arbitrary packet's source address,
    etharp_ip_input() now calls etharp_update() without ETHARP_TRY_HARD.
    etharp_query() now always DOES call ETHARP_TRY_HARD so that users
    querying an address will see it appear in the cache (DHCP could
    suffer from this when a server invalidly gave an in-use address.)
  * ipv4/ip_addr.h: Renamed ip_addr_maskcmp() to _netcmp() as we are
```

```
    comparing network addresses (identifiers), not the network masks
    themselves.
  * ipv4/ip_addr.c: ip_addr_isbroadcast() now checks that the given
    IP address actually belongs to the network of the given interface.

  2004-11-24 Kieran Mansley <kjm25@cam.ac.uk>
  * tcp.c: Increment pcb->snd_buf when ACK is received in SYN_SENT state.

(STABLE-1_1_0-RC1)

  2004-10-16 Kieran Mansley <kjm25@cam.ac.uk>
  * tcp.c: Add code to tcp_recved() to send an ACK (window update) immediately,
    even if one is already pending, if the rcv_wnd is above a threshold
    (currently TCP_WND/2). This avoids waiting for a timer to expire to send a
    delayed ACK in order to open the window if the stack is only receiving data.

  2004-09-12 Kieran Mansley <kjm25@cam.ac.uk>
  * tcp*.*: Retransmit time-out handling improvement by Sam Jansen.

  2004-08-20 Tony Mountifield <tony@softins.co.uk>
  * etharp.c: Make sure the first pbuf queued on an ARP entry
    is properly ref counted.

  2004-07-27 Tony Mountifield <tony@softins.co.uk>
  * debug.h: Added (int) cast in LWIP_DEBUGF() to avoid compiler
    warnings about comparison.
  * pbuf.c: Stopped compiler complaining of empty if statement
    when LWIP_DEBUGF() empty.  Closed an unclosed comment.
  * tcp.c: Stopped compiler complaining of empty if statement
    when LWIP_DEBUGF() empty.
  * ip.h Corrected IPH_TOS() macro: returns a byte, so doesn’t need htons().
  * inet.c: Added a couple of casts to quiet the compiler.
    No need to test isascii(c) before isdigit(c) or isxdigit(c).

  2004-07-22 Tony Mountifield <tony@softins.co.uk>
  * inet.c: Made data types consistent in inet_ntoa().
    Added casts for return values of checksum routines, to pacify compiler.
  * ip_frag.c, tcp_out.c, sockets.c, pbuf.c
    Small corrections to some debugging statements, to pacify compiler.

  2004-07-21 Tony Mountifield <tony@softins.co.uk>
  * etharp.c: Removed spurious semicolon and added missing end-of-comment.
  * ethernetif.c Updated low_level_output() to match prototype for
    netif->linkoutput and changed low_level_input() similarly for consistency.
  * api_msg.c: Changed recv_raw() from int to u8_t, to match prototype
    of raw_recv() in raw.h and so avoid compiler error.
  * sockets.c: Added trivial (int) cast to keep compiler happier.
  * ip.c, netif.c Changed debug statements to use the tidier ip4_addrN() macros.

(STABLE-1_0_0)

  ++ Changes:

  2004-07-05 Leon Woestenberg <leon.woestenberg@gmx.net>
  * sockets.*: Restructured LWIP_PRIVATE_TIMEVAL. Make sure
    your cc.h file defines this either 1 or 0. If non-defined,
    defaults to 1.
  * .c: Added <string.h> and <errno.h> includes where used.
  * etharp.c: Made some array indices unsigned.

  2004-06-27 Leon Woestenberg <leon.woestenberg@gmx.net>
  * netif.*: Added netif_set_up()/down().
  * dhcp.c: Changes to restart program flow.

  2004-05-07 Leon Woestenberg <leon.woestenberg@gmx.net>
  * etharp.c: In find_entry(), instead of a list traversal per candidate, do a
    single-pass lookup for different candidates. Should exploit locality.

  2004-04-29 Leon Woestenberg <leon.woestenberg@gmx.net>
  * tcp*.c: Cleaned up source comment documentation for Doxygen processing.
  * opt.h: ETHARP_ALWAYS_INSERT option removed to comply with ARP RFC.
  * etharp.c: update_arp_entry() only adds new ARP entries when adviced to by
    the caller. This deprecates the ETHARP_ALWAYS_INSERT overrule option.

  ++ Bug fixes:

  2004-04-27 Leon Woestenberg <leon.woestenberg@gmx.net>
  * etharp.c: Applied patch of bug #8708 by Toni Mountifield with a solution
    suggested by Timmy Brolin. Fix for 32-bit processors that cannot access
    non-aligned 32-bit words, such as soms 32-bit TCP/IP header fields. Fix
    is to prefix the 14-bit Ethernet headers with two padding bytes.

  2004-04-23 Leon Woestenberg <leon.woestenberg@gmx.net>
  * ip_addr.c: Fix in the ip_addr_isbroadcast() check.
  * etharp.c: Fixed the case where the packet that initiates the ARP request
    is not queued, and gets lost. Fixed the case where the packets destination
    address is already known; we now always queue the packet and perform an ARP
    request.

(STABLE-0_7_0)

  ++ Bug fixes:

  * Fixed TCP bug for SYN_SENT to ESTABLISHED state transition.
  * Fixed TCP bug in dequeueing of FIN from out of order segment queue.
  * Fixed two possible NULL references in rare cases.

(STABLE-0_6_6)

  ++ Bug fixes:

  * Fixed DHCP which did not include the IP address in DECLINE messages.

  ++ Changes:

  * etharp.c has been hauled over a bit.

(STABLE-0_6_5)

  ++ Bug fixes:

  * Fixed TCP bug induced by bad window resizing with unidirectional TCP traffic.
  * Packets sent from ARP queue had invalid source hardware address.
  ++ Changes:

  * Pass-by ARP requests do now update the cache.

  ++ New features:

  * No longer dependent on ctype.h.
  * New socket options.
  * Raw IP pcb support.

(STABLE-0_6_4)

  ++ Bug fixes:

  * Some debug formatters and casts fixed.
  * Numereous fixes in PPP.

  ++ Changes:

  * DEBUGF now is LWIP_DEBUGF
  * pbuf_dechain() has been re-enabled.
  * Mentioned the changed use of CVS branches in README.

(STABLE-0_6_3)

  ++ Bug fixes:

  * Fixed pool pbuf memory leak in pbuf_alloc().
    Occured if not enough PBUF_POOL pbufs for a packet pbuf chain.
    Reported by Savin Zlobec.

  * PBUF_POOL chains had their tot_len field not set for non-first
    pbufs. Fixed in pbuf_alloc().

  ++ New features:

  * Added PPP stack contributed by Marc Boucher

  ++ Changes:

  * Now drops short packets for ICMP/UDP/TCP protocols. More robust.

  * ARP queueuing now queues the latest packet instead of the first.
    This is the RFC recommended behaviour, but can be overridden in
    lwipopts.h.

(0.6.2)

  ++ Bugfixes:

  * TCP has been fixed to deal with the new use of the pbuf->ref
    counter.

  * DHCP dhcp_inform() crash bug fixed.

  ++ Changes:
```

```
  * Removed pbuf_pool_free_cache and pbuf_pool_alloc_cache. Also removed
    pbuf_refresh(). This has sped up pbuf pool operations considerably.
    Implemented by David Haas.

(0.6.1)

  ++ New features:

  * The packet buffer implementation has been enhanced to support
    zero-copy and copy-on-demand for packet buffers which have their
    payloads in application-managed memory.
    Implemented by David Haas.

    Use PBUF_REF to make a pbuf refer to RAM. lwIP will use zero-copy
    if an outgoing packet can be directly sent on the link, or perform
    a copy-on-demand when necessary.

    The application can safely assume the packet is sent, and the RAM
    is available to the application directly after calling udp_send()
    or similar function.

  ++ Bugfixes:

  * ARP_QUEUEING should now correctly work for all cases, including
    PBUF_REF.
    Implemented by Leon Woestenberg.

  ++ Changes:

  * IP_ADDR_ANY is no longer a NULL pointer. Instead, it is a pointer
    to a '0.0.0.0' IP address.

  * The packet buffer implementation is changed. The pbuf->ref counter
    meaning has changed, and several pbuf functions have been
    adapted accordingly.

  * netif drivers have to be changed to set the hardware address length field
    that must be initialized correctly by the driver (hint: 6 for Ethernet MAC).
    See the contrib/ports/c16x cs8900 driver as a driver example.

  * netif's have a dhcp field that must be initialized to NULL by the driver.
    See the contrib/ports/c16x cs8900 driver as a driver example.

(0.5.x) This file has been unmaintained up to 0.6.1. All changes are
  logged in CVS but have not been explained here.

(0.5.3) Changes since version 0.5.2

  ++ Bugfixes:

  * memp_malloc(MEMP_API_MSG) could fail with multiple application
    threads because it wasn't protected by semaphores.

  ++ Other changes:

  * struct ip_addr now packed.
  * The name of the time variable in arp.c has been changed to ctime
    to avoid conflicts with the time() function.

(0.5.2) Changes since version 0.5.1

  ++ New features:

  * A new TCP function, tcp_tmr(), now handles both TCP timers.

  ++ Bugfixes:

  * A bug in tcp_parseopt() could cause the stack to hang because of a
    malformed TCP option.

  * The address of new connections in the accept() function in the BSD
    socket library was not handled correctly.

  * pbuf_dechain() did not update the ->tot_len field of the tail.

  * Aborted TCP connections were not handled correctly in all
    situations.

  ++ Other changes:

  * All protocol header structs are now packed.

  * The ->len field in the tcp_seg structure now counts the actual
    amount of data, and does not add one for SYN and FIN segments.

(0.5.1) Changes since version 0.5.0

  ++ New features:

  * Possible to run as a user process under Linux.

  * Preliminary support for cross platform packed structs.

  * ARP timer now implemented.

  ++ Bugfixes:

  * TCP output queue length was badly initialized when opening
    connections.

  * TCP delayed ACKs were not sent correctly.

  * Explicit initialization of BSS segment variables.

  * read() in BSD socket library could drop data.

  * Problems with memory alignment.

  * Situations when all TCP buffers were used could lead to
    starvation.

  * TCP MSS option wasn't parsed correctly.
  * Problems with UDP checksum calculation.

  * IP multicast address tests had endianess problems.

  * ARP requests had wrong destination hardware address.

  ++ Other changes:

  * struct eth_addr changed from u16_t[3] array to u8_t[6].

  * A ->linkoutput() member was added to struct netif.

  * TCP and UDP ->dest_* struct members where changed to ->remote_*.

  * ntoh* macros are now null definitions for big endian CPUs.

(0.5.0) Changes since version 0.4.2

  ++ New features:

  * Redesigned operating system emulation layer to make porting easier.

  * Better control over TCP output buffers.

  * Documenation added.

  ++ Bugfixes:

  * Locking issues in buffer management.

  * Bugfixes in the sequential API.

  * IP forwarding could cause memory leakage. This has been fixed.

  ++ Other changes:

  * Directory structure somewhat changed; the core/ tree has been
    collapsed.

(0.4.2) Changes since version 0.4.1

  ++ New features:

  * Experimental ARP implementation added.

  * Skeleton Ethernet driver added.

  * Experimental BSD socket API library added.

  ++ Bugfixes:

  * In very intense situations, memory leakage could occur. This has
    been fixed.

  ++ Other changes:

  * Variables named "data" and "code" have been renamed in order to
```

```
    avoid name conflicts in certain compilers.

  * Variable++ have in appliciable cases been translated to ++variable
    since some compilers generate better code in the latter case.

(0.4.1) Changes since version 0.4

  ++ New features:

  * TCP: Connection attempts time out earlier than data
    transmissions. Nagle algorithm implemented. Push flag set on the
    last segment in a burst.

  * UDP: experimental support for UDP-Lite extensions.

  ++ Bugfixes:

  * TCP: out of order segments were in some cases handled incorrectly,
    and this has now been fixed. Delayed acknowledgements was broken
    in 0.4, has now been fixed. Binding to an address that is in use
    now results in an error. Reset connections sometimes hung an
    application; this has been fixed.

  * Checksum calculation sometimes failed for chained pbufs with odd
    lengths. This has been fixed.

  * API: a lot of bug fixes in the API. The UDP API has been improved
    and tested. Error reporting and handling has been
    improved. Logical flaws and race conditions for incoming TCP
    connections has been found and removed.

  * Memory manager: alignment issues. Reallocating memory sometimes
    failed, this has been fixed.

  * Generic library: bcopy was flawed and has been fixed.

  ++ Other changes:

  * API: all datatypes has been changed from generic ones such as
    ints, to specified ones such as u16_t. Functions that return
    errors now have the correct type (err_t).

  * General: A lot of code cleaned up and debugging code removed. Many
    portability issues have been fixed.

  * The license was changed; the advertising clause was removed.

  * C64 port added.

  * Thanks: Huge thanks go to Dagan Galarneau, Horst Garnetzke, Petri
    Kosunen, Mikael Caleres, and Frits Wilmink for reporting and
    fixing bugs!

(0.4) Changes since version 0.3.1

  * Memory management has been radically changed; instead of
    allocating memory from a shared heap, memory for objects that are
    rapidly allocated and deallocated is now kept in pools. Allocation
    and deallocation from those memory pools is very fast. The shared
    heap is still present but is used less frequently.

  * The memory, memory pool, and packet buffer subsystems now support
    4-, 2-, or 1-byte alignment.

  * "Out of memory" situations are handled in a more robust way.

  * Stack usage has been reduced.

  * Easier configuration of lwIP parameters such as memory usage,
    TTLs, statistics gathering, etc. All configuration parameters are
    now kept in a single header file "lwipopts.h".

  * The directory structure has been changed slightly so that all
    architecture specific files are kept under the src/arch
    hierarchy.

  * Error propagation has been improved, both in the protocol modules
    and in the API.

  * The code for the RTXC architecture has been implemented, tested
    and put to use.

  * Bugs have been found and corrected in the TCP, UDP, IP, API, and
    the Internet checksum modules.

  * Bugs related to porting between a 32-bit and a 16-bit architecture
    have been found and corrected.

  * The license has been changed slightly to conform more with the
    original BSD license, including the advertisement clause.

(0.3.1) Changes since version 0.3

  * Fix of a fatal bug in the buffer management. Pbufs with allocated
    RAM never returned the RAM when the pbuf was deallocated.

  * TCP congestion control, window updates and retransmissions did not
    work correctly. This has now been fixed.

  * Bugfixes in the API.

(0.3) Changes since version 0.2

  * New and improved directory structure. All include files are now
    kept in a dedicated include/ directory.

  * The API now has proper error handling. A new function,
    netconn_err(), now returns an error code for the connection in
    case of errors.

  * Improvements in the memory management subsystem. The system now
    keeps a pointer to the lowest free memory block. A new function,
    mem_malloc2() tries to allocate memory once, and if it fails tries
    to free some memory and retry the allocation.
  * Much testing has been done with limited memory
    configurations. lwIP now does a better job when overloaded.

  * Some bugfixes and improvements to the buffer (pbuf) subsystem.

  * Many bugfixes in the TCP code:

    - Fixed a bug in tcp_close().

    - The TCP receive window was incorrectly closed when out of
      sequence segments was received. This has been fixed.

    - Connections are now timed-out of the FIN-WAIT-2 state.

    - The initial congestion window could in some cases be too
      large. This has been fixed.

    - The retransmission queue could in some cases be screwed up. This
      has been fixed.

    - TCP RST flag now handled correctly.

    - Out of sequence data was in some cases never delivered to the
      application. This has been fixed.

    - Retransmitted segments now contain the correct acknowledgment
      number and advertised window.

    - TCP retransmission timeout backoffs are not correctly computed
      (ala BSD). After a number of retransmissions, TCP now gives up
      the connection.

  * TCP connections now are kept on three lists, one for active
    connections, one for listening connections, and one for
    connections that are in TIME-WAIT. This greatly speeds up the fast
    timeout processing for sending delayed ACKs.

  * TCP now provides proper feedback to the application when a
    connection has been successfully set up.

  * More comments have been added to the code. The code has also been
    somewhat cleaned up.

(0.2) Initial public release.
```

1.3 How to contribute to lwIP

```
1 Introduction

This document describes some guidelines for people participating
in lwIP development.

2 How to contribute to lwIP
```

```
Here is a short list of suggestions to anybody working with lwIP and
trying to contribute bug reports, fixes, enhancements, platform ports etc.
First of all as you may already know lwIP is a volunteer project so feedback
to fixes or questions might often come late. Hopefully the bug and patch tracking
features of Savannah help us not lose users' input.

2.1 Source code style:

1. do not use tabs.
2. indentation is two spaces per level (i.e. per tab).
3. end debug messages with a trailing newline (\n).
4. one space between keyword and opening bracket.
5. no space between function and opening bracket.
6. one space and no newline before opening curly braces of a block.
7. closing curly brace on a single line.
8. spaces surrounding assignment and comparisons.
9. don't initialize static and/or global variables to zero, the compiler takes ↩
   care of that.
10. use current source code style as further reference.

2.2 Source code documentation style:

1. JavaDoc compliant and Doxygen compatible.
2. Function documentation above functions in .c files, not .h files.
   (This forces you to synchronize documentation and implementation.)
3. Use current documentation style as further reference.

2.3 Bug reports and patches:

1. Make sure you are reporting bugs or send patches against the latest
   sources. (From the latest release and/or the current Git sources.)
2. If you think you found a bug make sure it's not already filed in the
   bugtracker at Savannah.
3. If you have a fix put the patch on Savannah. If it is a patch that affects
   both core and arch specific stuff please separate them so that the core can
   be applied separately while leaving the other patch 'open'. The preferred way
   is to NOT touch archs you can't test and let maintainers take care of them.
   This is a good way to see if they are used at all - the same goes for unix
   netifs except tapif.
4. Do not file a bug and post a fix to it to the patch area. Either a bug report
   or a patch will be enough.
   If you correct an existing bug then attach the patch to the bug rather than ↩
      creating a new entry in the patch area.
5. Patches should be specific to a single change or to related changes. Do not mix ↩
    bugfixes with spelling and other
   trivial fixes unless the bugfix is trivial too. Do not reorganize code and ↩
      rename identifiers in the same patch you
   change behaviour if not necessary. A patch is easier to read and understand if ↩
      it's to the point and short than
   if it's not to the point and long :) so the chances for it to be applied are ↩
      greater.

2.4 Platform porters:

1. If you have ported lwIP to a platform (an OS, a uC/processor or a combination ↩
   of these) and
   you think it could benefit others[1] you might want discuss this on the mailing ↩
       list. You
   can also ask for Git access to submit and maintain your port in the lwIP/ ↩
      contrib subdir.
```

1.4 CMake build system

```
Building lwIP
=============

lwIP uses a CMake based build system.

The CMake files in this project are designed to
be included into your own CMake files. They are
mainly variable definitions containing a list of
source files and predefined static libraries to
be linked against application code.

1) lwIP sources:
   src/Filelists.cmake provides file lists containing
   the lwIP core library.
   The file also contains two static libraries, lwipcore
   and lwipallapps, where you can link your app against.
   This is the file that is useful to all lwIP users.

2) Example applications:
   contrib/Filelists.cmake provides several file lists
   containing the example applications.
   The file also contains several static libraries
   for these example apps.
   This file is only useful for you, if you can use one
   of the examples in your application, which is normally
   not the case.

3) OS/platform port:
   Usually the OS port needs to be provided by the user.
   If a port to Linux, Windows or MacOS is useful for
   you, you can use
   contrib/ports/{win32, unix}/Filelists.cmake
   that contains file lists and libraries for
   these operating systems.

VARIABLES
=========
In all cases, you need to provide two variables.

"LWIP_DIR" pointing to the lwIP directory
Example:
set(LWIP_DIR ${CMAKE_CURRENT_SOURCE_DIR}/externals/lwip)

"LWIP_INCLUDE_DIRS" that contains the include base paths
- for lwIP itself (${LWIP_DIR}/src/include)
- for lwIP contrib if you use it (${LWIP_DIR}/contrib)
- to a directory containing an OS port
- to a directory containing lwipopts.h
```

```
Example:
set (LWIP_INCLUDE_DIRS
    "${LWIP_DIR}/src/include"
    "${LWIP_DIR}/contrib"
    "${LWIP_DIR}/contrib/ports/unix/port/include"
    "${LWIP_DIR}/contrib/examples/example_app"
)

Putting it all together
=======================
To get a working application, your CMake system
needs to provide the variables described above, e.g.
set (LWIP_DIR <path to lwip sources>)
set (LWIP_INCLUDE_DIRS
    "${LWIP_DIR}/src/include"
    "${LWIP_DIR}/contrib"
    "<path to my port>/include"
    "<path to lwipopts.h>"
)

You may add some defines:
set (LWIP_DEFINITIONS LWIP_DEBUG=1)

Then include the filelists you need:
include(${LWIP_DIR}/src/Filelists.cmake)
include(${LWIP_DIR}/contrib/Filelists.cmake)

Then, declare you executable:
add_executable(my_app <my source files> <my lwip port files>)

Add lwIP include dirs to your app:
target_include_directories(my_app PRIVATE ${LWIP_INCLUDE_DIRS})

Link your app against the lwIP libs from the filelists you need:
target_link_libraries(my_app lwipcontribapps lwipallapps lwipcore)

Working example
===============
Working build examples can be found in the
contrib/ports/{win32, unix}/example_app
subdirectory.
To use them, create a build directory and call cmake with
the lwIP root dir:

- mkdir build
- cd build
- cmake ..
- cmake --build .

The CMakeLists.txt will autoselect the correct port
for your system (supported: Linux, Windows, Darwin).

To configure the example app to your needs, you need to copy the file
    contrib/examples/example_app/lwipcfg.h.example
to
    contrib/examples/example_app/lwipcfg.h
```

```
and edit to your needs.

Makefile based build system
===========================
lwIP also maintains file lists for Makefile-based
build systems. Look for Filelists.mk files
in the source tree. We try to maintain them,
but lwIP's mainly focused build system is CMake.

MS Visual Studio
================
lwIP also provides basic support for MSVS in the win32 port
folder under 'msvc'. We try to maintain these files,
but lwIP's mainly focused build system is CMake.
```

1.5 Common pitfalls

Multiple Execution Contexts in lwIP code

The most common source of lwIP problems is to have multiple execution contexts inside the lwIP code.

lwIP can be used in two basic modes: Mainloop mode ("NO_SYS") (no OS/RTOS running on target system) or OS mode (TCPIP thread) (there is an OS running on the target system).

See also: Multithreading (especially the part about LWIP_ASSERT_CORE_LOCKED()!)

Mainloop Mode

In mainloop mode, only "raw" APIs can be used. The user has two possibilities to ensure there is only one execution context at a time in lwIP:

1) Deliver RX ethernet packets directly in interrupt context to lwIP by calling netif->input directly in interrupt. This implies all lwIP callback functions are called in IRQ context, which may cause further problems in application code: IRQ is blocked for a long time, multiple execution contexts in application code etc. When the application wants to call lwIP, it only needs to disable interrupts during the call. If timers are involved, even more locking code is needed to lock out timer IRQ and ethernet IRQ from each other, assuming these may be nested.

2) Run lwIP in a mainloop. There is example code here: Mainloop mode ("NO_SYS"). lwIP is *ONLY* called from mainloop callstacks here. The ethernet IRQ has to put received telegrams into a queue which is polled in the mainloop. Ensure lwIP is *NEVER* called from an interrupt, e.g. some SPI IRQ wants to forward data to udp_send() or tcp_write()!

OS Mode

In OS mode, "raw" APIs AND Sequential-style APIs can be used. Sequential-style APIs are designed to be called from threads other than the TCPIP thread, so there is nothing to consider here. But "raw" APIs functions must *ONLY* be called from TCPIP thread. It is a common error to call these from other threads or from IRQ contexts. Ethernet RX needs to deliver incoming packets in the correct way by sending a message to TCPIP thread, this is implemented in tcpip_input(). Again, ensure lwIP is *NEVER* called from an interrupt, e.g. some SPI IRQ wants to forward data to udp_send() or tcp_write()!

1) tcpip_callback() can be used get called back from TCPIP thread, it is safe to call any "raw" APIs from there.

2) Use LWIP_TCPIP_CORE_LOCKING. All "raw" APIs functions can be called when lwIP core lock is acquired, see LOCK_TCPIP_C and UNLOCK_TCPIP_CORE(). These macros cannot be used in an interrupt context! Note the OS must correctly handle priority inversion for this.

Cache / DMA issues

DMA-capable ethernet hardware and zero-copy RX

lwIP changes the content of RECEIVED pbufs in the TCP code path. This implies one or more cacheline(s) of the RX pbuf become dirty and need to be flushed before the memory is handed over to the DMA ethernet hardware for the next telegram to be received. See http://lists.nongnu.org/archive/html/lwip-devel/2017-12/msg00070.html for a more detailed explanation. Also keep in mind the user application may also write into pbufs, so it is generally a bug not to flush the data cache before handing a buffer to DMA hardware.

DMA-capable ethernet hardware and cacheline alignment

Nice description about DMA capable hardware and buffer handling: http://www.pebblebay.com/a-guide-to-using-direct-memory-access-in-embedded-systems-part-two/ Read especially sections "Cache coherency" and "Buffer alignment".

1.6 Debugging memory pool sizes

If you have issues with lwIP and you are using memory pools, check that your pools are correctly sized. To debug pool sizes, #define LWIP_STATS and MEMP_STATS to 1. Check the global variable lwip_stats.memp[] using a debugger. If the "err" member of a pool is > 0, the pool may be too small for your application and you need to increase its size.

1.7 Reporting bugs

Please report bugs in the lwIP bug tracker at savannah. BEFORE submitting, please check if the bug has already been reported! https://savannah.nongnu.org/bugs/?group=lwip

1.8 Zero-copy RX

The following code is an example for zero-copy RX ethernet driver:

```
typedef struct my_custom_pbuf
{
   struct pbuf_custom p;
   void* dma_descriptor;
} my_custom_pbuf_t;

LWIP_MEMPOOL_DECLARE(RX_POOL, 10, sizeof(my_custom_pbuf_t), "Zero-copy RX PBUF  ↩
    pool");

void my_pbuf_free_custom(void* p)
{
  SYS_ARCH_DECL_PROTECT(old_level);

  my_custom_pbuf_t* my_puf = (my_custom_pbuf_t*)p;

  // invalidate data cache here - lwIP and/or application may have written into  ↩
     buffer!
  // (invalidate is faster than flushing, and no one needs the correct data in the ↩
      buffer)
  invalidate_cpu_cache(p->payload, p->tot_len);

  SYS_ARCH_PROTECT(old_level);
  free_rx_dma_descriptor(my_pbuf->dma_descriptor);
  LWIP_MEMPOOL_FREE(RX_POOL, my_pbuf);
  SYS_ARCH_UNPROTECT(old_level);
}

void eth_rx_irq()
{
  dma_descriptor*   dma_desc = get_RX_DMA_descriptor_from_ethernet();
  my_custom_pbuf_t* my_pbuf  = (my_custom_pbuf_t*)LWIP_MEMPOOL_ALLOC(RX_POOL);

  my_pbuf->p.custom_free_function = my_pbuf_free_custom;
  my_pbuf->dma_descriptor         = dma_desc;

  invalidate_cpu_cache(dma_desc->rx_data, dma_desc->rx_length);

  struct pbuf* p = pbuf_alloced_custom(PBUF_RAW,
     dma_desc->rx_length,
     PBUF_REF,
     &my_pbuf->p,
     dma_desc->rx_data,
     dma_desc->max_buffer_size);

  if(netif->input(p, netif) != ERR_OK) {
    pbuf_free(p);
  }
}
```

1.9 System initialization

A truly complete and generic sequence for initializing the lwIP stack cannot be given because it depends on additional initializations for your runtime environment (e.g. timers).

We can give you some idea on how to proceed when using the raw API. We assume a configuration using a single Ethernet netif and the UDP and TCP transport layers, IPv4 and the DHCP client.

Call these functions in the order of appearance:

- lwip_init(): Initialize the lwIP stack and all of its subsystems.
- netif_add(struct netif *netif, ...): Adds your network interface to the netif_list. Allocate a struct netif and pass a pointer to this structure as the first argument. Give pointers to cleared ip_addr structures when using DHCP, or fill them with sane numbers otherwise. The state pointer may be NULL.

  The init function pointer must point to a initialization function for your Ethernet netif interface. The following code illustrates its use.

```
err_t netif_if_init(struct netif *netif)
{
  u8_t i;

  for (i = 0; i < ETHARP_HWADDR_LEN; i++) {
    netif->hwaddr[i] = some_eth_addr[i];
  }
  init_my_eth_device();
  return ERR_OK;
}
```

For Ethernet drivers, the input function pointer must point to the lwIP function ethernet_input() declared in "netif/etharp.h". Other drivers must use ip_input() declared in "lwip/ip.h".

- netif_set_default(struct netif *netif) Registers the default network interface.
- netif_set_link_up(struct netif *netif) This is the hardware link state; e.g. whether cable is plugged for wired Ethernet interface. This function must be called even if you don't know the current state. Having link up and link down events is optional but DHCP and IPv6 discover benefit well from those events.

- netif_set_up(struct netif *netif) This is the administrative (= software) state of the netif, when the netif is fully configured this function must be called.
- dhcp_start(struct netif *netif) Creates a new DHCP client for this interface on the first call. You can peek in the netif->dhcp struct for the actual DHCP status.
- sys_check_timeouts() When the system is running, you have to periodically call sys_check_timeouts() which will handle all timers for all protocols in the stack; add this to your main loop or equivalent.

1.10 Multithreading

lwIP started targeting single-threaded environments. When adding multi- threading support, instead of making the core thread-safe, another approach was chosen: there is one main thread running the lwIP core (also known as the "tcpip_thread"). When running in a multithreaded environment, raw API functions MUST only be called from the core thread since raw API functions are not protected from concurrent access (aside from pbuf- and memory management functions). Application threads using the sequential- or socket API communicate with this main thread through message passing.

As such, the list of functions that may be called from other threads or an ISR is very limited! Only functions from these API header files are thread-safe:

- api.h
- netbuf.h
- netdb.h
- netifapi.h
- pppapi.h
- sockets.h
- sys.h

Additionally, memory (de-)allocation functions may be called from multiple threads (not ISR!) with NO_SYS=0 since they are protected by SYS_LIGHTWEIGHT_PROT and/or semaphores.

Netconn or Socket API functions are thread safe against the core thread but they are not reentrant at the control block granularity level. That is, a UDP or TCP control block must not be shared among multiple threads without proper locking.

If SYS_LIGHTWEIGHT_PROT is set to 1 and LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT is set to 1, pbuf_free() may also be called from another thread or an ISR (since only then, mem_free - for PBUF_RAM - may be called from an ISR: otherwise, the HEAP is only protected by semaphores).

How to get threading done right

It is strongly recommended to implement the LWIP_ASSERT_CORE_LOCKED() macro in an application that uses multithreading. lwIP code has several places where a check for a correct thread context is implemented which greatly helps the user to get threading done right. See the example sys_arch.c files in unix and Win32 port in the lwIP/contrib subdirectory.

In short: Copy the functions sys_mark_tcpip_thread() and sys_check_core_locking() to your port and modify them to work with your OS. Then let LWIP_ASSERT_CORE_LOCKED() and LWIP_MARK_TCPIP_THREAD() point to these functions.

If you use LWIP_TCPIP_CORE_LOCKING, you also need to copy and adapt the functions sys_lock_tcpip_core() and sys_unlock_tcpip_ Let LOCK_TCPIP_CORE() and UNLOCK_TCPIP_CORE() point to these functions.

1.11 Optimization hints

The first thing you want to optimize is the lwip_standard_checksum() routine from src/core/inet.c. You can override this standard function with the #define LWIP_CHKSUM your_checksum_routine().

There are C examples given in inet.c or you might want to craft an assembly function for this. RFC1071 is a good introduction to this subject.

Other significant improvements can be made by supplying assembly or inline replacements for htons() and htonl() if you're using a little-endian architecture. #define lwip_htons(x) your_htons() #define lwip_htonl(x) your_htonl() If you #define them to htons() and htonl(), you should #define LWIP_DONT_PROVIDE_BYTEORDER_FUNCTIONS to prevent lwIP from defining htonx / ntohx compatibility macros.

Check your network interface driver if it reads at a higher speed than the maximum wire-speed. If the hardware isn't serviced frequently and fast enough buffer overflows are likely to occur.

E.g. when using the cs8900 driver, call cs8900if_service(ethif) as frequently as possible. When using an RTOS let the cs8900 interrupt wake a high priority task that services your driver using a binary semaphore or event flag. Some drivers might allow additional tuning to match your application and network.

For a production release it is recommended to set LWIP_STATS to 0. Note that speed performance isn't influenced much by simply setting high values to the memory options.

1.12 Deprecated List

Global dhcp_release (struct netif *netif) Use dhcp_release_and_stop() instead.

Global dhcp_stop (struct netif *netif) Use dhcp_release_and_stop() instead.

Global ip4_addr_cmp (addr1, addr2) Renamed to ip4_addr_eq

Global ip4_addr_netcmp (addr1, addr2, mask) Renamed to ip4_addr_net_eq

Global ip6_addr_cmp (addr1, addr2) Renamed to ip6_addr_eq

Global ip6_addr_cmp_packed (ip6addr, paddr, zone_idx) Renamed to ip6_addr_packed_eq

Global ip6_addr_cmp_zone (addr1, addr2) Renamed to ip6_addr_zone_eq

Global ip6_addr_cmp_zoneless (addr1, addr2) Renamed to ip6_addr_zoneless_eq

Global ip6_addr_netcmp (addr1, addr2) Renamed to ip6_addr_net_eq

Global ip6_addr_netcmp_zoneless (addr1, addr2) Renamed to ip6_addr_net_zoneless_eq

Global ip_addr_cmp (addr1, addr2) Renamed to ip_addr_eq

Global ip_addr_cmp_zoneless (addr1, addr2) Renamed to ip_addr_zoneless_eq

Global ip_addr_netcmp (addr1, addr2, mask) Renamed to ip_addr_net_eq

Global netifapi_dhcp_release (n) Use netifapi_dhcp_release_and_stop() instead.

Global netifapi_dhcp_stop (n) Use netifapi_dhcp_release_and_stop() instead.

Global tcpip_callback_with_block (function, ctx, block) use tcpip_try_callback() or tcpip_callback() instead

Chapter 2

Topic Documentation

2.1 lwIP

2.1.1 Modules

- Mainloop mode ("NO_SYS")
- OS mode (TCPIP thread)
- Porting (system abstraction layer)
- Version
- Options (lwipopts.h)

2.1.2 Detailed Description

2.1.3 Mainloop mode ("NO_SYS")

2.1.3.1 Functions

- void lwip_init (void)
- err_t ip_input (struct pbuf *p, struct netif *inp)
- err_t netif_input (struct pbuf *p, struct netif *inp)
- void sys_check_timeouts (void)
- err_t ethernet_input (struct pbuf *p, struct netif *netif)

2.1.3.2 Detailed Description

Use this mode if you do not run an OS on your system. #define NO_SYS to 1. Feed incoming packets to netif->input(pbuf, netif) function from mainloop, *not from interrupt context*. You can allocate a Packet buffers (PBUF) in interrupt context and put them into a queue which is processed from mainloop. Call sys_check_timeouts() periodically in the mainloop. Porting: implement all functions in Time, Critical sections and Compiler/platform abstraction. You can only use "raw" APIs in this mode. Sample code:

```
void
eth_mac_irq()
{
  /* Service MAC IRQ here */

  /* Allocate pbuf from pool (avoid using heap in interrupts) */
  struct pbuf* p = pbuf_alloc(PBUF_RAW, eth_data_count, PBUF_POOL);

  if(p != NULL) {
    /* Copy ethernet frame into pbuf */
    pbuf_take(p, eth_data, eth_data_count);

    /* Put in a queue which is processed in main loop */
    if(!queue_try_put(&queue, p)) {
      /* queue is full -> packet loss */
      pbuf_free(p);
    }
  }
}

static err_t
netif_output(struct netif *netif, struct pbuf *p)
{
  LINK_STATS_INC(link.xmit);

  /* Update SNMP stats (only if you use SNMP) */
  MIB2_STATS_NETIF_ADD(netif, ifoutoctets, p->tot_len);
  int unicast = ((p->payload[0] & 0x01) == 0);
  if (unicast) {
    MIB2_STATS_NETIF_INC(netif, ifoutucastpkts);
  } else {
    MIB2_STATS_NETIF_INC(netif, ifoutnucastpkts);
  }

  lock_interrupts();
  pbuf_copy_partial(p, mac_send_buffer, p->tot_len, 0);
  /* Start MAC transmit here */
  unlock_interrupts();

  return ERR_OK;
}

static void
netif_status_callback(struct netif *netif)
{
  printf("netif status changed %s\n", ip4addr_ntoa(netif_ip4_addr(netif)));
}

static err_t
netif_init(struct netif *netif)
{
  netif->linkoutput = netif_output;
  netif->output     = etharp_output;
  netif->output_ip6 = ethip6_output;
  netif->mtu        = ETHERNET_MTU;
  netif->flags      = NETIF_FLAG_BROADCAST | NETIF_FLAG_ETHARP |  ↩
```

```
      NETIF_FLAG_ETHERNET | NETIF_FLAG_IGMP | NETIF_FLAG_MLD6;
  MIB2_INIT_NETIF(netif, snmp_ifType_ethernet_csmacd, 100000000);

  SMEMCPY(netif->hwaddr, your_mac_address_goes_here, ETH_HWADDR_LEN);
  netif->hwaddr_len = ETH_HWADDR_LEN;

  return ERR_OK;
}

void
main(void)
{
  struct netif netif;

  lwip_init();

  netif_add(&netif, IP4_ADDR_ANY, IP4_ADDR_ANY, IP4_ADDR_ANY, NULL, netif_init,  ↩
      netif_input);
  netif.name[0] = 'e';
  netif.name[1] = '0';
  netif_create_ip6_linklocal_address(&netif, 1);
  netif_set_status_callback(&netif, netif_status_callback);
  netif_set_default(&netif);
  netif_set_up(&netif);

  /* Start DHCP and HTTPD */
  dhcp_start(&netif );
  httpd_init();

  while(1) {
    /* Check link state, e.g. via MDIO communication with PHY */
    if(link_state_changed()) {
      if(link_is_up()) {
        netif_set_link_up(&netif);
      } else {
        netif_set_link_down(&netif);
      }
    }

    /* Check for received frames, feed them to lwIP */
    lock_interrupts();
    struct pbuf* p = queue_try_get(&queue);
    unlock_interrupts();

    if(p != NULL) {
      LINK_STATS_INC(link.recv);

      /* Update SNMP stats (only if you use SNMP) */
      MIB2_STATS_NETIF_ADD(netif, ifinoctets, p->tot_len);
      int unicast = ((p->payload[0] & 0x01) == 0);
      if (unicast) {
        MIB2_STATS_NETIF_INC(netif, ifinucastpkts);
      } else {
        MIB2_STATS_NETIF_INC(netif, ifinnucastpkts);
      }

      if(netif.input(p, &netif) != ERR_OK) {
        pbuf_free(p);
      }
    }

    /* Cyclic lwIP timers check */
    sys_check_timeouts();

    /* your application goes here */
  }
}
```

2.1.3.3 Function Documentation

2.1.3.3.1 ethernet_input()

`err_t ethernet_input (struct pbuf * p, struct netif * netif)`

Process received ethernet frames. Using this function instead of directly calling ip_input and passing ARP frames through etharp in ethernetif_input, the ARP cache is protected from concurrent access. Don't call directly, pass to netif_add() and call netif->input().

Parameters

p	the received packet, p->payload pointing to the ethernet header
netif	the network interface on which the packet was received

See also LWIP_HOOK_UNKNOWN_ETH_PROTOCOL

ETHARP_SUPPORT_VLAN

LWIP_HOOK_VLAN_CHECK

2.1.3.3.2 ip_input()

`err_t ip_input (struct pbuf * p, struct netif * inp)`

If both IP versions are enabled, this function can dispatch packets to the correct one. Don't call directly, pass to netif_add() and call netif->input().

2.1.3.3.3 lwip_init()

`void lwip_init (void )`

Initialize all modules. Use this in NO_SYS mode. Use tcpip_init() otherwise.

2.1.3.3.4 netif_input()

`err_t netif_input (struct pbuf * p, struct netif * inp)`

Forwards a received packet for input processing with ethernet_input() or ip_input() depending on netif flags. Don't call directly, pass to netif_add() and call netif->input(). Only works if the netif driver correctly sets NETIF_FLAG_ETHARP and/or NETIF_FLAG_ETHERNET flag!

2.1.3.3.5 sys_check_timeouts()

```
void sys_check_timeouts (void )
```

Handle timeouts for NO_SYS==1 (i.e. without using tcpip_thread/sys_timeouts_mbox_fetch(). Uses sys_now() to call timeout handler functions when timeouts expire.

Must be called periodically from your main loop.

2.1.4 OS mode (TCPIP thread)

2.1.4.1 Macros

- #define tcpip_callback_with_block(function, ctx, block) ((block != 0)? tcpip_callback(function, ctx) : tcpip_try_callback(function, ctx))

2.1.4.2 Functions

- err_t tcpip_input (struct pbuf *p, struct netif *inp)
- err_t tcpip_callback (tcpip_callback_fn function, void *ctx)
- err_t tcpip_try_callback (tcpip_callback_fn function, void *ctx)
- struct tcpip_callback_msg * tcpip_callbackmsg_new (tcpip_callback_fn function, void *ctx)
- void tcpip_callbackmsg_delete (struct tcpip_callback_msg *msg)
- err_t tcpip_callbackmsg_trycallback (struct tcpip_callback_msg *msg)
- err_t tcpip_callbackmsg_trycallback_fromisr (struct tcpip_callback_msg *msg)
- void tcpip_init (tcpip_init_done_fn initfunc, void *arg)

2.1.4.3 Detailed Description

Use this mode if you run an OS on your system. It is recommended to use an RTOS that correctly handles priority inversion and to use LWIP_TCPIP_CORE_LOCKING. Porting: implement all functions in Porting (system abstraction layer). You can use "raw" APIs together with tcpip_callback, and all Sequential-style APIs.

2.1.4.4 Macro Definition Documentation

2.1.4.4.1 tcpip_callback_with_block

```
#define tcpip_callback_with_block( function, ctx, block)   ((block != 0)? tcpip_callback(
ctx) : tcpip_try_callback(function, ctx))
```

Deprecated use tcpip_try_callback() or tcpip_callback() instead

2.1.4.5 Function Documentation

2.1.4.5.1 tcpip_callback()

```
err_t tcpip_callback (tcpip_callback_fn function, void * ctx)
```

Call a specific function in the thread context of tcpip_thread for easy access synchronization. A function called in that way may access lwIP core code without fearing concurrent access. Blocks until the request is posted. Must not be called from interrupt context!

Parameters

Returns ERR_OK if the function was called, another err_t if not

See also tcpip_try_callback

function	the function to call
ctx	parameter passed to f

2.1.4.5.2 tcpip_callbackmsg_delete()

```
void tcpip_callbackmsg_delete (struct tcpip_callback_msg * msg)
```

Free a callback message allocated by tcpip_callbackmsg_new().

Parameters

msg	the message to free

See also tcpip_callbackmsg_new()

2.1.4.5.3 tcpip_callbackmsg_new()

```
struct tcpip_callback_msg * tcpip_callbackmsg_new (tcpip_callback_fn function, void * ctx)
```

Allocate a structure for a static callback message and initialize it. The message has a special type such that lwIP never frees it. This is intended to be used to send "static" messages from interrupt context, e.g. the message is allocated once and posted several times from an IRQ using tcpip_callbackmsg_trycallback(). Example usage: Trigger execution of an ethernet IRQ DPC routine in lwIP thread context.

Parameters

function	the function to call
ctx	parameter passed to function

Returns a struct pointer to pass to tcpip_callbackmsg_trycallback().

See also tcpip_callbackmsg_trycallback()

tcpip_callbackmsg_delete()

2.1.4.5.4 tcpip_callbackmsg_trycallback()

```
err_t tcpip_callbackmsg_trycallback (struct tcpip_callback_msg * msg)
```

Try to post a callback-message to the tcpip_thread tcpip_mbox.

Parameters

Returns sys_mbox_trypost() return code

See also tcpip_callbackmsg_new()

2.1.4.5.5 tcpip_callbackmsg_trycallback_fromisr()

```
err_t tcpip_callbackmsg_trycallback_fromisr (struct tcpip_callback_msg * msg)
```

Try to post a callback-message to the tcpip_thread mbox. Same as tcpip_callbackmsg_trycallback but calls sys_mbox_trypost_fromisr(), mainly to help FreeRTOS, where calls differ between task level and ISR level.

Parameters

Returns sys_mbox_trypost_fromisr() return code (without change, so this knowledge can be used to e.g. propagate "bool needs_scheduling")

See also tcpip_callbackmsg_new()

msg	pointer to the message to post

msg	pointer to the message to post

2.1.4.5.6 tcpip_init()

```
void tcpip_init (tcpip_init_done_fn initfunc, void * arg)
```

Initialize this module:

- initialize all sub modules
- start the tcpip_thread

Parameters

initfunc	a function to call when tcpip_thread is running and finished initializing
arg	argument to pass to initfunc

2.1.4.5.7 tcpip_input()

```
err_t tcpip_input (struct pbuf * p, struct netif * inp)
```

Pass a received packet to tcpip_thread for input processing with ethernet_input or ip_input. Don't call directly, pass to netif_add() and call netif->input().

Parameters

2.1.4.5.8 tcpip_try_callback()

```
err_t tcpip_try_callback (tcpip_callback_fn function, void * ctx)
```

Call a specific function in the thread context of tcpip_thread for easy access synchronization. A function called in that way may access lwIP core code without fearing concurrent access. Does NOT block when the request cannot be posted because the tcpip_mbox is full, but returns ERR_MEM instead. Can be called from interrupt context.

Parameters

Returns ERR_OK if the function was called, another err_t if not

See also tcpip_callback

2.1.5 Porting (system abstraction layer)

2.1.5.1 Modules

- Non-standard functions
- OS abstraction layer
- Time
- Critical sections
- Compiler/platform abstraction
- Performance measurement

p	the received packet, p->payload pointing to the Ethernet header or to an IP header (if inp doesn't have NETIF_FLAG_ETHARP or NETIF_FLAG_ETHERNET flags)
inp	the network interface on which the packet was received

function	the function to call
ctx	parameter passed to f

2.1.5.2 Detailed Description

2.1.5.3 Non-standard functions

2.1.5.3.1 Functions

- char * lwip_strnstr (const char *buffer, const char *token, size_t n)
- char * lwip_strnistr (const char *buffer, const char *token, size_t n)
- int lwip_stricmp (const char *str1, const char *str2)
- int lwip_strnicmp (const char *str1, const char *str2, size_t len)
- void lwip_itoa (char *result, size_t bufsize, int number)
- int lwip_memcmp_consttime (const void *s1, const void *s2, size_t len)

2.1.5.3.2 Detailed Description

lwIP provides default implementations for non-standard functions. These can be mapped to OS functions to reduce code footprint if desired. All defines related to this section must not be placed in lwipopts.h, but in arch/cc.h! These options cannot be #defined in lwipopts.h since they are not options of lwIP itself, but options of the lwIP port to your system.

2.1.5.3.3 Function Documentation

2.1.5.3.3.1 lwip_itoa()

```
void lwip_itoa (char * result, size_t bufsize, int number)
```

lwIP default implementation for itoa() non-standard function. This can be #defined to itoa() or snprintf(result, bufsize, "%d", number) depending on your platform port.

2.1.5.3.3.2 lwip_memcmp_consttime()

```
int lwip_memcmp_consttime (const void * s1, const void * s2, size_t len)
```

The goal of this function is to compare memory with constant runtime in order to prevent timing attacks to various parts in the stack. To do that, in contrast to memcmp(), it only returns: 0: equal != 0: not equal

2.1.5.3.3.3 lwip_stricmp()

```
int lwip_stricmp (const char * str1, const char * str2)
```

lwIP default implementation for stricmp() non-standard function. This can be #defined to stricmp() depending on your platform port.

2.1.5.3.3.4 lwip_strnicmp()

```
int lwip_strnicmp (const char * str1, const char * str2, size_t len)
```

lwIP default implementation for strnicmp() non-standard function. This can be #defined to strnicmp() depending on your platform port.

2.1.5.3.3.5 lwip_strnistr()

```
char * lwip_strnistr (const char * buffer, const char * token, size_t n)
```

lwIP default implementation for strnistr() non-standard function. This can be #defined to strnistr() depending on your platform port.

2.1.5.3.3.6 lwip_strnstr()

```
char * lwip_strnstr (const char * buffer, const char * token, size_t n)
```

lwIP default implementation for strnstr() non-standard function. This can be #defined to strnstr() depending on your platform port.

2.1.5.4 OS abstraction layer

2.1.5.4.1 Modules

- Semaphores
- Mutexes
- Mailboxes
- Misc

2.1.5.4.2 Detailed Description

No need to implement functions in this section in NO_SYS mode. The OS-specific code should be implemented in arch/sys_arch.h and sys_arch.c of your port.

The operating system emulation layer provides a common interface between the lwIP code and the underlying operating system kernel. The general idea is that porting lwIP to new architectures requires only small changes to a few header files and a new sys_arch implementation. It is also possible to do a sys_arch implementation that does not rely on any underlying operating system.

The sys_arch provides semaphores, mailboxes and mutexes to lwIP. For the full lwIP functionality, multiple threads support can be implemented in the sys_arch, but this is not required for the basic lwIP functionality. Timer scheduling is implemented in lwIP, but can be implemented by the sys_arch port (LWIP_TIMERS_CUSTOM==1).

In addition to the source file providing the functionality of sys_arch, the OS emulation layer must provide several header files defining macros used throughout lwip. The files required and the macros they must define are listed below the sys_arch description.

Since lwIP 1.4.0, semaphore, mutexes and mailbox functions are prototyped in a way that allows both using pointers or actual OS structures to be used. This way, memory required for such types can be either allocated in place (globally or on the stack) or on the heap (allocated internally in the "*_new()" functions).

Note:

Be careful with using mem_malloc() in sys_arch. When malloc() refers to mem_malloc() you can run into a circular function call problem. In mem.c mem_init() tries to allocate a semaphore using mem_malloc, which of course can't be performed when sys_arch uses mem_malloc.

2.1.5.4.3 Semaphores

2.1.5.4.3.1 Functions

- err_t sys_sem_new (sys_sem_t *sem, u8_t count)
- void sys_sem_signal (sys_sem_t *sem)
- u32_t sys_arch_sem_wait (sys_sem_t *sem, u32_t timeout)
- void sys_sem_free (sys_sem_t *sem)
- int sys_sem_valid (sys_sem_t *sem)
- void sys_sem_set_invalid (sys_sem_t *sem)

2.1.5.4.3.2 Detailed Description

Semaphores can be either counting or binary - lwIP works with both kinds. Semaphores are represented by the type "sys_sem_t" which is typedef'd in the sys_arch.h file. Mailboxes are equivalently represented by the type "sys_mbox_t". Mutexes are represented by the type "sys_mutex_t". lwIP does not place any restrictions on how these types are represented internally.

2.1.5.4.3.3 Function Documentation

2.1.5.4.3.4 sys_arch_sem_wait()

```
u32_t sys_arch_sem_wait (sys_sem_t * sem, u32_t timeout)
```

Blocks the thread while waiting for the semaphore to be signaled. If the "timeout" argument is non-zero, the thread should only be blocked for the specified time (measured in milliseconds). If the "timeout" argument is zero, the thread should be blocked until the semaphore is signalled.

The return value is SYS_ARCH_TIMEOUT if the semaphore wasn't signaled within the specified time or any other value if it was signaled (with or without waiting). Notice that lwIP implements a function with a similar name, sys_sem_wait(), that uses the sys_arch_sem_wait() function.

Parameters

sem	the semaphore to wait for
timeout	timeout in milliseconds to wait (0 = wait forever)

Returns SYS_ARCH_TIMEOUT on timeout, any other value on success

2.1.5.4.3.5 sys_sem_free()

```
void sys_sem_free (sys_sem_t * sem)
```

Deallocates a semaphore.

Parameters

sem	semaphore to delete

2.1.5.4.3.6 sys_sem_new()

```
err_t sys_sem_new (sys_sem_t * sem, u8_t count)
```

Create a new semaphore Creates a new semaphore. The semaphore is allocated to the memory that 'sem' points to (which can be both a pointer or the actual OS structure). The "count" argument specifies the initial state of the semaphore (which is either 0 or 1). If the semaphore has been created, ERR_OK should be returned. Returning any other error will provide a hint what went wrong, but except for assertions, no real error handling is implemented.

Parameters

sem	pointer to the semaphore to create
count	initial count of the semaphore

Returns ERR_OK if successful, another err_t otherwise

2.1.5.4.3.7 sys_sem_set_invalid()

```
void sys_sem_set_invalid (sys_sem_t * sem)
```

Invalidate a semaphore so that sys_sem_valid() returns 0. ATTENTION: This does NOT mean that the semaphore shall be deallocated: sys_sem_free() is always called before calling this function! This may also be a define, in which case the function is not prototyped.

2.1.5.4.3.8 sys_sem_signal()

```
void sys_sem_signal (sys_sem_t * sem)
```

Signals a semaphore

Parameters

sem	the semaphore to signal

2.1.5.4.3.9 sys_sem_valid()

```
int sys_sem_valid (sys_sem_t * sem)
```

Returns 1 if the semaphore is valid, 0 if it is not valid. When using pointers, a simple way is to check the pointer for != NULL. When directly using OS structures, implementing this may be more complex. This may also be a define, in which case the function is not prototyped.

2.1.5.4.4 Mutexes

2.1.5.4.4.1 Functions

- err_t sys_mutex_new (sys_mutex_t *mutex)
- void sys_mutex_lock (sys_mutex_t *mutex)
- void sys_mutex_unlock (sys_mutex_t *mutex)
- void sys_mutex_free (sys_mutex_t *mutex)
- int sys_mutex_valid (sys_mutex_t *mutex)
- void sys_mutex_set_invalid (sys_mutex_t *mutex)

2.1.5.4.4.2 Detailed Description

Mutexes are recommended to correctly handle priority inversion, especially if you use LWIP_CORE_LOCKING .

2.1.5.4.4.3 Function Documentation

2.1.5.4.4.4 sys_mutex_free()

```
void sys_mutex_free (sys_mutex_t * mutex)
```

Deallocates a mutex.

Parameters

mutex	the mutex to delete

2.1.5.4.4.5 sys_mutex_lock()

```
void sys_mutex_lock (sys_mutex_t * mutex)
```

Blocks the thread until the mutex can be grabbed.

Parameters

mutex	the mutex to lock

2.1.5.4.4.6 sys_mutex_new()

```
err_t sys_mutex_new (sys_mutex_t * mutex)
```

Create a new mutex. Note that mutexes are expected to not be taken recursively by the lwIP code, so both implementation types (recursive or non-recursive) should work. The mutex is allocated to the memory that 'mutex' points to (which can be both a pointer or the actual OS structure). If the mutex has been created, ERR_OK should be returned. Returning any other error will provide a hint what went wrong, but except for assertions, no real error handling is implemented.

Parameters

mutex	pointer to the mutex to create

Returns ERR_OK if successful, another err_t otherwise

2.1.5.4.4.7 sys_mutex_set_invalid()

```
void sys_mutex_set_invalid (sys_mutex_t * mutex)
```

Invalidate a mutex so that sys_mutex_valid() returns 0. ATTENTION: This does NOT mean that the mutex shall be deallocated: sys_mutex_free() is always called before calling this function! This may also be a define, in which case the function is not prototyped.

2.1.5.4.4.8 sys_mutex_unlock()

```
void sys_mutex_unlock (sys_mutex_t * mutex)
```

Releases the mutex previously locked through 'sys_mutex_lock()'.

Parameters

mutex	the mutex to unlock

2.1.5.4.4.9 sys_mutex_valid()

```
int sys_mutex_valid (sys_mutex_t * mutex)
```

Returns 1 if the mutes is valid, 0 if it is not valid. When using pointers, a simple way is to check the pointer for != NULL. When directly using OS structures, implementing this may be more complex. This may also be a define, in which case the function is not prototyped.

2.1.5.4.5 Mailboxes

2.1.5.4.5.1 Functions

- err_t sys_mbox_new (sys_mbox_t *mbox, int size)
- void sys_mbox_post (sys_mbox_t *mbox, void *msg)
- err_t sys_mbox_trypost (sys_mbox_t *mbox, void *msg)
- err_t sys_mbox_trypost_fromisr (sys_mbox_t *mbox, void *msg)
- u32_t sys_arch_mbox_fetch (sys_mbox_t *mbox, void **msg, u32_t timeout)
- u32_t sys_arch_mbox_tryfetch (sys_mbox_t *mbox, void **msg)
- void sys_mbox_free (sys_mbox_t *mbox)
- int sys_mbox_valid (sys_mbox_t *mbox)
- void sys_mbox_set_invalid (sys_mbox_t *mbox)

2.1.5.4.5.2 Detailed Description

Mailboxes should be implemented as a queue which allows multiple messages to be posted (implementing as a rendez-vous point where only one message can be posted at a time can have a highly negative impact on performance). A message in a mailbox is just a pointer, nothing more.

2.1.5.4.5.3 Function Documentation

2.1.5.4.5.4 sys_arch_mbox_fetch()

```
u32_t sys_arch_mbox_fetch (sys_mbox_t * mbox, void ** msg, u32_t timeout)
```

Blocks the thread until a message arrives in the mailbox, but does not block the thread longer than "timeout" milliseconds (similar to the sys_arch_sem_wait() function). If "timeout" is 0, the thread should be blocked until a message arrives. The "msg" argument is a result parameter that is set by the function (i.e., by doing "*msg = ptr"). The "msg" parameter maybe NULL to indicate that the message should be dropped. The return values are the same as for the sys_arch_sem_wait() function: SYS_ARCH_TIMEOUT if there was a timeout, any other value if a messages is received.

Note that a function with a similar name, sys_mbox_fetch(), is implemented by lwIP.

Parameters

mbox	mbox to get a message from
msg	pointer where the message is stored
timeout	maximum time (in milliseconds) to wait for a message (0 = wait forever)

Returns SYS_ARCH_TIMEOUT on timeout, any other value if a message has been received

2.1.5.4.5.5 sys_arch_mbox_tryfetch()

`u32_t sys_arch_mbox_tryfetch (sys_mbox_t * mbox, void ** msg)`

This is similar to sys_arch_mbox_fetch, however if a message is not present in the mailbox, it immediately returns with the code SYS_MBOX_EMPTY. On success 0 is returned. To allow for efficient implementations, this can be defined as a function-like macro in sys_arch.h instead of a normal function. For example, a naive implementation could be: #define sys_arch_mbox_tryfetch(mbox,msg) sys_arch_mbox_fetch(mbox,msg,1) although this would introduce unnecessary delays.

Parameters

mbox	mbox to get a message from
msg	pointer where the message is stored

Returns 0 (milliseconds) if a message has been received or SYS_MBOX_EMPTY if the mailbox is empty

2.1.5.4.5.6 sys_mbox_free()

`void sys_mbox_free (sys_mbox_t * mbox)`

Deallocates a mailbox. If there are messages still present in the mailbox when the mailbox is deallocated, it is an indication of a programming error in lwIP and the developer should be notified.

Parameters

mbox	mbox to delete

2.1.5.4.5.7 sys_mbox_new()

`err_t sys_mbox_new (sys_mbox_t * mbox, int size)`

Creates an empty mailbox for maximum "size" elements. Elements stored in mailboxes are pointers. You have to define macros "_MBOX_SIZE" in your lwipopts.h, or ignore this parameter in your implementation and use a default size. If the mailbox has been created, ERR_OK should be returned. Returning any other error will provide a hint what went wrong, but except for assertions, no real error handling is implemented.

Parameters

mbox	pointer to the mbox to create
size	(minimum) number of messages in this mbox

Returns ERR_OK if successful, another err_t otherwise

2.1.5.4.5.8 sys_mbox_post()

`void sys_mbox_post (sys_mbox_t * mbox, void * msg)`

Post a message to an mbox - may not fail -> blocks if full, only to be used from tasks NOT from ISR!

Parameters

2.1.5.4.5.9 sys_mbox_set_invalid()

`void sys_mbox_set_invalid (sys_mbox_t * mbox)`

Invalidate a mailbox so that sys_mbox_valid() returns 0. ATTENTION: This does NOT mean that the mailbox shall be deallocated: sys_mbox_free() is always called before calling this function! This may also be a define, in which case the function is not prototyped.

mbox	mbox to posts the message
msg	message to post (ATTENTION: can be NULL)

2.1.5.4.5.10 sys_mbox_trypost()

```
err_t sys_mbox_trypost (sys_mbox_t * mbox, void * msg)
```

Try to post a message to an mbox - may fail if full. Can be used from ISR (if the sys arch layer allows this). Returns ERR_MEM if it is full, else, ERR_OK if the "msg" is posted.

Parameters

mbox	mbox to posts the message
msg	message to post (ATTENTION: can be NULL)

2.1.5.4.5.11 sys_mbox_trypost_fromisr()

```
err_t sys_mbox_trypost_fromisr (sys_mbox_t * mbox, void * msg)
```

Try to post a message to an mbox - may fail if full. To be be used from ISR. Returns ERR_MEM if it is full, else, ERR_OK if the "msg" is posted.

Parameters

mbox	mbox to posts the message
msg	message to post (ATTENTION: can be NULL)

2.1.5.4.5.12 sys_mbox_valid()

```
int sys_mbox_valid (sys_mbox_t * mbox)
```

Returns 1 if the mailbox is valid, 0 if it is not valid. When using pointers, a simple way is to check the pointer for != NULL. When directly using OS structures, implementing this may be more complex. This may also be a define, in which case the function is not prototyped.

2.1.5.4.6 Misc

2.1.5.4.6.1 Functions

- void sys_msleep (u32_t ms)
- sys_thread_t sys_thread_new (const char *name, lwip_thread_fn thread, void *arg, int stacksize, int prio)
- void sys_init (void)

2.1.5.4.6.2 Detailed Description

2.1.5.4.6.3 Function Documentation

2.1.5.4.6.4 sys_init()

```
void sys_init (void )
```

sys_init() must be called before anything else. Initialize the sys_arch layer.

2.1.5.4.6.5 sys_msleep()

```
void sys_msleep (u32_t ms)
```

Sleep for specified number of ms

Sleep for some ms. Timeouts are NOT processed while sleeping.

Parameters

ms	number of milliseconds to sleep

2.1.5.4.6.6 sys_thread_new()

```
sys_thread_t sys_thread_new (const char * name, lwip_thread_fn thread, void * arg, int
stacksize, int prio)
```

The only thread function: Starts a new thread named "name" with priority "prio" that will begin its execution in the function "thread()". The "arg" argument will be passed as an argument to the thread() function. The stack size to used for this thread is the "stacksize" parameter. The id of the new thread is returned. Both the id and the priority are system dependent. ATTENTION: although this function returns a value, it MUST NOT FAIL (ports have to assert this!)

Parameters

name	human-readable name for the thread (used for debugging purposes)
thread	thread-function
arg	parameter passed to ’thread’
stacksize	stack size in bytes for the new thread (may be ignored by ports)
prio	priority of the new thread (may be ignored by ports)

2.1.5.5 Time

2.1.5.5.1 Functions

- u32_t sys_now (void)

2.1.5.5.2 Detailed Description

2.1.5.5.3 Function Documentation

2.1.5.5.3.1 sys_now()

```
u32_t sys_now (void )
```

Returns the current time in milliseconds, may be the same as sys_jiffies or at least based on it. Don’t care for wraparound, this is only used for time diffs. Not implementing this function means you cannot use some modules (e.g. TCP timestamps, internal timeouts for NO_SYS==1).

2.1.5.6 Critical sections

2.1.5.6.1 Macros

- #define SYS_ARCH_DECL_PROTECT(lev)   sys_prot_t lev
- #define SYS_ARCH_PROTECT(lev)   lev = sys_arch_protect()
- #define SYS_ARCH_UNPROTECT(lev)   sys_arch_unprotect(lev)

2.1.5.6.2 Detailed Description

Used to protect short regions of code against concurrent access.

- Your system is a bare-metal system (probably with an RTOS) and interrupts are under your control: Implement this as LockInterrupts() / UnlockInterrupts()
- Your system uses an RTOS with deferred interrupt handling from a worker thread: Implement as a global mutex or lock/unlock scheduler
- Your system uses a high-level OS with e.g. POSIX signals: Implement as a global mutex

2.1.5.6.3 Macro Definition Documentation

2.1.5.6.3.1 SYS_ARCH_DECL_PROTECT

```
#define SYS_ARCH_DECL_PROTECT( lev)   sys_prot_t lev
```

SYS_LIGHTWEIGHT_PROT define SYS_LIGHTWEIGHT_PROT in lwipopts.h if you want inter-task protection for certain critical regions during buffer allocation, deallocation and memory allocation and deallocation.

SYS_ARCH_DECL_PROTECT declare a protection variable. This macro will default to defining a variable of type sys_prot_t. If a particular port needs a different implementation, then this macro may be defined in sys_arch.h.

2.1.5.6.3.2 SYS_ARCH_PROTECT

```
#define SYS_ARCH_PROTECT( lev)   lev = sys_arch_protect()
```

SYS_ARCH_PROTECT Perform a "fast" protect. This could be implemented by disabling interrupts for an embedded system or by using a semaphore or mutex. The implementation should allow calling SYS_ARCH_PROTECT when already protected. The old protection level is returned in the variable "lev". This macro will default to calling the sys_arch_protect() function which should be implemented in sys_arch.c. If a particular port needs a different implementation, then this macro may be defined in sys_arch.h

2.1.5.6.3.3 SYS_ARCH_UNPROTECT

```
#define SYS_ARCH_UNPROTECT( lev)   sys_arch_unprotect(lev)
```

SYS_ARCH_UNPROTECT Perform a "fast" set of the protection level to "lev". This could be implemented by setting the interrupt level to "lev" within the MACRO or by using a semaphore or mutex. This macro will default to calling the sys_arch_unprotect() function which should be implemented in sys_arch.c. If a particular port needs a different implementation, then this macro may be defined in sys_arch.h

2.1.5.7 Compiler/platform abstraction

2.1.5.7.1 Macros

- #define BYTE_ORDER   LITTLE_ENDIAN
- #define LWIP_RAND()   ((u32_t)rand())
- #define LWIP_PLATFORM_DIAG(x)   do {printf x;} while(0)
- #define LWIP_PLATFORM_ASSERT(x)
- #define LWIP_NO_STDDEF_H   0
- #define LWIP_NO_STDINT_H   0
- #define LWIP_NO_INTTYPES_H   0

- #define LWIP_NO_LIMITS_H 0
- #define LWIP_NO_CTYPE_H 0
- #define LWIP_CONST_CAST(target_type, val) ((target_type)((ptrdiff_t)val))
- #define LWIP_ALIGNMENT_CAST(target_type, val) LWIP_CONST_CAST(target_type, val)
- #define LWIP_PTR_NUMERIC_CAST(target_type, val) LWIP_CONST_CAST(target_type, val)
- #define LWIP_PACKED_CAST(target_type, val) LWIP_CONST_CAST(target_type, val)
- #define LWIP_DECLARE_MEMORY_ALIGNED(variable_name, size) u8_t variable_name[LWIP_MEM_ALIGN_BUFFER(size)
- #define LWIP_MEM_ALIGN_SIZE(size) (((size) + MEM_ALIGNMENT - 1U) & ~(MEM_ALIGNMENT-1U))
- #define LWIP_MEM_ALIGN_BUFFER(size) (((size) + MEM_ALIGNMENT - 1U))
- #define LWIP_MEM_ALIGN(addr) ((void *)(((mem_ptr_t)(addr) + MEM_ALIGNMENT - 1) & ~(mem_ptr_t)(MEM_ALIGNMEN
  1)))
- #define PACK_STRUCT_BEGIN
- #define PACK_STRUCT_END
- #define PACK_STRUCT_STRUCT
- #define PACK_STRUCT_FIELD(x) x
- #define PACK_STRUCT_FLD_8(x) PACK_STRUCT_FIELD(x)
- #define PACK_STRUCT_FLD_S(x) PACK_STRUCT_FIELD(x)
- #define PACK_STRUCT_USE_INCLUDES
- #define LWIP_UNUSED_ARG(x) (void)x
- #define LWIP_PROVIDE_ERRNO

2.1.5.7.2 Detailed Description

All defines related to this section must not be placed in lwipopts.h, but in arch/cc.h! If the compiler does not provide memset() this file must include a definition of it, or include a file which defines it. These options cannot be #defined in lwipopts.h since they are not options of lwIP itself, but options of the lwIP port to your system.

2.1.5.7.3 Macro Definition Documentation

2.1.5.7.3.1 BYTE_ORDER

```
#define BYTE_ORDER   LITTLE_ENDIAN
```

Define the byte order of the system. Needed for conversion of network data to host byte order. Allowed values: LITTLE_ENDIAN and BIG_ENDIAN

2.1.5.7.3.2 LWIP_ALIGNMENT_CAST

```
#define LWIP_ALIGNMENT_CAST( target_type, val)   LWIP_CONST_CAST(target_type, val)
```

Get rid of alignment cast warnings (GCC -Wcast-align)

2.1.5.7.3.3 LWIP_CONST_CAST

```
#define LWIP_CONST_CAST( target_type, val)   ((target_type)((ptrdiff_t)val))
```

C++ const_cast<target_type>(val) equivalent to remove constness from a value (GCC -Wcast-qual)

2.1.5.7.3.4 LWIP_DECLARE_MEMORY_ALIGNED

```
#define LWIP_DECLARE_MEMORY_ALIGNED( variable_name, size)   u8_t variable_name[LWIP_MEM_AL
```

Allocates a memory buffer of specified size that is of sufficient size to align its start address using LWIP_MEM_ALIGN. You can declare your own version here e.g. to enforce alignment without adding trailing padding bytes (see LWIP_MEM_ALIGN_BUFFER) or your own section placement requirements. e.g. if you use gcc and need 32 bit alignment: #define LWIP_DECLARE_MEMORY_ALIG size) u8_t variable_name[size] __attribute__((aligned(4))) or more portable: #define LWIP_DECLARE_MEMORY_ALIGNED(variabl size) u32_t variable_name[(size + sizeof(u32_t) - 1) / sizeof(u32_t)]

2.1.5.7.3.5 LWIP_MEM_ALIGN

```
#define LWIP_MEM_ALIGN( addr)   ((void *)(((mem_ptr_t)(addr) + MEM_ALIGNMENT - 1) & ~(mem_
```

Align a memory pointer to the alignment defined by MEM_ALIGNMENT so that ADDR % MEM_ALIGNMENT == 0

2.1.5.7.3.6 LWIP_MEM_ALIGN_BUFFER

```
#define LWIP_MEM_ALIGN_BUFFER( size)   (((size) + MEM_ALIGNMENT - 1U))
```

Calculate safe memory size for an aligned buffer when using an unaligned type as storage. This includes a safety-margin on (MEM_ALIGNMENT - 1) at the start (e.g. if buffer is u8_t[] and actual data will be u32_t*)

2.1.5.7.3.7 LWIP_MEM_ALIGN_SIZE

```
#define LWIP_MEM_ALIGN_SIZE( size)   (((size) + MEM_ALIGNMENT - 1U) & ~(MEM_ALIGNMENT-1U))
```

Calculate memory size for an aligned buffer - returns the next highest multiple of MEM_ALIGNMENT (e.g. LWIP_MEM_ALIGN_SIZI and LWIP_MEM_ALIGN_SIZE(4) will both yield 4 for MEM_ALIGNMENT == 4).

2.1.5.7.3.8 LWIP_NO_CTYPE_H

```
#define LWIP_NO_CTYPE_H   0
```

Define this to 1 in arch/cc.h of your port if your compiler does not provide the ctype.h header. If ctype.h is available, a few character functions are mapped to the appropriate functions (lwip_islower, lwip_isdigit...), if not, a private implementation is provided.

2.1.5.7.3.9 LWIP_NO_INTTYPES_H

```
#define LWIP_NO_INTTYPES_H   0
```

Define this to 1 in arch/cc.h of your port if your compiler does not provide the inttypes.h header. You need to define the format strings listed in lwip/arch.h yourself in this case (X8_F, U16_F...).

2.1.5.7.3.10 LWIP_NO_LIMITS_H

```
#define LWIP_NO_LIMITS_H   0
```

Define this to 1 in arch/cc.h of your port if your compiler does not provide the limits.h header. You need to define the type limits yourself in this case (e.g. INT_MAX, SSIZE_MAX).

2.1.5.7.3.11 LWIP_NO_STDDEF_H

```
#define LWIP_NO_STDDEF_H   0
```

Define this to 1 in arch/cc.h of your port if you do not want to include stddef.h header to get size_t. You need to typedef size_t by yourself in this case.

2.1.5.7.3.12 LWIP_NO_STDINT_H

```
#define LWIP_NO_STDINT_H   0
```

Define this to 1 in arch/cc.h of your port if your compiler does not provide the stdint.h header. You need to typedef the generic types listed in lwip/arch.h yourself in this case (u8_t, u16_t...).

2.1.5.7.3.13 LWIP_PACKED_CAST

```
#define LWIP_PACKED_CAST( target_type, val)   LWIP_CONST_CAST(target_type, val)
```

Avoid warnings/errors related to implicitly casting away packed attributes by doing a explicit cast

2.1.5.7.3.14 LWIP_PLATFORM_ASSERT

`#define LWIP_PLATFORM_ASSERT( x)` **Value:**

```
do {printf("Assertion \"%s\" failed at line %d in %s\n ↩
    ", \
x, __LINE__, __FILE__); fflush(NULL); abort();} while ↩
    (0)
```

Platform specific assertion handling. Note the default implementation pulls in printf, fflush and abort, which may in turn pull in a lot of standard library code. In resource-constrained systems, this should be defined to something less resource-consuming.

2.1.5.7.3.15 LWIP_PLATFORM_DIAG

```
#define LWIP_PLATFORM_DIAG( x)   do {printf x;} while(0)
```

Platform specific diagnostic output. Note the default implementation pulls in printf, which may in turn pull in a lot of standard library code. In resource-constrained systems, this should be defined to something less resource-consuming.

2.1.5.7.3.16 LWIP_PROVIDE_ERRNO

```
#define LWIP_PROVIDE_ERRNO
```

LWIP_PROVIDE_ERRNO==1: Let lwIP provide ERRNO values and the 'errno' variable. If this is disabled, cc.h must either define 'errno', include <errno.h>, define LWIP_ERRNO_STDINCLUDE to get <errno.h> included or define LWIP_ERRNO_INCLUDE to <errno.h> or equivalent.

2.1.5.7.3.17 LWIP_PTR_NUMERIC_CAST

```
#define LWIP_PTR_NUMERIC_CAST( target_type, val)   LWIP_CONST_CAST(target_type, val)
```

Get rid of warnings related to pointer-to-numeric and vice-versa casts, e.g. "conversion from 'u8_t' to 'void *' of greater size"

2.1.5.7.3.18 LWIP_RAND

```
#define LWIP_RAND( )   ((u32_t)rand())
```

Define random number generator function of your system

2.1.5.7.3.19 LWIP_UNUSED_ARG

```
#define LWIP_UNUSED_ARG( x)    (void)x
```

Eliminates compiler warning about unused arguments (GCC -Wextra -Wunused).

2.1.5.7.3.20 PACK_STRUCT_BEGIN

```
#define PACK_STRUCT_BEGIN
```

Packed structs support. Placed BEFORE declaration of a packed struct. For examples of packed struct declarations, see include/lwip/prot/ subfolder. A port to GCC/clang is included in lwIP, if you use these compilers there is nothing to do here.

2.1.5.7.3.21 PACK_STRUCT_END

```
#define PACK_STRUCT_END
```

Packed structs support. Placed AFTER declaration of a packed struct. For examples of packed struct declarations, see include/lwip/prot/ subfolder. A port to GCC/clang is included in lwIP, if you use these compilers there is nothing to do here.

2.1.5.7.3.22 PACK_STRUCT_FIELD

```
#define PACK_STRUCT_FIELD( x)    x
```

Packed structs support. Wraps u32_t and u16_t members. For examples of packed struct declarations, see include/lwip/prot/ subfolder. A port to GCC/clang is included in lwIP, if you use these compilers there is nothing to do here.

2.1.5.7.3.23 PACK_STRUCT_FLD_8

```
#define PACK_STRUCT_FLD_8( x)    PACK_STRUCT_FIELD(x)
```

Packed structs support. Wraps u8_t members, where some compilers warn that packing is not necessary. For examples of packed struct declarations, see include/lwip/prot/ subfolder. A port to GCC/clang is included in lwIP, if you use these compilers there is nothing to do here.

2.1.5.7.3.24 PACK_STRUCT_FLD_S

```
#define PACK_STRUCT_FLD_S( x)    PACK_STRUCT_FIELD(x)
```

Packed structs support. Wraps members that are packed structs themselves, where some compilers warn that packing is not necessary. For examples of packed struct declarations, see include/lwip/prot/ subfolder. A port to GCC/clang is included in lwIP, if you use these compilers there is nothing to do here.

2.1.5.7.3.25 PACK_STRUCT_STRUCT

```
#define PACK_STRUCT_STRUCT
```

Packed structs support. Placed between end of declaration of a packed struct and trailing semicolon. For examples of packed struct declarations, see include/lwip/prot/ subfolder. A port to GCC/clang is included in lwIP, if you use these compilers there is nothing to do here.

2.1.5.7.3.26 PACK_STRUCT_USE_INCLUDES

```
#define PACK_STRUCT_USE_INCLUDES
```

PACK_STRUCT_USE_INCLUDES==1: Packed structs support using #include files before and after struct to be packed. The file included BEFORE the struct is "arch/bpstruct.h". The file included AFTER the struct is "arch/epstruct.h". This can be used to implement struct packing on MS Visual C compilers, see the Win32 port in the lwIP/contrib subdir for reference. For examples of packed struct declarations, see include/lwip/prot/ subfolder. A port to GCC/clang is included in lwIP, if you use these compilers there is nothing to do here.

2.1.5.8 Performance measurement

All defines related to this section must not be placed in lwipopts.h, but in arch/perf.h! Measurement calls made throughout lwip, these can be defined to nothing.

- PERF_START: start measuring something.
- PERF_STOP(x): stop measuring something, and record the result.

2.1.6 Version

2.1.6.1 Macros

- #define LWIP_VERSION_MAJOR 2
- #define LWIP_VERSION_MINOR 2
- #define LWIP_VERSION_REVISION 2
- #define LWIP_VERSION_RC LWIP_RC_DEVELOPMENT
- #define LWIP_RC_RELEASE 255
- #define LWIP_RC_DEVELOPMENT 0
- #define LWIP_VERSION
- #define LWIP_VERSION_STRING LWIP_VERSTR(LWIP_VERSION_MAJOR) "." LWIP_VERSTR(LWIP_VERSION_MINOR) "." LWIP_VERSTR(LWIP_VERSION_REVISION) LWIP_VERSION_STRING_SUFFIX

2.1.6.2 Detailed Description

2.1.6.3 Macro Definition Documentation

2.1.6.3.1 LWIP_RC_DEVELOPMENT

```
#define LWIP_RC_DEVELOPMENT   0
```

LWIP_VERSION_RC is set to LWIP_RC_DEVELOPMENT for Git versions

2.1.6.3.2 LWIP_RC_RELEASE

```
#define LWIP_RC_RELEASE   255
```

LWIP_VERSION_RC is set to LWIP_RC_RELEASE for official releases

2.1.6.3.3 LWIP_VERSION

`#define LWIP_VERSION` **Value:**

```
                        ((LWIP_VERSION_MAJOR) << 24   | (LWIP_VERSION_MINOR) << 16 | \
                        (LWIP_VERSION_REVISION) << 8 | (LWIP_VERSION_RC))
```

Provides the version of the stack

2.1.6.3.4 LWIP_VERSION_MAJOR

```
#define LWIP_VERSION_MAJOR   2
```

X.x.x: Major version of the stack

2.1.6.3.5 LWIP_VERSION_MINOR

```
#define LWIP_VERSION_MINOR   2
```

x.X.x: Minor version of the stack

2.1.6.3.6 LWIP_VERSION_RC

```
#define LWIP_VERSION_RC   LWIP_RC_DEVELOPMENT
```

For release candidates, this is set to 1..254 For official releases, this is set to 255 (LWIP_RC_RELEASE) For development versions (Git), this is set to 0 (LWIP_RC_DEVELOPMENT)

2.1.6.3.7 LWIP_VERSION_REVISION

```
#define LWIP_VERSION_REVISION   2
```

x.x.X: Revision of the stack

2.1.6.3.8 LWIP_VERSION_STRING

```
#define LWIP_VERSION_STRING   LWIP_VERSTR(LWIP_VERSION_MAJOR) "." LWIP_VERSTR(LWIP_VERSION_
"." LWIP_VERSTR(LWIP_VERSION_REVISION) LWIP_VERSION_STRING_SUFFIX
```

Provides the version of the stack as string

2.1.7 Options (lwipopts.h)

2.1.7.1 Modules

- Debugging
- Infrastructure
- Callback-style APIs
- Thread-safe APIs
- IPv4
- PBUF
- NETIF
- IPv6

2.1.7.2 Detailed Description

2.1.7.3 Debugging

2.1.7.3.1 Modules

- Assertion handling
- Statistics
- Debug messages
- Performance

2.1.7.3.2 Detailed Description

2.1.7.3.3 Assertion handling

2.1.7.3.3.1 Macros

- #define LWIP_NOASSERT

2.1.7.3.3.2 Detailed Description

2.1.7.3.3.3 Macro Definition Documentation

2.1.7.3.3.4 LWIP_NOASSERT

```
#define LWIP_NOASSERT
```

LWIP_NOASSERT: Disable LWIP_ASSERT checks: To disable assertions define LWIP_NOASSERT in arch/cc.h.

2.1.7.3.4 Statistics

2.1.7.3.4.1 Macros

- #define LWIP_STATS 1
- #define LWIP_STATS_DISPLAY 0
- #define LINK_STATS 1
- #define ETHARP_STATS (LWIP_ARP)
- #define IP_STATS 1
- #define IPFRAG_STATS (IP_REASSEMBLY || IP_FRAG)
- #define ICMP_STATS 1
- #define IGMP_STATS (LWIP_IGMP)
- #define UDP_STATS (LWIP_UDP)
- #define TCP_STATS (LWIP_TCP)
- #define MEM_STATS ((MEM_CUSTOM_ALLOCATOR == 0) && (MEM_USE_POOLS == 0))
- #define MEMP_STATS (MEMP_MEM_MALLOC == 0)
- #define SYS_STATS (NO_SYS == 0)
- #define IP6_STATS (LWIP_IPV6)
- #define ICMP6_STATS (LWIP_IPV6 && LWIP_ICMP6)
- #define IP6_FRAG_STATS (LWIP_IPV6 && (LWIP_IPV6_FRAG || LWIP_IPV6_REASS))
- #define MLD6_STATS (LWIP_IPV6 && LWIP_IPV6_MLD)
- #define ND6_STATS (LWIP_IPV6)
- #define MIB2_STATS 0

2.1.7.3.4.2 Detailed Description

2.1.7.3.4.3 Macro Definition Documentation

2.1.7.3.4.4 ETHARP_STATS

```
#define ETHARP_STATS   (LWIP_ARP)
```

ETHARP_STATS==1: Enable etharp stats.

2.1.7.3.4.5 ICMP6_STATS

```
#define ICMP6_STATS   (LWIP_IPV6 && LWIP_ICMP6)
```

ICMP6_STATS==1: Enable ICMP for IPv6 stats.

2.1.7.3.4.6 ICMP_STATS

```
#define ICMP_STATS   1
```

ICMP_STATS==1: Enable ICMP stats.

2.1.7.3.4.7 IGMP_STATS

```
#define IGMP_STATS   (LWIP_IGMP)
```

IGMP_STATS==1: Enable IGMP stats.

2.1.7.3.4.8 IP6_FRAG_STATS

```
#define IP6_FRAG_STATS   (LWIP_IPV6 && (LWIP_IPV6_FRAG || LWIP_IPV6_REASS))
```

IP6_FRAG_STATS==1: Enable IPv6 fragmentation stats.

2.1.7.3.4.9 IP6_STATS

```
#define IP6_STATS   (LWIP_IPV6)
```

IP6_STATS==1: Enable IPv6 stats.

2.1.7.3.4.10 IP_STATS

```
#define IP_STATS   1
```

IP_STATS==1: Enable IP stats.

2.1.7.3.4.11 IPFRAG_STATS

```
#define IPFRAG_STATS   (IP_REASSEMBLY || IP_FRAG)
```

IPFRAG_STATS==1: Enable IP fragmentation stats. Default is on if using either frag or reass.

2.1.7.3.4.12 LINK_STATS

```
#define LINK_STATS   1
```

LINK_STATS==1: Enable link stats.

2.1.7.3.4.13 LWIP_STATS

```
#define LWIP_STATS   1
```

LWIP_STATS==1: Enable statistics collection in lwip_stats.

2.1.7.3.4.14 LWIP_STATS_DISPLAY

```
#define LWIP_STATS_DISPLAY   0
```

LWIP_STATS_DISPLAY==1: Compile in the statistics output functions.

2.1.7.3.4.15 MEM_STATS

```
#define MEM_STATS   ((MEM_CUSTOM_ALLOCATOR == 0) && (MEM_USE_POOLS == 0))
```

MEM_STATS==1: Enable mem.c stats.

2.1.7.3.4.16 MEMP_STATS

```
#define MEMP_STATS   (MEMP_MEM_MALLOC == 0)
```

MEMP_STATS==1: Enable memp.c pool stats.

2.1.7.3.4.17 MIB2_STATS

```
#define MIB2_STATS   0
```

MIB2_STATS==1: Stats for SNMP MIB2.

2.1.7.3.4.18 MLD6_STATS

```
#define MLD6_STATS   (LWIP_IPV6 && LWIP_IPV6_MLD)
```

MLD6_STATS==1: Enable MLD for IPv6 stats.

2.1.7.3.4.19 ND6_STATS

```
#define ND6_STATS   (LWIP_IPV6)
```

ND6_STATS==1: Enable Neighbor discovery for IPv6 stats.

2.1.7.3.4.20 SYS_STATS

```
#define SYS_STATS   (NO_SYS == 0)
```

SYS_STATS==1: Enable system stats (sem and mbox counts, etc).

2.1.7.3.4.21 TCP_STATS

```
#define TCP_STATS   (LWIP_TCP)
```

TCP_STATS==1: Enable TCP stats. Default is on if TCP enabled, otherwise off.

2.1.7.3.4.22 UDP_STATS

```
#define UDP_STATS   (LWIP_UDP)
```

UDP_STATS==1: Enable UDP stats. Default is on if UDP enabled, otherwise off.

2.1.7.3.5 Debug messages

2.1.7.3.5.1 Modules

- LWIP_DBG_MIN_LEVEL and LWIP_DBG_TYPES_ON values

2.1.7.3.5.2 Macros

- #define LWIP_DBG_MIN_LEVEL LWIP_DBG_LEVEL_ALL
- #define LWIP_DBG_TYPES_ON LWIP_DBG_ON
- #define ETHARP_DEBUG LWIP_DBG_OFF
- #define NETIF_DEBUG LWIP_DBG_OFF
- #define PBUF_DEBUG LWIP_DBG_OFF
- #define API_LIB_DEBUG LWIP_DBG_OFF
- #define API_MSG_DEBUG LWIP_DBG_OFF
- #define SOCKETS_DEBUG LWIP_DBG_OFF
- #define ICMP_DEBUG LWIP_DBG_OFF
- #define IGMP_DEBUG LWIP_DBG_OFF
- #define INET_DEBUG LWIP_DBG_OFF
- #define IP_DEBUG LWIP_DBG_OFF
- #define IP_REASS_DEBUG LWIP_DBG_OFF
- #define RAW_DEBUG LWIP_DBG_OFF
- #define MEM_DEBUG LWIP_DBG_OFF
- #define MEMP_DEBUG LWIP_DBG_OFF
- #define SYS_DEBUG LWIP_DBG_OFF
- #define TIMERS_DEBUG LWIP_DBG_OFF
- #define TCP_DEBUG LWIP_DBG_OFF
- #define TCP_INPUT_DEBUG LWIP_DBG_OFF
- #define TCP_FR_DEBUG LWIP_DBG_OFF
- #define TCP_RTO_DEBUG LWIP_DBG_OFF
- #define TCP_CWND_DEBUG LWIP_DBG_OFF
- #define TCP_WND_DEBUG LWIP_DBG_OFF
- #define TCP_OUTPUT_DEBUG LWIP_DBG_OFF
- #define TCP_RST_DEBUG LWIP_DBG_OFF
- #define TCP_QLEN_DEBUG LWIP_DBG_OFF
- #define UDP_DEBUG LWIP_DBG_OFF
- #define TCPIP_DEBUG LWIP_DBG_OFF
- #define SLIP_DEBUG LWIP_DBG_OFF

- #define DHCP_DEBUG LWIP_DBG_OFF
- #define AUTOIP_DEBUG LWIP_DBG_OFF
- #define ACD_DEBUG LWIP_DBG_OFF
- #define DNS_DEBUG LWIP_DBG_OFF
- #define IP6_DEBUG LWIP_DBG_OFF
- #define DHCP6_DEBUG LWIP_DBG_OFF

2.1.7.3.5.3 Detailed Description

2.1.7.3.5.4 Macro Definition Documentation

2.1.7.3.5.5 ACD_DEBUG

```
#define ACD_DEBUG   LWIP_DBG_OFF
```

ACD_DEBUG: Enable debugging in acd.c.

2.1.7.3.5.6 API_LIB_DEBUG

```
#define API_LIB_DEBUG   LWIP_DBG_OFF
```

API_LIB_DEBUG: Enable debugging in api_lib.c.

2.1.7.3.5.7 API_MSG_DEBUG

```
#define API_MSG_DEBUG   LWIP_DBG_OFF
```

API_MSG_DEBUG: Enable debugging in api_msg.c.

2.1.7.3.5.8 AUTOIP_DEBUG

```
#define AUTOIP_DEBUG   LWIP_DBG_OFF
```

AUTOIP_DEBUG: Enable debugging in autoip.c.

2.1.7.3.5.9 DHCP6_DEBUG

```
#define DHCP6_DEBUG   LWIP_DBG_OFF
```

DHCP6_DEBUG: Enable debugging in dhcp6.c.

2.1.7.3.5.10 DHCP_DEBUG

```
#define DHCP_DEBUG   LWIP_DBG_OFF
```

DHCP_DEBUG: Enable debugging in dhcp.c.

2.1.7.3.5.11 DNS_DEBUG

```
#define DNS_DEBUG   LWIP_DBG_OFF
```

DNS_DEBUG: Enable debugging for DNS.

2.1.7.3.5.12 ETHARP_DEBUG

```
#define ETHARP_DEBUG   LWIP_DBG_OFF
```

ETHARP_DEBUG: Enable debugging in etharp.c.

2.1.7.3.5.13 ICMP_DEBUG

```
#define ICMP_DEBUG   LWIP_DBG_OFF
```

ICMP_DEBUG: Enable debugging in icmp.c.

2.1.7.3.5.14 IGMP_DEBUG

```
#define IGMP_DEBUG   LWIP_DBG_OFF
```

IGMP_DEBUG: Enable debugging in igmp.c.

2.1.7.3.5.15 INET_DEBUG

```
#define INET_DEBUG   LWIP_DBG_OFF
```

INET_DEBUG: Enable debugging in inet.c.

2.1.7.3.5.16 IP6_DEBUG

```
#define IP6_DEBUG   LWIP_DBG_OFF
```

IP6_DEBUG: Enable debugging for IPv6.

2.1.7.3.5.17 IP_DEBUG

```
#define IP_DEBUG   LWIP_DBG_OFF
```

IP_DEBUG: Enable debugging for IP.

2.1.7.3.5.18 IP_REASS_DEBUG

```
#define IP_REASS_DEBUG   LWIP_DBG_OFF
```

IP_REASS_DEBUG: Enable debugging in ip_frag.c for both frag & reass.

2.1.7.3.5.19 LWIP_DBG_MIN_LEVEL

```
#define LWIP_DBG_MIN_LEVEL   LWIP_DBG_LEVEL_ALL
```

LWIP_DBG_MIN_LEVEL: After masking, the value of the debug is compared against this value. If it is smaller, then debugging messages are written. **See also** LWIP_DBG_MIN_LEVEL and LWIP_DBG_TYPES_ON values

2.1.7.3.5.20 LWIP_DBG_TYPES_ON

```
#define LWIP_DBG_TYPES_ON   LWIP_DBG_ON
```

LWIP_DBG_TYPES_ON: A mask that can be used to globally enable/disable debug messages of certain types. **See also** LWIP_DBG_MIN_LEVEL and LWIP_DBG_TYPES_ON values

2.1.7.3.5.21 MEM_DEBUG

```
#define MEM_DEBUG   LWIP_DBG_OFF
```

MEM_DEBUG: Enable debugging in mem.c.

2.1.7.3.5.22 MEMP_DEBUG

```
#define MEMP_DEBUG   LWIP_DBG_OFF
```

MEMP_DEBUG: Enable debugging in memp.c.

2.1.7.3.5.23 NETIF_DEBUG

```
#define NETIF_DEBUG   LWIP_DBG_OFF
```

NETIF_DEBUG: Enable debugging in netif.c.

2.1.7.3.5.24 PBUF_DEBUG

```
#define PBUF_DEBUG   LWIP_DBG_OFF
```

PBUF_DEBUG: Enable debugging in pbuf.c.

2.1.7.3.5.25 RAW_DEBUG

```
#define RAW_DEBUG   LWIP_DBG_OFF
```

RAW_DEBUG: Enable debugging in raw.c.

2.1.7.3.5.26 SLIP_DEBUG

```
#define SLIP_DEBUG   LWIP_DBG_OFF
```

SLIP_DEBUG: Enable debugging in slipif.c.

2.1.7.3.5.27 SOCKETS_DEBUG

```
#define SOCKETS_DEBUG   LWIP_DBG_OFF
```

SOCKETS_DEBUG: Enable debugging in sockets.c.

2.1.7.3.5.28 SYS_DEBUG

```
#define SYS_DEBUG   LWIP_DBG_OFF
```

SYS_DEBUG: Enable debugging in sys.c.

2.1.7.3.5.29 TCP_CWND_DEBUG

```
#define TCP_CWND_DEBUG   LWIP_DBG_OFF
```

TCP_CWND_DEBUG: Enable debugging for TCP congestion window.

2.1.7.3.5.30 TCP_DEBUG

```
#define TCP_DEBUG   LWIP_DBG_OFF
```

TCP_DEBUG: Enable debugging for TCP.

2.1.7.3.5.31 TCP_FR_DEBUG

```
#define TCP_FR_DEBUG   LWIP_DBG_OFF
```

TCP_FR_DEBUG: Enable debugging in tcp_in.c for fast retransmit.

2.1.7.3.5.32 TCP_INPUT_DEBUG

```
#define TCP_INPUT_DEBUG   LWIP_DBG_OFF
```

TCP_INPUT_DEBUG: Enable debugging in tcp_in.c for incoming debug.

2.1.7.3.5.33 TCP_OUTPUT_DEBUG

```
#define TCP_OUTPUT_DEBUG   LWIP_DBG_OFF
```

TCP_OUTPUT_DEBUG: Enable debugging in tcp_out.c output functions.

2.1.7.3.5.34 TCP_QLEN_DEBUG

```
#define TCP_QLEN_DEBUG   LWIP_DBG_OFF
```

TCP_QLEN_DEBUG: Enable debugging for TCP queue lengths.

2.1.7.3.5.35 TCP_RST_DEBUG

```
#define TCP_RST_DEBUG   LWIP_DBG_OFF
```

TCP_RST_DEBUG: Enable debugging for TCP with the RST message.

2.1.7.3.5.36 TCP_RTO_DEBUG

```
#define TCP_RTO_DEBUG   LWIP_DBG_OFF
```

TCP_RTO_DEBUG: Enable debugging in TCP for retransmit timeout.

2.1.7.3.5.37 TCP_WND_DEBUG

```
#define TCP_WND_DEBUG   LWIP_DBG_OFF
```

TCP_WND_DEBUG: Enable debugging in tcp_in.c for window updating.

2.1.7.3.5.38 TCPIP_DEBUG

```
#define TCPIP_DEBUG   LWIP_DBG_OFF
```

TCPIP_DEBUG: Enable debugging in tcpip.c.

2.1.7.3.5.39 TIMERS_DEBUG

```
#define TIMERS_DEBUG   LWIP_DBG_OFF
```

TIMERS_DEBUG: Enable debugging in timers.c.

2.1.7.3.5.40 UDP_DEBUG

```
#define UDP_DEBUG   LWIP_DBG_OFF
```

UDP_DEBUG: Enable debugging in UDP.

2.1.7.3.5.41 LWIP_DBG_MIN_LEVEL and LWIP_DBG_TYPES_ON values

2.1.7.3.5.42 Debug level (LWIP_DBG_MIN_LEVEL)

- #define LWIP_DBG_LEVEL_ALL 0x00
- #define LWIP_DBG_LEVEL_WARNING 0x01
- #define LWIP_DBG_LEVEL_SERIOUS 0x02
- #define LWIP_DBG_LEVEL_SEVERE 0x03

2.1.7.3.5.43 Enable/disable debug messages completely (LWIP_DBG_TYPES_ON)

- #define LWIP_DBG_ON 0x80U
- #define LWIP_DBG_OFF 0x00U

2.1.7.3.5.44 Debug message types (LWIP_DBG_TYPES_ON)

- #define LWIP_DBG_TRACE 0x40U
- #define LWIP_DBG_STATE 0x20U
- #define LWIP_DBG_FRESH 0x10U
- #define LWIP_DBG_HALT 0x08U

2.1.7.3.5.45 Detailed Description

2.1.7.3.5.46 Macro Definition Documentation

2.1.7.3.5.47 LWIP_DBG_FRESH

```
#define LWIP_DBG_FRESH   0x10U
```

flag for LWIP_DEBUGF indicating newly added code, not thoroughly tested yet

2.1.7.3.5.48 LWIP_DBG_HALT

```
#define LWIP_DBG_HALT   0x08U
```

flag for LWIP_DEBUGF to halt after printing this debug message

2.1.7.3.5.49 LWIP_DBG_LEVEL_ALL

```
#define LWIP_DBG_LEVEL_ALL   0x00
```

Debug level: ALL messages

2.1.7.3.5.50 LWIP_DBG_LEVEL_SERIOUS

```
#define LWIP_DBG_LEVEL_SERIOUS   0x02
```

Debug level: Serious. memory allocation failures, ...

2.1.7.3.5.51 LWIP_DBG_LEVEL_SEVERE

```
#define LWIP_DBG_LEVEL_SEVERE   0x03
```

Debug level: Severe

2.1.7.3.5.52 LWIP_DBG_LEVEL_WARNING

```
#define LWIP_DBG_LEVEL_WARNING   0x01
```

Debug level: Warnings. bad checksums, dropped packets, ...

2.1.7.3.5.53 LWIP_DBG_OFF

```
#define LWIP_DBG_OFF   0x00U
```

flag for LWIP_DEBUGF to disable that debug message

2.1.7.3.5.54 LWIP_DBG_ON

```
#define LWIP_DBG_ON   0x80U
```

flag for LWIP_DEBUGF to enable that debug message

2.1.7.3.5.55 LWIP_DBG_STATE

```
#define LWIP_DBG_STATE   0x20U
```

flag for LWIP_DEBUGF indicating a state debug message (to follow module states)

2.1.7.3.5.56 LWIP_DBG_TRACE

```
#define LWIP_DBG_TRACE   0x40U
```

flag for LWIP_DEBUGF indicating a tracing message (to follow program flow)

2.1.7.3.6 Performance

2.1.7.3.6.1 Macros

- #define LWIP_PERF 0

2.1.7.3.6.2 Detailed Description

LWIP_TESTMODE: Changes to make unit test possible

2.1.7.3.6.3 Macro Definition Documentation

2.1.7.3.6.4 LWIP_PERF

```
#define LWIP_PERF   0
```

LWIP_PERF: Enable performance testing for lwIP (if enabled, arch/perf.h is included)

2.1.7.4 Infrastructure

2.1.7.4.1 Modules

- NO_SYS
- Timers
- memcpy

- Core locking and MPU
- Heap and memory pools
- Internal memory pools
- SNMP MIB2 callbacks
- Multicast
- Threading
- Checksum
- Hooks

2.1.7.4.2 Detailed Description

2.1.7.4.3 NO_SYS

2.1.7.4.3.1 Macros

- #define NO_SYS 0

2.1.7.4.3.2 Detailed Description

2.1.7.4.3.3 Macro Definition Documentation

2.1.7.4.3.4 NO_SYS

```
#define NO_SYS   0
```

NO_SYS==1: Use lwIP without OS-awareness (no thread, semaphores, mutexes or mboxes). This means threaded APIs cannot be used (socket, netconn, i.e. everything in the 'api' folder), only the callback-style raw API is available (and you have to watch out for yourself that you don't access lwIP functions/structures from more than one context at a time!)

2.1.7.4.4 Timers

2.1.7.4.4.1 Macros

- #define LWIP_TIMERS 1
- #define LWIP_TIMERS_CUSTOM 0

2.1.7.4.4.2 Detailed Description

2.1.7.4.4.3 Macro Definition Documentation

2.1.7.4.4.4 LWIP_TIMERS

```
#define LWIP_TIMERS   1
```

LWIP_TIMERS==0: Drop support for sys_timeout and lwip-internal cyclic timers. (the array of lwip-internal cyclic timers is still provided) (check NO_SYS_NO_TIMERS for compatibility to old versions)

2.1.7.4.4.5 LWIP_TIMERS_CUSTOM

```
#define LWIP_TIMERS_CUSTOM   0
```

LWIP_TIMERS_CUSTOM==1: Provide your own timer implementation. Function prototypes in timeouts.h and the array of lwip-internal cyclic timers are still included, but the implementation is not. The following functions will be required: sys_timeouts_init(), sys_timeout(), sys_untimeout(), sys_timeouts_mbox_fetch()

2.1.7.4.5 memcpy

2.1.7.4.5.1 Macros

- #define MEMCPY(dst, src, len) memcpy(dst,src,len)
- #define SMEMCPY(dst, src, len) memcpy(dst,src,len)
- #define MEMMOVE(dst, src, len) memmove(dst,src,len)

2.1.7.4.5.2 Detailed Description

2.1.7.4.5.3 Macro Definition Documentation

2.1.7.4.5.4 MEMCPY

```
#define MEMCPY( dst, src, len)   memcpy(dst,src,len)
```

MEMCPY: override this if you have a faster implementation at hand than the one included in your C library

2.1.7.4.5.5 MEMMOVE

```
#define MEMMOVE( dst, src, len)   memmove(dst,src,len)
```

MEMMOVE: override this if you have a faster implementation at hand than the one included in your C library. lwIP currently uses MEMMOVE only when IPv6 fragmentation support is enabled.

2.1.7.4.5.6 SMEMCPY

```
#define SMEMCPY( dst, src, len)   memcpy(dst,src,len)
```

SMEMCPY: override this with care! Some compilers (e.g. gcc) can inline a call to memcpy() if the length is known at compile time and is small.

2.1.7.4.6 Core locking and MPU

2.1.7.4.6.1 Macros

- #define LWIP_MPU_COMPATIBLE 0
- #define LWIP_TCPIP_CORE_LOCKING 1
- #define LWIP_TCPIP_CORE_LOCKING_INPUT 0
- #define SYS_LIGHTWEIGHT_PROT 1
- #define LWIP_ASSERT_CORE_LOCKED()
- #define LWIP_MARK_TCPIP_THREAD()

2.1.7.4.6.2 Detailed Description

2.1.7.4.6.3 Macro Definition Documentation

2.1.7.4.6.4 LWIP_ASSERT_CORE_LOCKED

```
#define LWIP_ASSERT_CORE_LOCKED( )
```

Macro/function to check whether lwIP's threading/locking requirements are satisfied during current function call. This macro usually calls a function that is implemented in the OS-dependent sys layer and performs the following checks:

- Not in ISR (this should be checked for NO_SYS==1, too!)
- If LWIP_TCPIP_CORE_LOCKING = 1: TCPIP core lock is held
- If LWIP_TCPIP_CORE_LOCKING = 0: function is called from TCPIP thread **See also** Multithreading

2.1.7.4.6.5 LWIP_MARK_TCPIP_THREAD

```
#define LWIP_MARK_TCPIP_THREAD( )
```

Called as first thing in the lwIP TCPIP thread. Can be used in conjunction with LWIP_ASSERT_CORE_LOCKED to check core locking. **See also** Multithreading

2.1.7.4.6.6 LWIP_MPU_COMPATIBLE

```
#define LWIP_MPU_COMPATIBLE   0
```

LWIP_MPU_COMPATIBLE: enables special memory management mechanism which makes lwip able to work on MPU (Memory Protection Unit) system by not passing stack-pointers to other threads (this decreases performance as memory is allocated from pools instead of keeping it on the stack)

2.1.7.4.6.7 LWIP_TCPIP_CORE_LOCKING

```
#define LWIP_TCPIP_CORE_LOCKING   1
```

LWIP_TCPIP_CORE_LOCKING Creates a global mutex that is held during TCPIP thread operations. Can be locked by client code to perform lwIP operations without changing into TCPIP thread using callbacks. See LOCK_TCPIP_CORE() and UNLOCK_TCPIP_CORE(). Your system should provide mutexes supporting priority inversion to use this.

2.1.7.4.6.8 LWIP_TCPIP_CORE_LOCKING_INPUT

```
#define LWIP_TCPIP_CORE_LOCKING_INPUT   0
```

LWIP_TCPIP_CORE_LOCKING_INPUT: when LWIP_TCPIP_CORE_LOCKING is enabled, this lets tcpip_input() grab the mutex for input packets as well, instead of allocating a message and passing it to tcpip_thread.

ATTENTION: this does not work when tcpip_input() is called from interrupt context!

2.1.7.4.6.9 SYS_LIGHTWEIGHT_PROT

```
#define SYS_LIGHTWEIGHT_PROT   1
```

SYS_LIGHTWEIGHT_PROT==1: enable inter-task protection (and task-vs-interrupt protection) for certain critical regions during buffer allocation, deallocation and memory allocation and deallocation. ATTENTION: This is required when using lwIP from more than one context! If you disable this, you must be sure what you are doing!

2.1.7.4.7 Heap and memory pools

2.1.7.4.7.1 Macros

- #define MEM_LIBC_MALLOC 0
- #define MEM_CUSTOM_ALLOCATOR 0
- #define MEMP_MEM_MALLOC 0
- #define MEMP_MEM_INIT 0
- #define MEM_ALIGNMENT 1

- #define MEM_SIZE 1600
- #define MEMP_OVERFLOW_CHECK 0
- #define MEMP_SANITY_CHECK 0
- #define MEM_OVERFLOW_CHECK 0
- #define MEM_SANITY_CHECK 0
- #define MEM_USE_POOLS 0
- #define MEM_USE_POOLS_TRY_BIGGER_POOL 0
- #define MEMP_USE_CUSTOM_POOLS 0
- #define LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT 0

2.1.7.4.7.2 Detailed Description

2.1.7.4.7.3 Macro Definition Documentation

2.1.7.4.7.4 LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT

```
#define LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT    0
```

Set this to 1 if you want to free PBUF_RAM pbufs (or call mem_free()) from interrupt context (or another context that doesn't allow waiting for a semaphore). If set to 1, mem_malloc will be protected by a semaphore and SYS_ARCH_PROTECT, while mem_free will only use SYS_ARCH_PROTECT. mem_malloc SYS_ARCH_UNPROTECTs with each loop so that mem_free can run.

ATTENTION: As you can see from the above description, this leads to dis-/ enabling interrupts often, which can be slow! Also, on low memory, mem_malloc can need longer.

If you don't want that, at least for NO_SYS=0, you can still use the following functions to enqueue a deallocation call which then runs in the tcpip_thread context:

- pbuf_free_callback(p);
- mem_free_callback(m);

2.1.7.4.7.5 MEM_ALIGNMENT

```
#define MEM_ALIGNMENT    1
```

MEM_ALIGNMENT: should be set to the alignment of the CPU 4 byte alignment -> #define MEM_ALIGNMENT 4 2 byte alignment -> #define MEM_ALIGNMENT 2

2.1.7.4.7.6 MEM_CUSTOM_ALLOCATOR

```
#define MEM_CUSTOM_ALLOCATOR    0
```

MEM_CUSTOM_ALLOCATOR==1: Use malloc/free/realloc provided by a custom implementation instead of the lwip internal allocator. Can save code size if you already use it. If enabled, you have to define those functions: #define MEM_CUSTOM_FREE my_free #define MEM_CUSTOM_MALLOC my_malloc #define MEM_CUSTOM_CALLOC my_calloc

2.1.7.4.7.7 MEM_LIBC_MALLOC

```
#define MEM_LIBC_MALLOC    0
```

MEM_LIBC_MALLOC==1: Use malloc/free/realloc provided by your C-library instead of the lwip internal allocator. Can save code size if you already use it. Specialized case of MEM_CUSTOM_ALLOCATOR. **See also** MEM_CUSTOM_ALLOCATOR

2.1.7.4.7.8 MEM_OVERFLOW_CHECK

```
#define MEM_OVERFLOW_CHECK   0
```

MEM_OVERFLOW_CHECK: mem overflow protection reserves a configurable amount of bytes before and after each heap allocation chunk and fills it with a prominent default value. MEM_OVERFLOW_CHECK == 0 no checking MEM_OVERFLOW_CHECK == 1 checks each element when it is freed MEM_OVERFLOW_CHECK >= 2 checks all heap elements every time mem_malloc() or mem_free() is called (useful but slow!)

2.1.7.4.7.9 MEM_SANITY_CHECK

```
#define MEM_SANITY_CHECK   0
```

MEM_SANITY_CHECK==1: run a sanity check after each mem_free() to make sure that the linked list of heap elements is not corrupted.

2.1.7.4.7.10 MEM_SIZE

```
#define MEM_SIZE   1600
```

MEM_SIZE: the size of the heap memory. If the application will send a lot of data that needs to be copied, this should be set high.

2.1.7.4.7.11 MEM_USE_POOLS

```
#define MEM_USE_POOLS   0
```

MEM_USE_POOLS==1: Use an alternative to malloc() by allocating from a set of memory pools of various sizes. When mem_malloc is called, an element of the smallest pool that can provide the length needed is returned. To use this, MEMP_USE_CUSTOM
also has to be enabled.

2.1.7.4.7.12 MEM_USE_POOLS_TRY_BIGGER_POOL

```
#define MEM_USE_POOLS_TRY_BIGGER_POOL   0
```

MEM_USE_POOLS_TRY_BIGGER_POOL==1: if one malloc-pool is empty, try the next bigger pool - WARNING: THIS MIGHT WASTE MEMORY but it can make a system more reliable.

2.1.7.4.7.13 MEMP_MEM_INIT

```
#define MEMP_MEM_INIT   0
```

MEMP_MEM_INIT==1: Force use of memset to initialize pool memory. Useful if pool are moved in uninitialized section of memory. This will ensure default values in pcbs struct are well initialized in all conditions.

2.1.7.4.7.14 MEMP_MEM_MALLOC

```
#define MEMP_MEM_MALLOC   0
```

MEMP_MEM_MALLOC==1: Use mem_malloc/mem_free instead of the lwip pool allocator. Especially useful with MEM_LIBC_MAI
but handle with care regarding execution speed (heap alloc can be much slower than pool alloc) and usage from interrupts (especially if your netif driver allocates PBUF_POOL pbufs for received frames from interrupt)! ATTENTION: Currently, this uses the heap for ALL pools (also for private pools, not only for internal pools defined in memp_std.h)!

2.1.7.4.7.15 MEMP_OVERFLOW_CHECK

```
#define MEMP_OVERFLOW_CHECK   0
```

MEMP_OVERFLOW_CHECK: memp overflow protection reserves a configurable amount of bytes before and after each memp element in every pool and fills it with a prominent default value. MEMP_OVERFLOW_CHECK == 0 no checking MEMP_OVERFLOW == 1 checks each element when it is freed MEMP_OVERFLOW_CHECK >= 2 checks each element in every pool every time memp_malloc() or memp_free() is called (useful but slow!)

2.1.7.4.7.16 MEMP_SANITY_CHECK

```
#define MEMP_SANITY_CHECK   0
```

MEMP_SANITY_CHECK==1: run a sanity check after each memp_free() to make sure that there are no cycles in the linked lists.

2.1.7.4.7.17 MEMP_USE_CUSTOM_POOLS

```
#define MEMP_USE_CUSTOM_POOLS   0
```

MEMP_USE_CUSTOM_POOLS==1: whether to include a user file lwippools.h that defines additional pools beyond the "standard" ones required by lwIP. If you set this to 1, you must have lwippools.h in your include path somewhere.

2.1.7.4.8 Internal memory pools

2.1.7.4.8.1 Macros

- #define MEMP_NUM_PBUF 16
- #define MEMP_NUM_RAW_PCB 4
- #define MEMP_NUM_UDP_PCB 4
- #define MEMP_NUM_TCP_PCB 5
- #define MEMP_NUM_TCP_PCB_LISTEN 8
- #define MEMP_NUM_TCP_SEG 16
- #define MEMP_NUM_ALTCP_PCB MEMP_NUM_TCP_PCB
- #define MEMP_NUM_REASSDATA 5
- #define MEMP_NUM_FRAG_PBUF 15
- #define MEMP_NUM_ARP_QUEUE 30
- #define MEMP_NUM_IGMP_GROUP 8
- #define LWIP_NUM_SYS_TIMEOUT_INTERNAL (LWIP_TCP + IP_REASSEMBLY + LWIP_ARP + (2*LWIP_DHCP) + LWIP_ACD + LWIP_IGMP + LWIP_DNS + PPP_NUM_TIMEOUTS + (LWIP_IPV6 * (1 + LWIP_IPV6_REASS + LWIP_IPV6_MLD + LWIP_IPV6_DHCP6)))
- #define MEMP_NUM_SYS_TIMEOUT LWIP_NUM_SYS_TIMEOUT_INTERNAL
- #define MEMP_NUM_NETBUF 2
- #define MEMP_NUM_NETCONN 4
- #define MEMP_NUM_SELECT_CB 4
- #define MEMP_NUM_TCPIP_MSG_API 8

- #define MEMP_NUM_TCPIP_MSG_INPKT 8
- #define MEMP_NUM_NETDB 1
- #define MEMP_NUM_LOCALHOSTLIST 1
- #define PBUF_POOL_SIZE 16
- #define MEMP_NUM_API_MSG MEMP_NUM_TCPIP_MSG_API
- #define MEMP_NUM_DNS_API_MSG MEMP_NUM_TCPIP_MSG_API
- #define MEMP_NUM_SOCKET_SETGETSOCKOPT_DATA MEMP_NUM_TCPIP_MSG_API
- #define MEMP_NUM_NETIFAPI_MSG MEMP_NUM_TCPIP_MSG_API

2.1.7.4.8.2 Detailed Description

2.1.7.4.8.3 Macro Definition Documentation

2.1.7.4.8.4 LWIP_NUM_SYS_TIMEOUT_INTERNAL

```
#define LWIP_NUM_SYS_TIMEOUT_INTERNAL   (LWIP_TCP + IP_REASSEMBLY + LWIP_ARP + (2*LWIP_DHC
+ LWIP_ACD + LWIP_IGMP + LWIP_DNS + PPP_NUM_TIMEOUTS + (LWIP_IPV6 * (1 + LWIP_IPV6_REASS
+ LWIP_IPV6_MLD + LWIP_IPV6_DHCP6)))
```

The number of sys timeouts used by the core stack (not apps) The default number of timeouts is calculated here for all enabled modules.

2.1.7.4.8.5 MEMP_NUM_ALTCP_PCB

```
#define MEMP_NUM_ALTCP_PCB   MEMP_NUM_TCP_PCB
```

MEMP_NUM_ALTCP_PCB: the number of simultaneously active altcp layer pcbs. (requires the LWIP_ALTCP option) Connections with multiple layers require more than one altcp_pcb (e.g. TLS over TCP requires 2 altcp_pcbs, one for TLS and one for TCP).

2.1.7.4.8.6 MEMP_NUM_API_MSG

```
#define MEMP_NUM_API_MSG   MEMP_NUM_TCPIP_MSG_API
```

MEMP_NUM_API_MSG: the number of concurrently active calls to various socket, netconn, and tcpip functions

2.1.7.4.8.7 MEMP_NUM_ARP_QUEUE

```
#define MEMP_NUM_ARP_QUEUE   30
```

MEMP_NUM_ARP_QUEUE: the number of simultaneously queued outgoing packets (pbufs) that are waiting for an ARP request (to resolve their destination address) to finish. (requires the ARP_QUEUEING option)

2.1.7.4.8.8 MEMP_NUM_DNS_API_MSG

```
#define MEMP_NUM_DNS_API_MSG   MEMP_NUM_TCPIP_MSG_API
```

MEMP_NUM_DNS_API_MSG: the number of concurrently active calls to netconn_gethostbyname

2.1.7.4.8.9 MEMP_NUM_FRAG_PBUF

```
#define MEMP_NUM_FRAG_PBUF   15
```

MEMP_NUM_FRAG_PBUF: the number of IP fragments simultaneously sent (fragments, not whole packets!). This is only used with LWIP_NETIF_TX_SINGLE_PBUF==0 and only has to be > 1 with DMA-enabled MACs where the packet is not yet sent when netif->output returns.

2.1.7.4.8.10 MEMP_NUM_IGMP_GROUP

```
#define MEMP_NUM_IGMP_GROUP   8
```

MEMP_NUM_IGMP_GROUP: The number of multicast groups whose network interfaces can be members at the same time (one per netif - allsystems group -, plus one per netif membership). (requires the LWIP_IGMP option)

2.1.7.4.8.11 MEMP_NUM_LOCALHOSTLIST

```
#define MEMP_NUM_LOCALHOSTLIST   1
```

MEMP_NUM_LOCALHOSTLIST: the number of host entries in the local host list if DNS_LOCAL_HOSTLIST_IS_DYNAMIC==1.

2.1.7.4.8.12 MEMP_NUM_NETBUF

```
#define MEMP_NUM_NETBUF   2
```

MEMP_NUM_NETBUF: the number of struct netbufs. (only needed if you use the sequential API, like api_lib.c)

2.1.7.4.8.13 MEMP_NUM_NETCONN

```
#define MEMP_NUM_NETCONN   4
```

MEMP_NUM_NETCONN: the number of struct netconns. (only needed if you use the sequential API, like api_lib.c)

2.1.7.4.8.14 MEMP_NUM_NETDB

```
#define MEMP_NUM_NETDB   1
```

MEMP_NUM_NETDB: the number of concurrently running lwip_addrinfo() calls (before freeing the corresponding memory using lwip_freeaddrinfo()).

2.1.7.4.8.15 MEMP_NUM_NETIFAPI_MSG

```
#define MEMP_NUM_NETIFAPI_MSG   MEMP_NUM_TCPIP_MSG_API
```

MEMP_NUM_NETIFAPI_MSG: the number of concurrently active calls to the netifapi functions

2.1.7.4.8.16 MEMP_NUM_PBUF

```
#define MEMP_NUM_PBUF   16
```

MEMP_NUM_PBUF: the number of memp struct pbufs (used for PBUF_ROM and PBUF_REF). If the application sends a lot of data out of ROM (or other static memory), this should be set high.

2.1.7.4.8.17 MEMP_NUM_RAW_PCB

```
#define MEMP_NUM_RAW_PCB   4
```

MEMP_NUM_RAW_PCB: Number of raw connection PCBs (requires the LWIP_RAW option)

2.1.7.4.8.18 MEMP_NUM_REASSDATA

```
#define MEMP_NUM_REASSDATA   5
```

MEMP_NUM_REASSDATA: the number of IP packets simultaneously queued for reassembly (whole packets, not fragments!)

2.1.7.4.8.19 MEMP_NUM_SELECT_CB

```
#define MEMP_NUM_SELECT_CB   4
```

MEMP_NUM_SELECT_CB: the number of struct lwip_select_cb. (Only needed if you have LWIP_MPU_COMPATIBLE==1 and use the socket API. In that case, you need one per thread calling lwip_select.)

2.1.7.4.8.20 MEMP_NUM_SOCKET_SETGETSOCKOPT_DATA

```
#define MEMP_NUM_SOCKET_SETGETSOCKOPT_DATA   MEMP_NUM_TCPIP_MSG_API
```

MEMP_NUM_SOCKET_SETGETSOCKOPT_DATA: the number of concurrently active calls to getsockopt/setsockopt

2.1.7.4.8.21 MEMP_NUM_SYS_TIMEOUT

```
#define MEMP_NUM_SYS_TIMEOUT   LWIP_NUM_SYS_TIMEOUT_INTERNAL
```

MEMP_NUM_SYS_TIMEOUT: the number of simultaneously active timeouts. The default number of timeouts is calculated here for all enabled modules. The formula expects settings to be either '0' or '1'.

2.1.7.4.8.22 MEMP_NUM_TCP_PCB

```
#define MEMP_NUM_TCP_PCB   5
```

MEMP_NUM_TCP_PCB: the number of simultaneously active TCP connections. (requires the LWIP_TCP option)

2.1.7.4.8.23 MEMP_NUM_TCP_PCB_LISTEN

```
#define MEMP_NUM_TCP_PCB_LISTEN   8
```

MEMP_NUM_TCP_PCB_LISTEN: the number of listening TCP connections. (requires the LWIP_TCP option)

2.1.7.4.8.24 MEMP_NUM_TCP_SEG

```
#define MEMP_NUM_TCP_SEG   16
```

MEMP_NUM_TCP_SEG: the number of simultaneously queued TCP segments. (requires the LWIP_TCP option)

2.1.7.4.8.25 MEMP_NUM_TCPIP_MSG_API

```
#define MEMP_NUM_TCPIP_MSG_API   8
```

MEMP_NUM_TCPIP_MSG_API: the number of struct tcpip_msg, which are used for callback/timeout API communication. (only needed if you use tcpip.c)

2.1.7.4.8.26 MEMP_NUM_TCPIP_MSG_INPKT

```
#define MEMP_NUM_TCPIP_MSG_INPKT   8
```

MEMP_NUM_TCPIP_MSG_INPKT: the number of struct tcpip_msg, which are used for incoming packets. (only needed if you use tcpip.c)

2.1.7.4.8.27 MEMP_NUM_UDP_PCB

```
#define MEMP_NUM_UDP_PCB   4
```

MEMP_NUM_UDP_PCB: the number of UDP protocol control blocks. One per active UDP "connection". (requires the LWIP_UDP option)

2.1.7.4.8.28 PBUF_POOL_SIZE

```
#define PBUF_POOL_SIZE   16
```

PBUF_POOL_SIZE: the number of buffers in the pbuf pool.

2.1.7.4.9 SNMP MIB2 callbacks

2.1.7.4.9.1 Macros

- #define LWIP_MIB2_CALLBACKS 0

2.1.7.4.9.2 Detailed Description

2.1.7.4.9.3 Macro Definition Documentation

2.1.7.4.9.4 LWIP_MIB2_CALLBACKS

```
#define LWIP_MIB2_CALLBACKS   0
```

LWIP_MIB2_CALLBACKS==1: Turn on SNMP MIB2 callbacks. Turn this on to get callbacks needed to implement MIB2. Usually MIB2_STATS should be enabled, too.

2.1.7.4.10 Multicast

2.1.7.4.10.1 Macros

- #define LWIP_MULTICAST_TX_OPTIONS ((LWIP_IGMP || LWIP_IPV6_MLD) && (LWIP_UDP || LWIP_RAW))

2.1.7.4.10.2 Detailed Description

2.1.7.4.10.3 Macro Definition Documentation

2.1.7.4.10.4 LWIP_MULTICAST_TX_OPTIONS

```
#define LWIP_MULTICAST_TX_OPTIONS   ((LWIP_IGMP || LWIP_IPV6_MLD) && (LWIP_UDP || LWIP_RAW
```

LWIP_MULTICAST_TX_OPTIONS==1: Enable multicast TX support like the socket options IP_MULTICAST_TTL/IP_MULTICAST as well as (currently only) core support for the corresponding IPv6 options.

2.1.7.4.11 Threading

2.1.7.4.11.1 Macros

- #define TCPIP_THREAD_NAME "tcpip_thread"
- #define TCPIP_THREAD_STACKSIZE 0
- #define TCPIP_THREAD_PRIO 1
- #define TCPIP_MBOX_SIZE 0

- #define LWIP_TCPIP_THREAD_ALIVE()
- #define SLIPIF_THREAD_NAME "slipif_loop"
- #define SLIPIF_THREAD_STACKSIZE 0
- #define SLIPIF_THREAD_PRIO 1
- #define DEFAULT_THREAD_NAME "lwIP"
- #define DEFAULT_THREAD_STACKSIZE 0
- #define DEFAULT_THREAD_PRIO 1
- #define DEFAULT_RAW_RECVMBOX_SIZE 0
- #define DEFAULT_UDP_RECVMBOX_SIZE 0
- #define DEFAULT_TCP_RECVMBOX_SIZE 0
- #define DEFAULT_ACCEPTMBOX_SIZE 0

2.1.7.4.11.2 Detailed Description

2.1.7.4.11.3 Macro Definition Documentation

2.1.7.4.11.4 DEFAULT_ACCEPTMBOX_SIZE

```
#define DEFAULT_ACCEPTMBOX_SIZE   0
```

DEFAULT_ACCEPTMBOX_SIZE: The mailbox size for the incoming connections. The queue size value itself is platform-dependent, but is passed to sys_mbox_new() when the acceptmbox is created.

2.1.7.4.11.5 DEFAULT_RAW_RECVMBOX_SIZE

```
#define DEFAULT_RAW_RECVMBOX_SIZE   0
```

DEFAULT_RAW_RECVMBOX_SIZE: The mailbox size for the incoming packets on a NETCONN_RAW. The queue size value itself is platform-dependent, but is passed to sys_mbox_new() when the recvmbox is created.

2.1.7.4.11.6 DEFAULT_TCP_RECVMBOX_SIZE

```
#define DEFAULT_TCP_RECVMBOX_SIZE   0
```

DEFAULT_TCP_RECVMBOX_SIZE: The mailbox size for the incoming packets on a NETCONN_TCP. The queue size value itself is platform-dependent, but is passed to sys_mbox_new() when the recvmbox is created.

2.1.7.4.11.7 DEFAULT_THREAD_NAME

```
#define DEFAULT_THREAD_NAME   "lwIP"
```

DEFAULT_THREAD_NAME: The name assigned to any other lwIP thread.

2.1.7.4.11.8 DEFAULT_THREAD_PRIO

```
#define DEFAULT_THREAD_PRIO   1
```

DEFAULT_THREAD_PRIO: The priority assigned to any other lwIP thread. The priority value itself is platform-dependent, but is passed to sys_thread_new() when the thread is created.

2.1.7.4.11.9 DEFAULT_THREAD_STACKSIZE

```
#define DEFAULT_THREAD_STACKSIZE   0
```

DEFAULT_THREAD_STACKSIZE: The stack size used by any other lwIP thread. The stack size value itself is platform-dependent, but is passed to sys_thread_new() when the thread is created.

2.1.7.4.11.10 DEFAULT_UDP_RECVMBOX_SIZE

```
#define DEFAULT_UDP_RECVMBOX_SIZE   0
```

DEFAULT_UDP_RECVMBOX_SIZE: The mailbox size for the incoming packets on a NETCONN_UDP. The queue size value itself is platform-dependent, but is passed to sys_mbox_new() when the recvmbox is created.

2.1.7.4.11.11 LWIP_TCPIP_THREAD_ALIVE

```
#define LWIP_TCPIP_THREAD_ALIVE( )
```

Define this to something that triggers a watchdog. This is called from tcpip_thread after processing a message.

2.1.7.4.11.12 SLIPIF_THREAD_NAME

```
#define SLIPIF_THREAD_NAME   "slipif_loop"
```

SLIPIF_THREAD_NAME: The name assigned to the slipif_loop thread.

2.1.7.4.11.13 SLIPIF_THREAD_PRIO

```
#define SLIPIF_THREAD_PRIO   1
```

SLIPIF_THREAD_PRIO: The priority assigned to the slipif_loop thread. The priority value itself is platform-dependent, but is passed to sys_thread_new() when the thread is created.

2.1.7.4.11.14 SLIPIF_THREAD_STACKSIZE

```
#define SLIPIF_THREAD_STACKSIZE   0
```

SLIP_THREAD_STACKSIZE: The stack size used by the slipif_loop thread. The stack size value itself is platform-dependent, but is passed to sys_thread_new() when the thread is created.

2.1.7.4.11.15 TCPIP_MBOX_SIZE

```
#define TCPIP_MBOX_SIZE   0
```

TCPIP_MBOX_SIZE: The mailbox size for the tcpip thread messages The queue size value itself is platform-dependent, but is passed to sys_mbox_new() when tcpip_init is called.

2.1.7.4.11.16 TCPIP_THREAD_NAME

```
#define TCPIP_THREAD_NAME   "tcpip_thread"
```

TCPIP_THREAD_NAME: The name assigned to the main tcpip thread.

2.1.7.4.11.17 TCPIP_THREAD_PRIO

```
#define TCPIP_THREAD_PRIO   1
```

TCPIP_THREAD_PRIO: The priority assigned to the main tcpip thread. The priority value itself is platform-dependent, but is passed to sys_thread_new() when the thread is created.

2.1.7.4.11.18 TCPIP_THREAD_STACKSIZE

```
#define TCPIP_THREAD_STACKSIZE    0
```

TCPIP_THREAD_STACKSIZE: The stack size used by the main tcpip thread. The stack size value itself is platform-dependent, but is passed to sys_thread_new() when the thread is created.

2.1.7.4.12 Checksum

2.1.7.4.12.1 Macros

- #define LWIP_CHECKSUM_CTRL_PER_NETIF 0
- #define CHECKSUM_GEN_IP 1
- #define CHECKSUM_GEN_UDP 1
- #define CHECKSUM_GEN_TCP 1
- #define CHECKSUM_GEN_ICMP 1
- #define CHECKSUM_GEN_ICMP6 1
- #define CHECKSUM_CHECK_IP 1
- #define CHECKSUM_CHECK_UDP 1
- #define CHECKSUM_CHECK_TCP 1
- #define CHECKSUM_CHECK_ICMP 1
- #define CHECKSUM_CHECK_ICMP6 1
- #define LWIP_CHECKSUM_ON_COPY 0

2.1.7.4.12.2 Detailed Description

2.1.7.4.12.3 Macro Definition Documentation

2.1.7.4.12.4 CHECKSUM_CHECK_ICMP

```
#define CHECKSUM_CHECK_ICMP    1
```

CHECKSUM_CHECK_ICMP==1: Check checksums in software for incoming ICMP packets.

2.1.7.4.12.5 CHECKSUM_CHECK_ICMP6

```
#define CHECKSUM_CHECK_ICMP6    1
```

CHECKSUM_CHECK_ICMP6==1: Check checksums in software for incoming ICMPv6 packets

2.1.7.4.12.6 CHECKSUM_CHECK_IP

```
#define CHECKSUM_CHECK_IP    1
```

CHECKSUM_CHECK_IP==1: Check checksums in software for incoming IP packets.

2.1.7.4.12.7 CHECKSUM_CHECK_TCP

```
#define CHECKSUM_CHECK_TCP    1
```

CHECKSUM_CHECK_TCP==1: Check checksums in software for incoming TCP packets.

2.1.7.4.12.8 CHECKSUM_CHECK_UDP

```
#define CHECKSUM_CHECK_UDP   1
```

CHECKSUM_CHECK_UDP==1: Check checksums in software for incoming UDP packets.

2.1.7.4.12.9 CHECKSUM_GEN_ICMP

```
#define CHECKSUM_GEN_ICMP   1
```

CHECKSUM_GEN_ICMP==1: Generate checksums in software for outgoing ICMP packets.

2.1.7.4.12.10 CHECKSUM_GEN_ICMP6

```
#define CHECKSUM_GEN_ICMP6   1
```

CHECKSUM_GEN_ICMP6==1: Generate checksums in software for outgoing ICMP6 packets.

2.1.7.4.12.11 CHECKSUM_GEN_IP

```
#define CHECKSUM_GEN_IP   1
```

CHECKSUM_GEN_IP==1: Generate checksums in software for outgoing IP packets.

2.1.7.4.12.12 CHECKSUM_GEN_TCP

```
#define CHECKSUM_GEN_TCP   1
```

CHECKSUM_GEN_TCP==1: Generate checksums in software for outgoing TCP packets.

2.1.7.4.12.13 CHECKSUM_GEN_UDP

```
#define CHECKSUM_GEN_UDP   1
```

CHECKSUM_GEN_UDP==1: Generate checksums in software for outgoing UDP packets.

2.1.7.4.12.14 LWIP_CHECKSUM_CTRL_PER_NETIF

```
#define LWIP_CHECKSUM_CTRL_PER_NETIF   0
```

LWIP_CHECKSUM_CTRL_PER_NETIF==1: Checksum generation/check can be enabled/disabled per netif. ATTENTION: if enabled, the CHECKSUM_GEN_* and CHECKSUM_CHECK_* defines must be enabled!

2.1.7.4.12.15 LWIP_CHECKSUM_ON_COPY

```
#define LWIP_CHECKSUM_ON_COPY   0
```

LWIP_CHECKSUM_ON_COPY==1: Calculate checksum when copying data from application buffers to pbufs.

2.1.7.4.13 Hooks

2.1.7.4.13.1 Macros

- #define LWIP_HOOK_FILENAME "path/to/my/lwip_hooks.h"
- #define LWIP_HOOK_TCP_ISN(local_ip, local_port, remote_ip, remote_port)
- #define LWIP_HOOK_TCP_INPACKET_PCB(pcb, hdr, optlen, opt1len, opt2, p)

- #define LWIP_HOOK_TCP_OUT_TCPOPT_LENGTH(pcb, internal_len)
- #define LWIP_HOOK_TCP_OUT_ADD_TCPOPTS(p, hdr, pcb, opts)
- #define LWIP_HOOK_IP4_INPUT(pbuf, input_netif)
- #define LWIP_HOOK_IP4_ROUTE()
- #define LWIP_HOOK_IP4_ROUTE_SRC(src, dest)
- #define LWIP_HOOK_IP4_CANFORWARD(src, dest)
- #define LWIP_HOOK_ETHARP_GET_GW(netif, dest)
- #define LWIP_HOOK_IP6_INPUT(pbuf, input_netif)
- #define LWIP_HOOK_IP6_ROUTE(src, dest)
- #define LWIP_HOOK_ND6_GET_GW(netif, dest)
- #define LWIP_HOOK_VLAN_CHECK(netif, eth_hdr, vlan_hdr)
- #define LWIP_HOOK_VLAN_SET(netif, p, src, dst, eth_type)
- #define LWIP_HOOK_MEMP_AVAILABLE(memp_t_type)
- #define LWIP_HOOK_UNKNOWN_ETH_PROTOCOL(pbuf, netif)
- #define LWIP_HOOK_DHCP_APPEND_OPTIONS(netif, dhcp, state, msg, msg_type, options_len_ptr)
- #define LWIP_HOOK_DHCP_PARSE_OPTION(netif, dhcp, state, msg, msg_type, option, len, pbuf, offset)
- #define LWIP_HOOK_DHCP6_APPEND_OPTIONS(netif, dhcp6, state, msg, msg_type, options_len_ptr, max_len)
- #define LWIP_HOOK_SOCKETS_SETSOCKOPT(s, sock, level, optname, optval, optlen, err)
- #define LWIP_HOOK_SOCKETS_GETSOCKOPT(s, sock, level, optname, optval, optlen, err)
- #define LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE(name, addr, addrtype, err)

2.1.7.4.13.2 Detailed Description

Hooks are undefined by default, define them to a function if you need them.

2.1.7.4.13.3 Macro Definition Documentation

2.1.7.4.13.4 LWIP_HOOK_DHCP6_APPEND_OPTIONS

```
#define LWIP_HOOK_DHCP6_APPEND_OPTIONS( netif, dhcp6, state, msg, msg_type, options_len_pt
max_len)
```

LWIP_HOOK_DHCP6_APPEND_OPTIONS(netif, dhcp6, state, msg, msg_type, options_len_ptr, max_len): Called from various dhcp6 functions when sending a DHCP6 message. This hook is called just before the DHCP6 message is sent, so the options are at the end of a DHCP6 message. Signature:

```
void my_hook(struct netif *netif, struct dhcp6 *dhcp, u8_t state, struct dhcp6_msg ↩
    *msg,
            u8_t msg_type, u16_t *options_len_ptr);
```

Arguments:

- netif: struct netif that the packet will be sent through
- dhcp6: struct dhcp6 on that netif

- state: current dhcp6 state (dhcp6_state_enum_t as an u8_t)
- msg: struct dhcp6_msg that will be sent
- msg_type: dhcp6 message type to be sent (u8_t)
- options_len_ptr: pointer to the current length of options in the dhcp6_msg "msg" (must be increased when options are added!)

Options need to appended like this: u8_t *options = (u8_t *)(msg + 1); LWIP_ASSERT("dhcp option overflow", sizeof(struct dhcp6_msg) + *options_len_ptr + newoptlen <= max_len); options[(*options_len_ptr)++] = <option_data>; [...]

2.1.7.4.13.5 LWIP_HOOK_DHCP_APPEND_OPTIONS

```
#define LWIP_HOOK_DHCP_APPEND_OPTIONS( netif, dhcp, state, msg, msg_type, options_len_ptr)
```

LWIP_HOOK_DHCP_APPEND_OPTIONS(netif, dhcp, state, msg, msg_type, options_len_ptr): Called from various dhcp functions when sending a DHCP message. This hook is called just before the DHCP message trailer is added, so the options are at the end of a DHCP message. Signature:

```
void my_hook(struct netif *netif, struct dhcp *dhcp, u8_t state, struct dhcp_msg * ↩
   msg,
             u8_t msg_type, u16_t *options_len_ptr);
```

Arguments:

- netif: struct netif that the packet will be sent through
- dhcp: struct dhcp on that netif
- state: current dhcp state (dhcp_state_enum_t as an u8_t)
- msg: struct dhcp_msg that will be sent
- msg_type: dhcp message type to be sent (u8_t)
- options_len_ptr: pointer to the current length of options in the dhcp_msg "msg" (must be increased when options are added!)

Options need to appended like this: LWIP_ASSERT("dhcp option overflow", *options_len_ptr + option_len + 2 <= DHCP_OPTIONS_L
msg->options[(*options_len_ptr)++] = <option_number>; msg->options[(*options_len_ptr)++] = <option_len>; msg->options[(*option
= <option_bytes>; [...]

2.1.7.4.13.6 LWIP_HOOK_DHCP_PARSE_OPTION

```
#define LWIP_HOOK_DHCP_PARSE_OPTION( netif, dhcp, state, msg, msg_type, option, len, pbuf,
offset)
```

LWIP_HOOK_DHCP_PARSE_OPTION(netif, dhcp, state, msg, msg_type, option, len, pbuf, option_value_offset): Called from dhcp_parse_reply when receiving a DHCP message. This hook is called for every option in the received message that is not handled internally. Signature:

```
void my_hook(struct netif *netif, struct dhcp *dhcp, u8_t state, struct dhcp_msg * ↩
   msg,
             u8_t msg_type, u8_t option, u8_t option_len, struct pbuf *pbuf, u16_t ↩
                option_value_offset);
```

Arguments:

- netif: struct netif that the packet will be sent through
- dhcp: struct dhcp on that netif

- state: current dhcp state (dhcp_state_enum_t as an u8_t)
- msg: struct dhcp_msg that was received
- msg_type: dhcp message type received (u8_t, ATTENTION: only valid after the message type option has been parsed!)
- option: option value (u8_t)
- len: option data length (u8_t)
- pbuf: pbuf where option data is contained
- option_value_offset: offset in pbuf where option data begins

A nice way to get the option contents is pbuf_get_contiguous(): u8_t buf[32]; u8_t *ptr = (u8_t*)pbuf_get_contiguous(p, buf, sizeof(buf), LWIP_MIN(option_len, sizeof(buf)), offset);

2.1.7.4.13.7 LWIP_HOOK_ETHARP_GET_GW

```
#define LWIP_HOOK_ETHARP_GET_GW( netif, dest)
```

LWIP_HOOK_ETHARP_GET_GW(netif, dest): Called from etharp_output() (IPv4) Signature:

```
const ip4_addr_t *my_hook(struct netif *netif, const ip4_addr_t *dest);
```

Arguments:

- netif: the netif used for sending
- dest: the destination IPv4 address Return values:
- the IPv4 address of the gateway to handle the specified destination IPv4 address
- NULL, in which case the netif's default gateway is used

The returned address MUST be directly reachable on the specified netif! This function is meant to implement advanced IPv4 routing together with LWIP_HOOK_IP4_ROUTE(). The actual routing/gateway table implementation is not part of lwIP but can e.g. be hidden in the netif's state argument.

2.1.7.4.13.8 LWIP_HOOK_FILENAME

```
#define LWIP_HOOK_FILENAME   "path/to/my/lwip_hooks.h"
```

LWIP_HOOK_FILENAME: Custom filename to #include in files that provide hooks. Declare your hook function prototypes in there, you may also #include all headers providing data types that are need in this file.

2.1.7.4.13.9 LWIP_HOOK_IP4_CANFORWARD

```
#define LWIP_HOOK_IP4_CANFORWARD( src, dest)
```

LWIP_HOOK_IP4_CANFORWARD(src, dest): Check if an IPv4 can be forwarded - called from: ip4_input() -> ip4_forward() -> ip4_canforward() (IPv4)

- source address is available via ip4_current_src_addr()
- calling an output function in this context (e.g. multicast router) is allowed Signature:

  ```
  int my_hook(struct pbuf *p, u32_t dest_addr_hostorder);
  ```

  Arguments:

- p: packet to forward
- dest: destination IPv4 address Returns values:
- 1: forward
- 0: don't forward
- -1: no decision. In that case, ip4_canforward() continues as normal.

2.1.7.4.13.10 LWIP_HOOK_IP4_INPUT

```
#define LWIP_HOOK_IP4_INPUT( pbuf, input_netif)
```

LWIP_HOOK_IP4_INPUT(pbuf, input_netif): Called from ip_input() (IPv4) Signature:

```
int my_hook(struct pbuf *pbuf, struct netif *input_netif);
```

Arguments:

- pbuf: received struct pbuf passed to ip_input()
- input_netif: struct netif on which the packet has been received Return values:
- 0: Hook has not consumed the packet, packet is processed as normal
- != 0: Hook has consumed the packet. If the hook consumed the packet, 'pbuf' is in the responsibility of the hook (i.e. free it when done).

2.1.7.4.13.11 LWIP_HOOK_IP4_ROUTE

```
#define LWIP_HOOK_IP4_ROUTE( )
```

LWIP_HOOK_IP4_ROUTE(dest): Called from ip_route() (IPv4) Signature:

```
struct netif *my_hook(const ip4_addr_t *dest);
```

Arguments:

- dest: destination IPv4 address Returns values:
- the destination netif
- NULL if no destination netif is found. In that case, ip_route() continues as normal.

2.1.7.4.13.12 LWIP_HOOK_IP4_ROUTE_SRC

```
#define LWIP_HOOK_IP4_ROUTE_SRC( src, dest)
```

LWIP_HOOK_IP4_ROUTE_SRC(src, dest): Source-based routing for IPv4 - called from ip_route() (IPv4) Signature:

```
struct netif *my_hook(const ip4_addr_t *src, const ip4_addr_t *dest);
```

Arguments:

- src: local/source IPv4 address
- dest: destination IPv4 address Returns values:
- the destination netif
- NULL if no destination netif is found. In that case, ip_route() continues as normal.

2.1.7.4.13.13 LWIP_HOOK_IP6_INPUT

```
#define LWIP_HOOK_IP6_INPUT( pbuf, input_netif)
```

LWIP_HOOK_IP6_INPUT(pbuf, input_netif): Called from ip6_input() (IPv6) Signature:

```
int my_hook(struct pbuf *pbuf, struct netif *input_netif);
```

Arguments:

- pbuf: received struct pbuf passed to ip6_input()
- input_netif: struct netif on which the packet has been received Return values:
- 0: Hook has not consumed the packet, packet is processed as normal
- != 0: Hook has consumed the packet. If the hook consumed the packet, 'pbuf' is in the responsibility of the hook (i.e. free it when done).

2.1.7.4.13.14 LWIP_HOOK_IP6_ROUTE

```
#define LWIP_HOOK_IP6_ROUTE( src, dest)
```

LWIP_HOOK_IP6_ROUTE(src, dest): Called from ip_route() (IPv6) Signature:

```
struct netif *my_hook(const ip6_addr_t *dest, const ip6_addr_t *src);
```

Arguments:

- src: source IPv6 address
- dest: destination IPv6 address Return values:
- the destination netif
- NULL if no destination netif is found. In that case, ip6_route() continues as normal.

2.1.7.4.13.15 LWIP_HOOK_MEMP_AVAILABLE

```
#define LWIP_HOOK_MEMP_AVAILABLE( memp_t_type)
```

LWIP_HOOK_MEMP_AVAILABLE(memp_t_type): Called from memp_free() when a memp pool was empty and an item is now available Signature:

```
void my_hook(memp_t type);
```

2.1.7.4.13.16 LWIP_HOOK_ND6_GET_GW

```
#define LWIP_HOOK_ND6_GET_GW( netif, dest)
```

LWIP_HOOK_ND6_GET_GW(netif, dest): Called from nd6_get_next_hop_entry() (IPv6) Signature:

```
const ip6_addr_t *my_hook(struct netif *netif, const ip6_addr_t *dest);
```

Arguments:

- netif: the netif used for sending
- dest: the destination IPv6 address Return values:

- the IPv6 address of the next hop to handle the specified destination IPv6 address
- NULL, in which case a NDP-discovered router is used instead

The returned address MUST be directly reachable on the specified netif! This function is meant to implement advanced IPv6 routing together with LWIP_HOOK_IP6_ROUTE(). The actual routing/gateway table implementation is not part of lwIP but can e.g. be hidden in the netif's state argument.

2.1.7.4.13.17 LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE

```
#define LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE( name, addr, addrtype, err)
```

LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE(name, addr, addrtype, err) Called from netconn APIs (not usable with callback apps) allowing an external DNS resolver (which uses sequential API) to handle the query. Signature:

```
int my_hook(const char *name, ip_addr_t *addr, u8_t addrtype, err_t *err)
```

Arguments:

- name: hostname to resolve
- addr: output host address
- addrtype: type of address to query
- err: output error Return values:
- 0: Hook has not consumed hostname query, query continues into DNS module
- != 0: Hook has consumed the query

err must also be checked to determine if the hook consumed the query, but the query failed

2.1.7.4.13.18 LWIP_HOOK_SOCKETS_GETSOCKOPT

```
#define LWIP_HOOK_SOCKETS_GETSOCKOPT( s, sock, level, optname, optval, optlen, err)
```

LWIP_HOOK_SOCKETS_GETSOCKOPT(s, sock, level, optname, optval, optlen, err) Called from socket API to implement getsockopt() for options not provided by lwIP. Core lock is held when this hook is called. Signature:

```
int my_hook(int s, struct lwip_sock *sock, int level, int optname, void *optval,  ↩
    socklen_t *optlen, int *err)
```

Arguments:

- s: socket file descriptor
- sock: internal socket descriptor (see lwip/priv/sockets_priv.h)
- level: protocol level at which the option resides
- optname: option to get
- optval: value to get
- optlen: size of optval
- err: output error Return values:
- 0: Hook has not consumed the option, code continues as normal (to internal options)
- != 0: Hook has consumed the option, 'err' is returned

2.1.7.4.13.19 LWIP_HOOK_SOCKETS_SETSOCKOPT

```
#define LWIP_HOOK_SOCKETS_SETSOCKOPT( s, sock, level, optname, optval, optlen, err)
```

LWIP_HOOK_SOCKETS_SETSOCKOPT(s, sock, level, optname, optval, optlen, err) Called from socket API to implement setsockopt() for options not provided by lwIP. Core lock is held when this hook is called. Signature:

```
int my_hook(int s, struct lwip_sock *sock, int level, int optname, const void * ↩
    optval, socklen_t optlen, int *err)
```

Arguments:

- s: socket file descriptor
- sock: internal socket descriptor (see lwip/priv/sockets_priv.h)
- level: protocol level at which the option resides
- optname: option to set
- optval: value to set
- optlen: size of optval
- err: output error Return values:
- 0: Hook has not consumed the option, code continues as normal (to internal options)
- != 0: Hook has consumed the option, 'err' is returned

2.1.7.4.13.20 LWIP_HOOK_TCP_INPACKET_PCB

```
#define LWIP_HOOK_TCP_INPACKET_PCB( pcb, hdr, optlen, opt1len, opt2, p)
```

LWIP_HOOK_TCP_INPACKET_PCB: Hook for intercepting incoming packets before they are passed to a pcb. This allows updating some state or even dropping a packet. Signature:

```
err_t my_hook_tcp_inpkt(struct tcp_pcb *pcb, struct tcp_hdr *hdr, u16_t optlen,  ↩
    u16_t opt1len, u8_t *opt2, struct pbuf *p);
```

Arguments:

- pcb: tcp_pcb selected for input of this packet (ATTENTION: this may be struct tcp_pcb_listen if pcb->state == LISTEN)
- hdr: pointer to tcp header (ATTENTION: tcp options may not be in one piece!)
- optlen: tcp option length
- opt1len: tcp option length 1st part
- opt2: if this is != NULL, tcp options are split among 2 pbufs. In that case, options start at right after the tcp header ('(u8_t*)(hdr + 1)') for the first 'opt1len' bytes and the rest starts at 'opt2'. opt2len can be simply calculated: 'opt2len = optlen - opt1len;'
- p: input packet, p->payload points to application data (that's why tcp hdr and options are passed in separately) Return value:
- ERR_OK: continue input of this packet as normal
- != ERR_OK: drop this packet for input (don't continue input processing)

ATTENTION: don't call any tcp api functions that might change tcp state (pcb state or any pcb lists) from this callback!

2.1.7.4.13.21 LWIP_HOOK_TCP_ISN

```
#define LWIP_HOOK_TCP_ISN( local_ip, local_port, remote_ip, remote_port)
```

LWIP_HOOK_TCP_ISN: Hook for generation of the Initial Sequence Number (ISN) for a new TCP connection. The default lwIP ISN generation algorithm is very basic and may allow for TCP spoofing attacks. This hook provides the means to implement the standardized ISN generation algorithm from RFC 6528 (see contrib/adons/tcp_isn), or any other desired algorithm as a replacement. Called from tcp_connect() and tcp_listen_input() when an ISN is needed for a new TCP connection, if TCP support (LWIP_TCP) is enabled. Signature:

```
u32_t my_hook_tcp_isn(const ip_addr_t* local_ip, u16_t local_port, const ip_addr_t ↩
    * remote_ip, u16_t remote_port);
```

- it may be necessary to use "struct ip_addr" (ip4_addr, ip6_addr) instead of "ip_addr_t" in function declarations Arguments:
- local_ip: pointer to the local IP address of the connection
- local_port: local port number of the connection (host-byte order)
- remote_ip: pointer to the remote IP address of the connection
- remote_port: remote port number of the connection (host-byte order) Return value:
- the 32-bit Initial Sequence Number to use for the new TCP connection.

2.1.7.4.13.22 LWIP_HOOK_TCP_OUT_ADD_TCPOPTS

```
#define LWIP_HOOK_TCP_OUT_ADD_TCPOPTS( p, hdr, pcb, opts)
```

LWIP_HOOK_TCP_OUT_ADD_TCPOPTS: Hook for adding custom options to outgoing tcp segments. Space for these custom options has to be reserved via LWIP_HOOK_TCP_OUT_TCPOPT_LENGTH. Signature:

```
u32_t *my_hook_tcp_out_add_tcpopts(struct pbuf *p, struct tcp_hdr *hdr, const  ↩
    struct tcp_pcb *pcb, u32_t *opts);
```

Arguments:

- p: output packet, p->payload pointing to tcp header, data follows
- hdr: tcp header
- pcb: tcp_pcb that transmits (ATTENTION: this may be NULL or struct tcp_pcb_listen if pcb->state == LISTEN)
- opts: pointer where to add the custom options (there may already be options between the header and these) Return value:
- pointer pointing directly after the inserted options

ATTENTION: don't call any tcp api functions that might change tcp state (pcb state or any pcb lists) from this callback!

2.1.7.4.13.23 LWIP_HOOK_TCP_OUT_TCPOPT_LENGTH

```
#define LWIP_HOOK_TCP_OUT_TCPOPT_LENGTH( pcb, internal_len)
```

LWIP_HOOK_TCP_OUT_TCPOPT_LENGTH: Hook for increasing the size of the options allocated with a tcp header. Together with LWIP_HOOK_TCP_OUT_ADD_TCPOPTS, this can be used to add custom options to outgoing tcp segments. Signature:

```
u8_t my_hook_tcp_out_tcpopt_length(const struct tcp_pcb *pcb, u8_t  ↩
    internal_option_length);
```

Arguments:

- pcb: tcp_pcb that transmits (ATTENTION: this may be NULL or struct tcp_pcb_listen if pcb->state == LISTEN)
- internal_option_length: tcp option length used by the stack internally Return value:
- a number of bytes to allocate for tcp options (internal_option_length <= ret <= 40)

ATTENTION: don't call any tcp api functions that might change tcp state (pcb state or any pcb lists) from this callback!

2.1.7.4.13.24 LWIP_HOOK_UNKNOWN_ETH_PROTOCOL

```
#define LWIP_HOOK_UNKNOWN_ETH_PROTOCOL( pbuf, netif)
```

LWIP_HOOK_UNKNOWN_ETH_PROTOCOL(pbuf, netif): Called from ethernet_input() when an unknown eth type is encountered. Signature:

```
err_t my_hook(struct pbuf* pbuf, struct netif* netif);
```

Arguments:

- p: rx packet with unknown eth type
- netif: netif on which the packet has been received Return values:
- ERR_OK if packet is accepted (hook function now owns the pbuf)
- any error code otherwise (pbuf is freed)

Payload points to ethernet header!

2.1.7.4.13.25 LWIP_HOOK_VLAN_CHECK

```
#define LWIP_HOOK_VLAN_CHECK( netif, eth_hdr, vlan_hdr)
```

LWIP_HOOK_VLAN_CHECK(netif, eth_hdr, vlan_hdr): Called from ethernet_input() if VLAN support is enabled Signature:

```
int my_hook(struct netif *netif, struct eth_hdr *eth_hdr, struct eth_vlan_hdr * ↩
    vlan_hdr);
```

Arguments:

- netif: struct netif on which the packet has been received
- eth_hdr: struct eth_hdr of the packet
- vlan_hdr: struct eth_vlan_hdr of the packet Return values:
- 0: Packet must be dropped.
- != 0: Packet must be accepted.

2.1.7.4.13.26 LWIP_HOOK_VLAN_SET

```
#define LWIP_HOOK_VLAN_SET( netif, p, src, dst, eth_type)
```

LWIP_HOOK_VLAN_SET: Hook can be used to set prio_vid field of vlan_hdr. If you need to store data on per-netif basis to implement this callback, see Client data handling. Called from ethernet_output() if VLAN support (ETHARP_SUPPORT_VLAN) is enabled. Signature:

```
s32_t my_hook_vlan_set(struct netif* netif, struct pbuf* pbuf, const struct ↩
    eth_addr* src, const struct eth_addr* dst, u16_t eth_type);
```

Arguments:

- netif: struct netif that the packet will be sent through
- p: struct pbuf packet to be sent
- src: source eth address
- dst: destination eth address
- eth_type: ethernet type to packet to be sent

Return values:

- <0: Packet shall not contain VLAN header.
- 0 <= return value <= 0xFFFF: Packet shall contain VLAN header. Return value is prio_vid in host byte order.

2.1.7.5 Callback-style APIs

2.1.7.5.1 Modules

- RAW
- DNS
- UDP
- TCP

2.1.7.5.2 Detailed Description

2.1.7.5.3 RAW

2.1.7.5.3.1 Macros

- #define LWIP_RAW 0
- #define RAW_TTL IP_DEFAULT_TTL

2.1.7.5.3.2 Detailed Description

2.1.7.5.3.3 Macro Definition Documentation

2.1.7.5.3.4 LWIP_RAW

```
#define LWIP_RAW   0
```

LWIP_RAW==1: Enable application layer to hook into the IP layer itself.

2.1.7.5.3.5 RAW_TTL

```
#define RAW_TTL   IP_DEFAULT_TTL
```

LWIP_RAW==1: Enable application layer to hook into the IP layer itself.

2.1.7.5.4 DNS

2.1.7.5.4.1 Macros

- #define LWIP_DNS 0
- #define DNS_TABLE_SIZE 4
- #define DNS_MAX_NAME_LENGTH 256
- #define DNS_MAX_SERVERS 2
- #define DNS_MAX_RETRIES 4
- #define DNS_DOES_NAME_CHECK 1
- #define LWIP_DNS_SECURE (LWIP_DNS_SECURE_RAND_XID | LWIP_DNS_SECURE_NO_MULTIPLE_OUTSTANDING | LWIP_DNS_SECURE_RAND_SRC_PORT)
- #define DNS_LOCAL_HOSTLIST 0
- #define DNS_LOCAL_HOSTLIST_IS_DYNAMIC 0
- #define LWIP_DNS_SUPPORT_MDNS_QUERIES 0

2.1.7.5.4.2 Detailed Description

2.1.7.5.4.3 Macro Definition Documentation

2.1.7.5.4.4 DNS_DOES_NAME_CHECK

```
#define DNS_DOES_NAME_CHECK   1
```

DNS do a name checking between the query and the response.

2.1.7.5.4.5 DNS_LOCAL_HOSTLIST

```
#define DNS_LOCAL_HOSTLIST   0
```

DNS_LOCAL_HOSTLIST: Implements a local host-to-address list. If enabled, you have to define an initializer: #define DNS_LOCAL_HOSTLIST_INIT {DNS_LOCAL_HOSTLIST_ELEM("host_ip4", IPADDR4_INIT_BYTES(1,2,3,4)), \ DNS_LOCAL IPADDR6_INIT_HOST(123, 234, 345, 456)}

Instead, you can also use an external function: #define DNS_LOOKUP_LOCAL_EXTERN(name, namelen, addr, dns_addrtype) my_lookup_function(name, namelen, addr, dns_addrtype) with function signature: extern err_t my_lookup_function(const char *name, size_t namelen, ip_addr_t *addr, u8_t dns_addrtype) that looks up the IP address and returns ERR_OK if found (LWIP_DNS_AD is passed in dns_addrtype).

2.1.7.5.4.6 DNS_LOCAL_HOSTLIST_IS_DYNAMIC

```
#define DNS_LOCAL_HOSTLIST_IS_DYNAMIC   0
```

If this is turned on, the local host-list can be dynamically changed at runtime.

2.1.7.5.4.7 DNS_MAX_NAME_LENGTH

```
#define DNS_MAX_NAME_LENGTH   256
```

DNS maximum host name length supported in the name table.

2.1.7.5.4.8 DNS_MAX_RETRIES

```
#define DNS_MAX_RETRIES   4
```

DNS maximum number of retries when asking for a name, before "timeout".

2.1.7.5.4.9 DNS_MAX_SERVERS

```
#define DNS_MAX_SERVERS   2
```

The maximum of DNS servers The first server can be initialized automatically by defining DNS_SERVER_ADDRESS(ipaddr), where 'ipaddr' is an 'ip_addr_t*'

2.1.7.5.4.10 DNS_TABLE_SIZE

```
#define DNS_TABLE_SIZE   4
```

DNS maximum number of entries to maintain locally.

2.1.7.5.4.11 LWIP_DNS

```
#define LWIP_DNS   0
```

LWIP_DNS==1: Turn on DNS module. UDP must be available for DNS transport.

2.1.7.5.4.12 LWIP_DNS_SECURE

```
#define LWIP_DNS_SECURE   (LWIP_DNS_SECURE_RAND_XID | LWIP_DNS_SECURE_NO_MULTIPLE_OUTSTAND
| LWIP_DNS_SECURE_RAND_SRC_PORT)
```

LWIP_DNS_SECURE: controls the security level of the DNS implementation Use all DNS security features by default. This is overridable but should only be needed by very small targets or when using against non standard DNS servers.

2.1.7.5.4.13 LWIP_DNS_SUPPORT_MDNS_QUERIES

```
#define LWIP_DNS_SUPPORT_MDNS_QUERIES   0
```

Set this to 1 to enable querying ".local" names via mDNS using a One-Shot Multicast DNS Query

2.1.7.5.5 UDP

2.1.7.5.5.1 Macros

- #define LWIP_UDP 1
- #define LWIP_UDPLITE 0
- #define UDP_TTL IP_DEFAULT_TTL
- #define LWIP_NETBUF_RECVINFO 0

2.1.7.5.5.2 Detailed Description

2.1.7.5.5.3 Macro Definition Documentation

2.1.7.5.5.4 LWIP_NETBUF_RECVINFO

```
#define LWIP_NETBUF_RECVINFO   0
```

LWIP_NETBUF_RECVINFO==1: append destination addr and port to every netbuf.

2.1.7.5.5.5 LWIP_UDP

```
#define LWIP_UDP   1
```

LWIP_UDP==1: Turn on UDP.

2.1.7.5.5.6 LWIP_UDPLITE

```
#define LWIP_UDPLITE   0
```

LWIP_UDPLITE==1: Turn on UDP-Lite. (Requires LWIP_UDP)

2.1.7.5.5.7 UDP_TTL

```
#define UDP_TTL   IP_DEFAULT_TTL
```

UDP_TTL: Default Time-To-Live value.

2.1.7.5.6 TCP

2.1.7.5.6.1 Macros

- #define LWIP_TCP 1
- #define TCP_TTL IP_DEFAULT_TTL
- #define TCP_WND (4 * TCP_MSS)
- #define TCP_MAXRTX 12
- #define TCP_SYNMAXRTX 6
- #define TCP_QUEUE_OOSEQ LWIP_TCP
- #define LWIP_TCP_SACK_OUT 0
- #define LWIP_TCP_MAX_SACK_NUM 4
- #define TCP_MSS 536
- #define TCP_CALCULATE_EFF_SEND_MSS 1
- #define LWIP_TCP_RTO_TIME 3000
- #define TCP_SND_BUF (2 * TCP_MSS)
- #define TCP_SND_QUEUELEN ((4 * (TCP_SND_BUF) + (TCP_MSS - 1))/(TCP_MSS))
- #define TCP_SNDLOWAT LWIP_MIN(LWIP_MAX(((TCP_SND_BUF)/2), (2 * TCP_MSS) + 1), (TCP_SND_BUF) - 1)
- #define TCP_SNDQUEUELOWAT LWIP_MAX(((TCP_SND_QUEUELEN)/2), 5)
- #define TCP_OOSEQ_MAX_BYTES 0

- #define TCP_OOSEQ_MAX_PBUFS 0
- #define TCP_LISTEN_BACKLOG 0
- #define TCP_DEFAULT_LISTEN_BACKLOG 0xff
- #define TCP_OVERSIZE TCP_MSS
- #define LWIP_TCP_TIMESTAMPS 0
- #define TCP_WND_UPDATE_THRESHOLD LWIP_MIN((TCP_WND / 4), (TCP_MSS * 4))
- #define LWIP_EVENT_API 0
- #define LWIP_WND_SCALE 0
- #define LWIP_TCP_PCB_NUM_EXT_ARGS 0
- #define LWIP_ALTCP 0
- #define LWIP_ALTCP_TLS 0

2.1.7.5.6.2 Detailed Description

2.1.7.5.6.3 Macro Definition Documentation

2.1.7.5.6.4 LWIP_ALTCP

```
#define LWIP_ALTCP   0
```

LWIP_ALTCP==1: enable the altcp API. altcp is an abstraction layer that prevents applications linking against the tcp.h functions but provides the same functionality. It is used to e.g. add SSL/TLS or proxy-connect support to an application written for the tcp callback API without that application knowing the protocol details.

With LWIP_ALTCP==0, applications written against the altcp API can still be compiled but are directly linked against the tcp.h callback API and then cannot use layered protocols.

See Application layered TCP Introduction

2.1.7.5.6.5 LWIP_ALTCP_TLS

```
#define LWIP_ALTCP_TLS   0
```

LWIP_ALTCP_TLS==1: enable TLS support for altcp API. This needs a port of the functions in altcp_tls.h to a TLS library. A port to ARM mbedtls is provided with lwIP, see apps/altcp_tls/ directory and LWIP_ALTCP_TLS_MBEDTLS option.

2.1.7.5.6.6 LWIP_EVENT_API

```
#define LWIP_EVENT_API   0
```

LWIP_EVENT_API and LWIP_CALLBACK_API: Only one of these should be set to 1. LWIP_EVENT_API==1: The user defines lwip_tcp_event() to receive all events (accept, sent, etc) that happen in the system. LWIP_CALLBACK_API==1: The PCB callback function is called directly for the event. This is the default.

2.1.7.5.6.7 LWIP_TCP

```
#define LWIP_TCP   1
```

LWIP_TCP==1: Turn on TCP.

2.1.7.5.6.8 LWIP_TCP_MAX_SACK_NUM

```
#define LWIP_TCP_MAX_SACK_NUM   4
```

LWIP_TCP_MAX_SACK_NUM: The maximum number of SACK values to include in TCP segments. Must be at least 1, but is only used if LWIP_TCP_SACK_OUT is enabled. NOTE: Even though we never send more than 3 or 4 SACK ranges in a single segment (depending on other options), setting this option to values greater than 4 is not pointless. This is basically the max number of SACK ranges we want to keep track of. As new data is delivered, some of the SACK ranges may be removed or merged. In that case some of those older SACK ranges may be used again. The amount of memory used to store SACK ranges is LWIP_TCP_MAX_SACK_NUM * 8 bytes for each TCP PCB.

2.1.7.5.6.9 LWIP_TCP_PCB_NUM_EXT_ARGS

```
#define LWIP_TCP_PCB_NUM_EXT_ARGS   0
```

LWIP_TCP_PCB_NUM_EXT_ARGS: When this is > 0, every tcp pcb (including listen pcb) includes a number of additional argument entries in an array (see tcp_ext_arg_alloc_id)

2.1.7.5.6.10 LWIP_TCP_RTO_TIME

```
#define LWIP_TCP_RTO_TIME   3000
```

LWIP_TCP_RTO_TIME: Initial TCP retransmission timeout (ms). This defaults to 3 seconds as traditionally defined in the TCP protocol. For improving timely recovery on faster networks, this value could be lowered down to 1 second (RFC 6298)

2.1.7.5.6.11 LWIP_TCP_SACK_OUT

```
#define LWIP_TCP_SACK_OUT   0
```

LWIP_TCP_SACK_OUT==1: TCP will support sending selective acknowledgements (SACKs).

2.1.7.5.6.12 LWIP_TCP_TIMESTAMPS

```
#define LWIP_TCP_TIMESTAMPS   0
```

LWIP_TCP_TIMESTAMPS==1: support the TCP timestamp option. The timestamp option is currently only used to help remote hosts, it is not really used locally. Therefore, it is only enabled when a TS option is received in the initial SYN packet from a remote host.

2.1.7.5.6.13 LWIP_WND_SCALE

```
#define LWIP_WND_SCALE   0
```

LWIP_WND_SCALE and TCP_RCV_SCALE: Set LWIP_WND_SCALE to 1 to enable window scaling. Set TCP_RCV_SCALE to the desired scaling factor (shift count in the range of [0..14]). When LWIP_WND_SCALE is enabled but TCP_RCV_SCALE is 0, we can use a large send window while having a small receive window only.

2.1.7.5.6.14 TCP_CALCULATE_EFF_SEND_MSS

```
#define TCP_CALCULATE_EFF_SEND_MSS   1
```

TCP_CALCULATE_EFF_SEND_MSS: "The maximum size of a segment that TCP really sends, the 'effective send MSS,' MUST be the smaller of the send MSS (which reflects the available reassembly buffer size at the remote host) and the largest size permitted by the IP layer" (RFC 1122) Setting this to 1 enables code that checks TCP_MSS against the MTU of the netif used for a connection and limits the MSS if it would be too big otherwise.

2.1.7.5.6.15 TCP_DEFAULT_LISTEN_BACKLOG

```
#define TCP_DEFAULT_LISTEN_BACKLOG   0xff
```

The maximum allowed backlog for TCP listen netconns. This backlog is used unless another is explicitly specified. 0xff is the maximum (u8_t).

2.1.7.5.6.16 TCP_LISTEN_BACKLOG

```
#define TCP_LISTEN_BACKLOG   0
```

TCP_OOSEQ_PBUFS_LIMIT(pcb): Return the maximum number of pbufs to be queued on ooseq per pcb, given the pcb. Only valid for TCP_QUEUE_OOSEQ==1 && TCP_OOSEQ_MAX_PBUFS==1. Use this to override TCP_OOSEQ_MAX_PBUFS to a dynamic value per pcb. TCP_LISTEN_BACKLOG: Enable the backlog option for tcp listen pcb.

2.1.7.5.6.17 TCP_MAXRTX

```
#define TCP_MAXRTX   12
```

TCP_MAXRTX: Maximum number of retransmissions of data segments.

2.1.7.5.6.18 TCP_MSS

```
#define TCP_MSS   536
```

TCP_MSS: TCP Maximum segment size. (default is 536, a conservative default, you might want to increase this.) For the receive side, this MSS is advertised to the remote side when opening a connection. For the transmit size, this MSS sets an upper limit on the MSS advertised by the remote host.

2.1.7.5.6.19 TCP_OOSEQ_MAX_BYTES

```
#define TCP_OOSEQ_MAX_BYTES   0
```

TCP_OOSEQ_MAX_BYTES: The default maximum number of bytes queued on ooseq per pcb if TCP_OOSEQ_BYTES_LIMIT is not defined. Default is 0 (no limit). Only valid for TCP_QUEUE_OOSEQ==1.

2.1.7.5.6.20 TCP_OOSEQ_MAX_PBUFS

```
#define TCP_OOSEQ_MAX_PBUFS   0
```

TCP_OOSEQ_BYTES_LIMIT(pcb): Return the maximum number of bytes to be queued on ooseq per pcb, given the pcb. Only valid for TCP_QUEUE_OOSEQ==1 && TCP_OOSEQ_MAX_BYTES==1. Use this to override TCP_OOSEQ_MAX_BYTES to a dynamic value per pcb. TCP_OOSEQ_MAX_PBUFS: The default maximum number of pbufs queued on ooseq per pcb if TCP_OOSEQ_BYTES_LIMIT is not defined. Default is 0 (no limit). Only valid for TCP_QUEUE_OOSEQ==1.

2.1.7.5.6.21 TCP_OVERSIZE

```
#define TCP_OVERSIZE   TCP_MSS
```

TCP_OVERSIZE: The maximum number of bytes that tcp_write may allocate ahead of time in an attempt to create shorter pbuf chains for transmission. The meaningful range is 0 to TCP_MSS. Some suggested values are:

0: Disable oversized allocation. Each tcp_write() allocates a new pbuf (old behaviour). 1: Allocate size-aligned pbufs with minimal excess. Use this if your scatter-gather DMA requires aligned fragments. 128: Limit the pbuf/memory overhead to 20%. TCP_MSS: Try to create unfragmented TCP packets. TCP_MSS/4: Try to create 4 fragments or less per TCP packet.

2.1.7.5.6.22 TCP_QUEUE_OOSEQ

```
#define TCP_QUEUE_OOSEQ   LWIP_TCP
```

TCP_QUEUE_OOSEQ==1: TCP will queue segments that arrive out of order. Define to 0 if your device is low on memory.

2.1.7.5.6.23 TCP_SND_BUF

```
#define TCP_SND_BUF   (2 * TCP_MSS)
```

TCP_SND_BUF: TCP sender buffer space (bytes). To achieve good performance, this should be at least 2 * TCP_MSS.

2.1.7.5.6.24 TCP_SND_QUEUELEN

```
#define TCP_SND_QUEUELEN   ((4 * (TCP_SND_BUF) + (TCP_MSS - 1))/(TCP_MSS))
```

TCP_SND_QUEUELEN: TCP sender buffer space (pbufs). This must be at least as much as (2 * TCP_SND_BUF/TCP_MSS) for things to work.

2.1.7.5.6.25 TCP_SNDLOWAT

```
#define TCP_SNDLOWAT   LWIP_MIN(LWIP_MAX(((TCP_SND_BUF)/2), (2 * TCP_MSS) + 1), (TCP_SND_B
- 1)
```

TCP_SNDLOWAT: TCP writable space (bytes). This must be less than TCP_SND_BUF. It is the amount of space which must be available in the TCP snd_buf for select to return writable (combined with TCP_SNDQUEUELOWAT).

2.1.7.5.6.26 TCP_SNDQUEUELOWAT

```
#define TCP_SNDQUEUELOWAT   LWIP_MAX(((TCP_SND_QUEUELEN)/2), 5)
```

TCP_SNDQUEUELOWAT: TCP writable bufs (pbuf count). This must be less than TCP_SND_QUEUELEN. If the number of pbufs queued on a pcb drops below this number, select returns writable (combined with TCP_SNDLOWAT).

2.1.7.5.6.27 TCP_SYNMAXRTX

```
#define TCP_SYNMAXRTX   6
```

TCP_SYNMAXRTX: Maximum number of retransmissions of SYN segments.

2.1.7.5.6.28 TCP_TTL

```
#define TCP_TTL   IP_DEFAULT_TTL
```

TCP_TTL: Default Time-To-Live value.

2.1.7.5.6.29 TCP_WND

```
#define TCP_WND   (4 * TCP_MSS)
```

TCP_WND: The size of a TCP window. This must be at least (2 * TCP_MSS) for things to work well. ATTENTION: when using TCP_RCV_SCALE, TCP_WND is the total size with scaling applied. Maximum window value in the TCP header will be TCP_WND >> TCP_RCV_SCALE

2.1.7.5.6.30 TCP_WND_UPDATE_THRESHOLD

```
#define TCP_WND_UPDATE_THRESHOLD   LWIP_MIN((TCP_WND / 4), (TCP_MSS * 4))
```

TCP_WND_UPDATE_THRESHOLD: difference in window to trigger an explicit window update

2.1.7.6 Thread-safe APIs

2.1.7.6.1 Modules

- Netconn
- Sockets

2.1.7.6.2 Detailed Description

2.1.7.6.3 Netconn

2.1.7.6.3.1 Macros

- #define LWIP_NETCONN 1
- #define LWIP_TCPIP_TIMEOUT 0
- #define LWIP_NETCONN_SEM_PER_THREAD 0
- #define LWIP_NETCONN_FULLDUPLEX 0

2.1.7.6.3.2 Detailed Description

2.1.7.6.3.3 Macro Definition Documentation

2.1.7.6.3.4 LWIP_NETCONN

```
#define LWIP_NETCONN   1
```

LWIP_NETCONN==1: Enable Netconn API (require to use api_lib.c)

2.1.7.6.3.5 LWIP_NETCONN_FULLDUPLEX

```
#define LWIP_NETCONN_FULLDUPLEX   0
```

LWIP_NETCONN_FULLDUPLEX==1: Enable code that allows reading from one thread, writing from a 2nd thread and closing from a 3rd thread at the same time. LWIP_NETCONN_SEM_PER_THREAD==1 is required to use one socket/netconn from multiple threads at once!

2.1.7.6.3.6 LWIP_NETCONN_SEM_PER_THREAD

```
#define LWIP_NETCONN_SEM_PER_THREAD   0
```

LWIP_NETCONN_SEM_PER_THREAD==1: Use one (thread-local) semaphore per thread calling socket/netconn functions instead of allocating one semaphore per netconn (and per select etc.) ATTENTION: a thread-local semaphore for API calls is needed:

- LWIP_NETCONN_THREAD_SEM_GET() returning a sys_sem_t*
- LWIP_NETCONN_THREAD_SEM_ALLOC() creating the semaphore
- LWIP_NETCONN_THREAD_SEM_FREE() freeing the semaphore The latter 2 can be invoked up by calling netconn_thread_init()/ne
  Ports may call these for threads created with sys_thread_new().

2.1.7.6.3.7 LWIP_TCPIP_TIMEOUT

```
#define LWIP_TCPIP_TIMEOUT   0
```

LWIP_TCPIP_TIMEOUT==1: Enable tcpip_timeout/tcpip_untimeout to create timers running in tcpip_thread from another thread.

2.1.7.6.4 Sockets

2.1.7.6.4.1 Macros

- #define LWIP_SOCKET 1
- #define LWIP_COMPAT_SOCKETS 1
- #define LWIP_POSIX_SOCKETS_IO_NAMES 1
- #define LWIP_SOCKET_OFFSET 0
- #define LWIP_SOCKET_EXTERNAL_HEADERS 0
- #define LWIP_TCP_KEEPALIVE 0
- #define LWIP_SO_SNDTIMEO 0
- #define LWIP_SO_RCVTIMEO 0
- #define LWIP_SO_SNDRCVTIMEO_NONSTANDARD 0
- #define LWIP_SO_RCVBUF 0
- #define LWIP_SO_LINGER 0
- #define RECV_BUFSIZE_DEFAULT INT_MAX
- #define LWIP_TCP_CLOSE_TIMEOUT_MS_DEFAULT 20000
- #define SO_REUSE 0
- #define SO_REUSE_RXTOALL 0
- #define LWIP_FIONREAD_LINUXMODE 0
- #define LWIP_SOCKET_SELECT 1
- #define LWIP_SOCKET_POLL 1

2.1.7.6.4.2 Detailed Description

2.1.7.6.4.3 Macro Definition Documentation

2.1.7.6.4.4 LWIP_COMPAT_SOCKETS

```
#define LWIP_COMPAT_SOCKETS   1
```

LWIP_COMPAT_SOCKETS==1: Enable BSD-style sockets functions names through defines. LWIP_COMPAT_SOCKETS==2: Same as ==1 but correctly named functions are created. While this helps code completion, it might conflict with existing libraries. (only used if you use sockets.c)

2.1.7.6.4.5 LWIP_FIONREAD_LINUXMODE

```
#define LWIP_FIONREAD_LINUXMODE   0
```

LWIP_FIONREAD_LINUXMODE==0 (default): ioctl/FIONREAD returns the amount of pending data in the network buffer. This is the way windows does it. It's the default for lwIP since it is smaller. LWIP_FIONREAD_LINUXMODE==1: ioctl/-FIONREAD returns the size of the next pending datagram in bytes. This is the way linux does it. This code is only here for compatibility.

2.1.7.6.4.6 LWIP_POSIX_SOCKETS_IO_NAMES

```
#define LWIP_POSIX_SOCKETS_IO_NAMES   1
```

LWIP_POSIX_SOCKETS_IO_NAMES==1: Enable POSIX-style sockets functions names. Disable this option if you use a POSIX operating system that uses the same names (read, write & close). (only used if you use sockets.c)

2.1.7.6.4.7 LWIP_SO_LINGER

```
#define LWIP_SO_LINGER   0
```

LWIP_SO_LINGER==1: Enable SO_LINGER processing.

2.1.7.6.4.8 LWIP_SO_RCVBUF

```
#define LWIP_SO_RCVBUF   0
```

LWIP_SO_RCVBUF==1: Enable SO_RCVBUF processing.

2.1.7.6.4.9 LWIP_SO_RCVTIMEO

```
#define LWIP_SO_RCVTIMEO   0
```

LWIP_SO_RCVTIMEO==1: Enable receive timeout for sockets/netconns and SO_RCVTIMEO processing.

2.1.7.6.4.10 LWIP_SO_SNDRCVTIMEO_NONSTANDARD

```
#define LWIP_SO_SNDRCVTIMEO_NONSTANDARD   0
```

LWIP_SO_SNDRCVTIMEO_NONSTANDARD==1: SO_RCVTIMEO/SO_SNDTIMEO take an int (milliseconds, much like winsock does) instead of a struct timeval (default).

2.1.7.6.4.11 LWIP_SO_SNDTIMEO

```
#define LWIP_SO_SNDTIMEO   0
```

LWIP_SO_SNDTIMEO==1: Enable send timeout for sockets/netconns and SO_SNDTIMEO processing.

2.1.7.6.4.12 LWIP_SOCKET

```
#define LWIP_SOCKET   1
```

LWIP_SOCKET==1: Enable Socket API (require to use sockets.c)

2.1.7.6.4.13 LWIP_SOCKET_EXTERNAL_HEADERS

```
#define LWIP_SOCKET_EXTERNAL_HEADERS   0
```

LWIP_SOCKET_EXTERNAL_HEADERS==1: Use external headers instead of sockets.h and inet.h. In this case, user must provide its own headers by setting the values for LWIP_SOCKET_EXTERNAL_HEADER_SOCKETS_H and LWIP_SOCKET_EXTERN to appropriate include file names and the whole content of the default sockets.h and inet.h is skipped.

2.1.7.6.4.14 LWIP_SOCKET_OFFSET

```
#define LWIP_SOCKET_OFFSET   0
```

LWIP_SOCKET_OFFSET==n: Increases the file descriptor number created by LwIP with n. This can be useful when there are multiple APIs which create file descriptors. When they all start with a different offset and you won't make them overlap you can re implement read/write/close/ioctl/fnctl to send the requested action to the right library (sharing select will need more work though).

2.1.7.6.4.15 LWIP_SOCKET_POLL

```
#define LWIP_SOCKET_POLL   1
```

LWIP_SOCKET_POLL==1 (default): enable poll() for sockets (including struct pollfd, nfds_t, and constants)

2.1.7.6.4.16 LWIP_SOCKET_SELECT

```
#define LWIP_SOCKET_SELECT   1
```

LWIP_SOCKET_SELECT==1 (default): enable select() for sockets (uses a netconn callback to keep track of events). This saves RAM (counters per socket) and code (netconn event callback), which should improve performance a bit).

2.1.7.6.4.17 LWIP_TCP_CLOSE_TIMEOUT_MS_DEFAULT

```
#define LWIP_TCP_CLOSE_TIMEOUT_MS_DEFAULT   20000
```

By default, TCP socket/netconn close waits 20 seconds max to send the FIN

2.1.7.6.4.18 LWIP_TCP_KEEPALIVE

```
#define LWIP_TCP_KEEPALIVE   0
```

LWIP_TCP_KEEPALIVE==1: Enable TCP_KEEPIDLE, TCP_KEEPINTVL and TCP_KEEPCNT options processing. Note that TCP_KEEPIDLE and TCP_KEEPINTVL have to be set in seconds. (does not require sockets.c, and will affect tcp.c)

2.1.7.6.4.19 RECV_BUFSIZE_DEFAULT

```
#define RECV_BUFSIZE_DEFAULT   INT_MAX
```

If LWIP_SO_RCVBUF is used, this is the default value for recv_bufsize.

2.1.7.6.4.20 SO_REUSE

```
#define SO_REUSE   0
```

SO_REUSE==1: Enable SO_REUSEADDR option.

2.1.7.6.4.21 SO_REUSE_RXTOALL

```
#define SO_REUSE_RXTOALL   0
```

SO_REUSE_RXTOALL==1: Pass a copy of incoming broadcast/multicast packets to all local matches if SO_REUSEADDR is turned on. WARNING: Adds a memcpy for every packet if passing to more than one pcb!

2.1.7.7 IPv4

2.1.7.7.1 Modules

- ARP
- ICMP
- DHCP
- AUTOIP
- ACD
- IGMP

2.1.7.7.2 Macros

- #define LWIP_IPV4 1
- #define IP_FORWARD 0
- #define IP_REASSEMBLY 1
- #define IP_FRAG 1
- #define IP_OPTIONS_ALLOWED 1
- #define IP_REASS_MAXAGE 15
- #define IP_REASS_MAX_PBUFS 10
- #define IP_DEFAULT_TTL 255
- #define IP_SOF_BROADCAST 0
- #define IP_SOF_BROADCAST_RECV 0
- #define IP_FORWARD_ALLOW_TX_ON_RX_NETIF 0

2.1.7.7.3 Detailed Description

2.1.7.7.4 Macro Definition Documentation

2.1.7.7.4.1 IP_DEFAULT_TTL

```
#define IP_DEFAULT_TTL   255
```

IP_DEFAULT_TTL: Default value for Time-To-Live used by transport layers.

2.1.7.7.4.2 IP_FORWARD

```
#define IP_FORWARD   0
```

IP_FORWARD==1: Enables the ability to forward IP packets across network interfaces. If you are going to run lwIP on a device with only one network interface, define this to 0.

2.1.7.7.4.3 IP_FORWARD_ALLOW_TX_ON_RX_NETIF

```
#define IP_FORWARD_ALLOW_TX_ON_RX_NETIF   0
```

IP_FORWARD_ALLOW_TX_ON_RX_NETIF==1: allow ip_forward() to send packets back out on the netif where it was received. This should only be used for wireless networks. ATTENTION: When this is 1, make sure your netif driver correctly marks incoming link-layer-broadcast/multicast packets as such using the corresponding pbuf flags!

2.1.7.7.4.4 IP_FRAG

```
#define IP_FRAG   1
```

IP_FRAG==1: Fragment outgoing IP packets if their size exceeds MTU. Note that this option does not affect incoming packet sizes, which can be controlled via IP_REASSEMBLY.

2.1.7.7.4.5 IP_OPTIONS_ALLOWED

```
#define IP_OPTIONS_ALLOWED   1
```

IP_OPTIONS_ALLOWED: Defines the behavior for IP options. IP_OPTIONS_ALLOWED==0: All packets with IP options are dropped. IP_OPTIONS_ALLOWED==1: IP options are allowed (but not parsed).

2.1.7.7.4.6 IP_REASS_MAX_PBUFS

```
#define IP_REASS_MAX_PBUFS   10
```

IP_REASS_MAX_PBUFS: Total maximum amount of pbufs waiting to be reassembled. Since the received pbufs are enqueued, be sure to configure PBUF_POOL_SIZE > IP_REASS_MAX_PBUFS so that the stack is still able to receive packets even if the maximum amount of fragments is enqueued for reassembly! When IPv4 *and* IPv6 are enabled, this even changes to (PBUF_POOL_SIZE > 2 * IP_REASS_MAX_PBUFS)!

2.1.7.7.4.7 IP_REASS_MAXAGE

```
#define IP_REASS_MAXAGE   15
```

IP_REASS_MAXAGE: Maximum time (in multiples of IP_TMR_INTERVAL - so seconds, normally) a fragmented IP packet waits for all fragments to arrive. If not all fragments arrived in this time, the whole packet is discarded.

2.1.7.7.4.8 IP_REASSEMBLY

```
#define IP_REASSEMBLY   1
```

IP_REASSEMBLY==1: Reassemble incoming fragmented IP packets. Note that this option does not affect outgoing packet sizes, which can be controlled via IP_FRAG.

2.1.7.7.4.9 IP_SOF_BROADCAST

```
#define IP_SOF_BROADCAST   0
```

IP_SOF_BROADCAST=1: Use the SOF_BROADCAST field to enable broadcast filter per pcb on udp and raw send operations. To enable broadcast filter on recv operations, you also have to set IP_SOF_BROADCAST_RECV=1.

2.1.7.7.4.10 IP_SOF_BROADCAST_RECV

```
#define IP_SOF_BROADCAST_RECV   0
```

IP_SOF_BROADCAST_RECV (requires IP_SOF_BROADCAST=1) enable the broadcast filter on recv operations.

2.1.7.7.4.11 LWIP_IPV4

```
#define LWIP_IPV4   1
```

LWIP_IPV4==1: Enable IPv4

2.1.7.7.5 ARP

2.1.7.7.5.1 Macros

- #define LWIP_ARP 1
- #define ARP_TABLE_SIZE 10
- #define ARP_MAXAGE 300
- #define ARP_QUEUEING 0
- #define ARP_QUEUE_LEN 3
- #define ETHARP_SUPPORT_VLAN 0
- #define LWIP_ETHERNET LWIP_ARP
- #define ETH_PAD_SIZE 0
- #define ETHARP_SUPPORT_STATIC_ENTRIES 0
- #define ETHARP_TABLE_MATCH_NETIF !LWIP_SINGLE_NETIF

2.1.7.7.5.2 Detailed Description

2.1.7.7.5.3 Macro Definition Documentation

2.1.7.7.5.4 ARP_MAXAGE

```
#define ARP_MAXAGE   300
```

the time an ARP entry stays valid after its last update, for ARP_TMR_INTERVAL = 1000, this is (60 * 5) seconds = 5 minutes.

2.1.7.7.5.5 ARP_QUEUE_LEN

```
#define ARP_QUEUE_LEN   3
```

The maximum number of packets which may be queued for each unresolved address by other network layers. Defaults to 3, 0 means disabled. Old packets are dropped, new packets are queued.

2.1.7.7.5.6 ARP_QUEUEING

```
#define ARP_QUEUEING   0
```

ARP_QUEUEING==1: Multiple outgoing packets are queued during hardware address resolution. By default, only the most recent packet is queued per IP address. This is sufficient for most protocols and mainly reduces TCP connection startup time. Set this to 1 if you know your application sends more than one packet in a row to an IP address that is not in the ARP cache.

2.1.7.7.5.7 ARP_TABLE_SIZE

```
#define ARP_TABLE_SIZE   10
```

ARP_TABLE_SIZE: Number of active MAC-IP address pairs cached.

2.1.7.7.5.8 ETH_PAD_SIZE

```
#define ETH_PAD_SIZE   0
```

ETH_PAD_SIZE: number of bytes added before the ethernet header to ensure alignment of payload after that header. Since the header is 14 bytes long, without this padding e.g. addresses in the IP header will not be aligned on a 32-bit boundary, so setting this to 2 can speed up 32-bit-platforms.

2.1.7.7.5.9 ETHARP_SUPPORT_STATIC_ENTRIES

```
#define ETHARP_SUPPORT_STATIC_ENTRIES   0
```

ETHARP_SUPPORT_STATIC_ENTRIES==1: enable code to support static ARP table entries (using etharp_add_static_entry/etharp_re

2.1.7.7.5.10 ETHARP_SUPPORT_VLAN

```
#define ETHARP_SUPPORT_VLAN   0
```

ETHARP_SUPPORT_VLAN==1: support receiving and sending ethernet packets with VLAN header. See the description of LWIP_HOOK_VLAN_CHECK and LWIP_HOOK_VLAN_SET hooks to check/set VLAN headers. Additionally, you can define ETHARP_VLAN_CHECK to an u16_t VLAN ID to check. If ETHARP_VLAN_CHECK is defined, only VLAN-traffic for this VLAN is accepted. If ETHARP_VLAN_CHECK is not defined, all traffic is accepted. Alternatively, define a function/define ETHARP_VLAN_CHECK_FN(eth_hdr, vlan) that returns 1 to accept a packet or 0 to drop a packet.

2.1.7.7.5.11 ETHARP_TABLE_MATCH_NETIF

```
#define ETHARP_TABLE_MATCH_NETIF   !LWIP_SINGLE_NETIF
```

ETHARP_TABLE_MATCH_NETIF==1: Match netif for ARP table entries. If disabled, duplicate IP address on multiple netifs are not supported (but this should only occur for AutoIP).

2.1.7.7.5.12 LWIP_ARP

```
#define LWIP_ARP   1
```

LWIP_ARP==1: Enable ARP functionality.

2.1.7.7.5.13 LWIP_ETHERNET

```
#define LWIP_ETHERNET   LWIP_ARP
```

LWIP_VLAN_PCP==1: Enable outgoing VLAN tagging of frames on a per-PCB basis for QoS purposes. With this feature enabled, each PCB has a new variable: "netif_hints.tci" (Tag Control Identifier). The TCI contains three fields: VID, CFI and PCP.

- VID is the VLAN ID, which should be set to zero.
- The "CFI" bit is used to enable or disable VLAN tags for the PCB.
- PCP (Priority Code Point) is a 3 bit field used for Ethernet level QoS. See pcb_tci_*() functions to get/set/clear this. LWIP_ETHERNET enable ethernet support even though ARP might be disabled

2.1.7.7.6 ICMP

2.1.7.7.6.1 Macros

- #define LWIP_ICMP 1
- #define ICMP_TTL IP_DEFAULT_TTL
- #define LWIP_BROADCAST_PING 0
- #define LWIP_MULTICAST_PING 0

2.1.7.7.6.2 Detailed Description

2.1.7.7.6.3 Macro Definition Documentation

2.1.7.7.6.4 ICMP_TTL

```
#define ICMP_TTL   IP_DEFAULT_TTL
```

ICMP_TTL: Default value for Time-To-Live used by ICMP packets.

2.1.7.7.6.5 LWIP_BROADCAST_PING

```
#define LWIP_BROADCAST_PING   0
```

LWIP_BROADCAST_PING==1: respond to broadcast pings (default is unicast only)

2.1.7.7.6.6 LWIP_ICMP

```
#define LWIP_ICMP   1
```

LWIP_ICMP==1: Enable ICMP module inside the IP stack. Be careful, disable that make your product non-compliant to RFC1122

2.1.7.7.6.7 LWIP_MULTICAST_PING

```
#define LWIP_MULTICAST_PING   0
```

LWIP_MULTICAST_PING==1: respond to multicast pings (default is unicast only)

2.1.7.7.7 DHCP

2.1.7.7.7.1 Macros

- #define LWIP_DHCP 0
- #define LWIP_DHCP_DOES_ACD_CHECK LWIP_DHCP
- #define LWIP_DHCP_BOOTP_FILE 0
- #define LWIP_DHCP_GET_NTP_SRV 0
- #define LWIP_DHCP_MAX_NTP_SERVERS 1
- #define LWIP_DHCP_MAX_DNS_SERVERS DNS_MAX_SERVERS
- #define LWIP_DHCP_DISCOVER_ADD_HOSTNAME 1

2.1.7.7.7.2 Detailed Description

2.1.7.7.7.3 Macro Definition Documentation

2.1.7.7.7.4 LWIP_DHCP

```
#define LWIP_DHCP   0
```

LWIP_DHCP==1: Enable DHCP module.

2.1.7.7.7.5 LWIP_DHCP_BOOTP_FILE

```
#define LWIP_DHCP_BOOTP_FILE   0
```

LWIP_DHCP_BOOTP_FILE==1: Store offered_si_addr and boot_file_name.

2.1.7.7.7.6 LWIP_DHCP_DISCOVER_ADD_HOSTNAME

```
#define LWIP_DHCP_DISCOVER_ADD_HOSTNAME   1
```

LWIP_DHCP_DISCOVER_ADD_HOSTNAME: Set to 0 to not include hostname opt in discover packets. If the hostname is not set in the DISCOVER packet, then some servers might issue an OFFER with hostname configured and consequently reject the REQUEST with any other hostname.

2.1.7.7.7.7 LWIP_DHCP_DOES_ACD_CHECK

```
#define LWIP_DHCP_DOES_ACD_CHECK   LWIP_DHCP
```

LWIP_DHCP_DOES_ACD_CHECK==1: Perform address conflict detection on the dhcp address.

2.1.7.7.7.8 LWIP_DHCP_GET_NTP_SRV

```
#define LWIP_DHCP_GET_NTP_SRV   0
```

LWIP_DHCP_GETS_NTP==1: Request NTP servers with discover/select. For each response packet, an callback is called, which has to be provided by the port: void dhcp_set_ntp_servers(u8_t num_ntp_servers, ip_addr_t* ntp_server_addrs);

2.1.7.7.7.9 LWIP_DHCP_MAX_DNS_SERVERS

```
#define LWIP_DHCP_MAX_DNS_SERVERS   DNS_MAX_SERVERS
```

LWIP_DHCP_MAX_DNS_SERVERS > 0: Request DNS servers with discover/select. DNS servers received in the response are passed to DNS via dns_setserver() (up to the maximum limit defined here).

2.1.7.7.7.10 LWIP_DHCP_MAX_NTP_SERVERS

```
#define LWIP_DHCP_MAX_NTP_SERVERS   1
```

The maximum of NTP servers requested

2.1.7.7.8 AUTOIP

2.1.7.7.8.1 Macros

- #define LWIP_AUTOIP 0
- #define LWIP_DHCP_AUTOIP_COOP 0
- #define LWIP_DHCP_AUTOIP_COOP_TRIES 9

2.1.7.7.8.2 Detailed Description

2.1.7.7.8.3 Macro Definition Documentation

2.1.7.7.8.4 LWIP_AUTOIP

```
#define LWIP_AUTOIP   0
```

LWIP_AUTOIP==1: Enable AUTOIP module.

2.1.7.7.8.5 LWIP_DHCP_AUTOIP_COOP

```
#define LWIP_DHCP_AUTOIP_COOP   0
```

LWIP_DHCP_AUTOIP_COOP==1: Allow DHCP and AUTOIP to be both enabled on the same interface at the same time.

2.1.7.7.8.6 LWIP_DHCP_AUTOIP_COOP_TRIES

```
#define LWIP_DHCP_AUTOIP_COOP_TRIES   9
```

LWIP_DHCP_AUTOIP_COOP_TRIES: Set to the number of DHCP DISCOVER probes that should be sent before falling back on AUTOIP (the DHCP client keeps running in this case). This can be set as low as 1 to get an AutoIP address very quickly, but you should be prepared to handle a changing IP address when DHCP overrides AutoIP.

2.1.7.7.9 ACD

2.1.7.7.9.1 Macros

- #define LWIP_ACD (LWIP_AUTOIP || LWIP_DHCP_DOES_ACD_CHECK)

2.1.7.7.9.2 Detailed Description

2.1.7.7.9.3 Macro Definition Documentation

2.1.7.7.9.4 LWIP_ACD

```
#define LWIP_ACD   (LWIP_AUTOIP || LWIP_DHCP_DOES_ACD_CHECK)
```

LWIP_ACD==1: Enable ACD module. ACD module is needed when using AUTOIP.

2.1.7.7.10 IGMP

2.1.7.7.10.1 Macros

- #define LWIP_IGMP 0

2.1.7.7.10.2 Detailed Description

2.1.7.7.10.3 Macro Definition Documentation

2.1.7.7.10.4 LWIP_IGMP

```
#define LWIP_IGMP   0
```

LWIP_IGMP==1: Turn on IGMP module.

2.1.7.8 PBUF

2.1.7.8.1 Macros

- #define PBUF_LINK_HLEN (14 + ETH_PAD_SIZE)
- #define PBUF_LINK_ENCAPSULATION_HLEN 0
- #define PBUF_POOL_BUFSIZE LWIP_MEM_ALIGN_SIZE(TCP_MSS+PBUF_IP_HLEN+PBUF_TRANSPORT_HLEN+PBUF_
- #define LWIP_PBUF_REF_T u8_t
- #define LWIP_PBUF_CUSTOM_DATA
- #define LWIP_PBUF_CUSTOM_DATA_INIT(p)

2.1.7.8.2 Detailed Description

2.1.7.8.3 Macro Definition Documentation

2.1.7.8.3.1 LWIP_PBUF_CUSTOM_DATA

```
#define LWIP_PBUF_CUSTOM_DATA
```

LWIP_PBUF_CUSTOM_DATA: Store private data on pbufs (e.g. timestamps) This extends struct pbuf so user can store custom data on every pbuf. e.g.: #define LWIP_PBUF_CUSTOM_DATA u32_t myref;

2.1.7.8.3.2 LWIP_PBUF_CUSTOM_DATA_INIT

```
#define LWIP_PBUF_CUSTOM_DATA_INIT( p)
```

LWIP_PBUF_CUSTOM_DATA_INIT: Initialize private data on pbufs. e.g. for the above example definition: #define LWIP_PBUF_CUS
(p)->myref = 0

2.1.7.8.3.3 LWIP_PBUF_REF_T

```
#define LWIP_PBUF_REF_T   u8_t
```

LWIP_PBUF_REF_T: Refcount type in pbuf. Default width of u8_t can be increased if 255 refs are not enough for you.

2.1.7.8.3.4 PBUF_LINK_ENCAPSULATION_HLEN

```
#define PBUF_LINK_ENCAPSULATION_HLEN   0
```

PBUF_LINK_ENCAPSULATION_HLEN: the number of bytes that should be allocated for an additional encapsulation header before ethernet headers (e.g. 802.11)

2.1.7.8.3.5 PBUF_LINK_HLEN

```
#define PBUF_LINK_HLEN   (14 + ETH_PAD_SIZE)
```

PBUF_LINK_HLEN: the number of bytes that should be allocated for a link level header. The default is 14, the standard value for Ethernet.

2.1.7.8.3.6 PBUF_POOL_BUFSIZE

```
#define PBUF_POOL_BUFSIZE   LWIP_MEM_ALIGN_SIZE(TCP_MSS+PBUF_IP_HLEN+PBUF_TRANSPORT_HLEN+P
```

PBUF_POOL_BUFSIZE: the size of each pbuf in the pbuf pool. The default is designed to accommodate single full size TCP frame in one pbuf, including TCP_MSS, IP header, and link header.

2.1.7.9 NETIF

2.1.7.9.1 Modules

- Loopback interface

2.1.7.9.2 Macros

- #define LWIP_SINGLE_NETIF 0
- #define LWIP_NETIF_HOSTNAME 0
- #define LWIP_NETIF_API 0
- #define LWIP_NETIF_STATUS_CALLBACK 0
- #define LWIP_NETIF_EXT_STATUS_CALLBACK 0
- #define LWIP_NETIF_LINK_CALLBACK 0
- #define LWIP_NETIF_REMOVE_CALLBACK 0
- #define LWIP_NETIF_HWADDRHINT 0
- #define LWIP_NETIF_TX_SINGLE_PBUF 0
- #define LWIP_NUM_NETIF_CLIENT_DATA 0

2.1.7.9.3 Detailed Description

2.1.7.9.4 Macro Definition Documentation

2.1.7.9.4.1 LWIP_NETIF_API

```
#define LWIP_NETIF_API   0
```

LWIP_NETIF_API==1: Support netif api (in netifapi.c)

2.1.7.9.4.2 LWIP_NETIF_EXT_STATUS_CALLBACK

```
#define LWIP_NETIF_EXT_STATUS_CALLBACK   0
```

LWIP_NETIF_EXT_STATUS_CALLBACK==1: Support an extended callback function for several netif related event that supports multiple subscribers. **See also** netif_ext_status_callback

2.1.7.9.4.3 LWIP_NETIF_HOSTNAME

```
#define LWIP_NETIF_HOSTNAME   0
```

LWIP_NETIF_HOSTNAME==1: use DHCP_OPTION_HOSTNAME with netif's hostname field.

2.1.7.9.4.4 LWIP_NETIF_HWADDRHINT

```
#define LWIP_NETIF_HWADDRHINT   0
```

LWIP_NETIF_HWADDRHINT==1: Cache link-layer-address hints (e.g. table indices) in struct netif. TCP and UDP can make use of this to prevent scanning the ARP table for every sent packet. While this is faster for big ARP tables or many concurrent connections, it might be counterproductive if you have a tiny ARP table or if there never are concurrent connections.

2.1.7.9.4.5 LWIP_NETIF_LINK_CALLBACK

```
#define LWIP_NETIF_LINK_CALLBACK   0
```

LWIP_NETIF_LINK_CALLBACK==1: Support a callback function from an interface whenever the link changes (i.e., link down)

2.1.7.9.4.6 LWIP_NETIF_REMOVE_CALLBACK

```
#define LWIP_NETIF_REMOVE_CALLBACK   0
```

LWIP_NETIF_REMOVE_CALLBACK==1: Support a callback function that is called when a netif has been removed

2.1.7.9.4.7 LWIP_NETIF_STATUS_CALLBACK

```
#define LWIP_NETIF_STATUS_CALLBACK   0
```

LWIP_NETIF_STATUS_CALLBACK==1: Support a callback function whenever an interface changes its up/down status (i.e., due to DHCP IP acquisition)

2.1.7.9.4.8 LWIP_NETIF_TX_SINGLE_PBUF

```
#define LWIP_NETIF_TX_SINGLE_PBUF   0
```

LWIP_NETIF_TX_SINGLE_PBUF: if this is set to 1, lwIP *tries* to put all data to be sent into one single pbuf. This is for compatibility with DMA-enabled MACs that do not support scatter-gather. Beware that this might involve CPU-memcpy before transmitting that would not be needed without this flag! Use this only if you need to!

ATTENTION: a driver should *NOT* rely on getting single pbufs but check TX pbufs for being in one piece. If not, pbuf_clone can be used to get a single pbuf: if (p->next != NULL) { struct pbuf *q = pbuf_clone(PBUF_RAW, PBUF_RAM, p); if (q == NULL) { return ERR_MEM; } p = q; ATTENTION: do NOT free the old 'p' as the ref belongs to the caller! }

2.1.7.9.4.9 LWIP_NUM_NETIF_CLIENT_DATA

```
#define LWIP_NUM_NETIF_CLIENT_DATA   0
```

LWIP_NUM_NETIF_CLIENT_DATA: Number of clients that may store data in client_data member array of struct netif (max. 256).

2.1.7.9.4.10 LWIP_SINGLE_NETIF

```
#define LWIP_SINGLE_NETIF   0
```

LWIP_SINGLE_NETIF==1: use a single netif only. This is the common case for small real-life targets. Some code like routing etc. can be left out.

2.1.7.9.5 Loopback interface

2.1.7.9.5.1 Macros

- #define LWIP_HAVE_LOOPIF (LWIP_NETIF_LOOPBACK && !LWIP_SINGLE_NETIF)
- #define LWIP_LOOPIF_MULTICAST 0
- #define LWIP_NETIF_LOOPBACK 0
- #define LWIP_LOOPBACK_MAX_PBUFS 0
- #define LWIP_NETIF_LOOPBACK_MULTITHREADING (!NO_SYS)

2.1.7.9.5.2 Detailed Description

2.1.7.9.5.3 Macro Definition Documentation

2.1.7.9.5.4 LWIP_HAVE_LOOPIF

```
#define LWIP_HAVE_LOOPIF   (LWIP_NETIF_LOOPBACK && !LWIP_SINGLE_NETIF)
```

LWIP_HAVE_LOOPIF==1: Support loop interface (127.0.0.1). This is only needed when no real netifs are available. If at least one other netif is available, loopback traffic uses this netif.

2.1.7.9.5.5 LWIP_LOOPBACK_MAX_PBUFS

```
#define LWIP_LOOPBACK_MAX_PBUFS   0
```

LWIP_LOOPBACK_MAX_PBUFS: Maximum number of pbufs on queue for loopback sending for each netif (0 = disabled)

2.1.7.9.5.6 LWIP_LOOPIF_MULTICAST

```
#define LWIP_LOOPIF_MULTICAST   0
```

LWIP_LOOPIF_MULTICAST==1: Support multicast/IGMP on loop interface (127.0.0.1).

2.1.7.9.5.7 LWIP_NETIF_LOOPBACK

```
#define LWIP_NETIF_LOOPBACK   0
```

LWIP_NETIF_LOOPBACK==1: Support sending packets with a destination IP address equal to the netif IP address, looping them back up the stack.

2.1.7.9.5.8 LWIP_NETIF_LOOPBACK_MULTITHREADING

```
#define LWIP_NETIF_LOOPBACK_MULTITHREADING   (!NO_SYS)
```

LWIP_NETIF_LOOPBACK_MULTITHREADING: Indicates whether threading is enabled in the system, as netifs must change how they behave depending on this setting for the LWIP_NETIF_LOOPBACK option to work. Setting this is needed to avoid reentering non-reentrant functions like tcp_input(). LWIP_NETIF_LOOPBACK_MULTITHREADING==1: Indicates that the user is using a multithreaded environment like tcpip.c. In this case, netif->input() is called directly. LWIP_NETIF_LOOPBACK_MULTI
Indicates a polling (or NO_SYS) setup. The packets are put on a list and netif_poll() must be called in the main application loop.

2.1.7.10 IPv6

2.1.7.10.1 Modules

- ICMP6
- Multicast listener discovery
- Neighbor discovery
- DHCPv6

2.1.7.10.2 Macros

- #define LWIP_IPV6 0
- #define IPV6_REASS_MAXAGE 60
- #define LWIP_IPV6_SCOPES (LWIP_IPV6 && !LWIP_SINGLE_NETIF)
- #define LWIP_IPV6_SCOPES_DEBUG 0
- #define LWIP_IPV6_NUM_ADDRESSES 3
- #define LWIP_IPV6_FORWARD 0
- #define LWIP_IPV6_FRAG 1
- #define LWIP_IPV6_REASS LWIP_IPV6
- #define LWIP_IPV6_SEND_ROUTER_SOLICIT LWIP_IPV6
- #define LWIP_IPV6_AUTOCONFIG LWIP_IPV6
- #define LWIP_IPV6_ADDRESS_LIFETIMES LWIP_IPV6_AUTOCONFIG
- #define LWIP_IPV6_DUP_DETECT_ATTEMPTS 1

2.1.7.10.3 Detailed Description

2.1.7.10.4 Macro Definition Documentation

2.1.7.10.4.1 IPV6_REASS_MAXAGE

```
#define IPV6_REASS_MAXAGE   60
```

IPV6_REASS_MAXAGE: Maximum time (in multiples of IP6_REASS_TMR_INTERVAL - so seconds, normally) a fragmented IP packet waits for all fragments to arrive. If not all fragments arrived in this time, the whole packet is discarded.

2.1.7.10.4.2 LWIP_IPV6

```
#define LWIP_IPV6   0
```

LWIP_IPV6==1: Enable IPv6

2.1.7.10.4.3 LWIP_IPV6_ADDRESS_LIFETIMES

```
#define LWIP_IPV6_ADDRESS_LIFETIMES   LWIP_IPV6_AUTOCONFIG
```

LWIP_IPV6_ADDRESS_LIFETIMES==1: Keep valid and preferred lifetimes for each IPv6 address. Required for LWIP_IPV6_AUTOC
May still be enabled otherwise, in which case the application may assign address lifetimes with the appropriate macros. Addresses with no lifetime are assumed to be static. If this option is disabled, all addresses are assumed to be static.

2.1.7.10.4.4 LWIP_IPV6_AUTOCONFIG

```
#define LWIP_IPV6_AUTOCONFIG   LWIP_IPV6
```

LWIP_IPV6_AUTOCONFIG==1: Enable stateless address autoconfiguration as per RFC 4862.

2.1.7.10.4.5 LWIP_IPV6_DUP_DETECT_ATTEMPTS

```
#define LWIP_IPV6_DUP_DETECT_ATTEMPTS   1
```

LWIP_IPV6_DUP_DETECT_ATTEMPTS=[0..7]: Number of duplicate address detection attempts.

2.1.7.10.4.6 LWIP_IPV6_FORWARD

```
#define LWIP_IPV6_FORWARD   0
```

LWIP_IPV6_FORWARD==1: Forward IPv6 packets across netifs

2.1.7.10.4.7 LWIP_IPV6_FRAG

```
#define LWIP_IPV6_FRAG   1
```

LWIP_IPV6_FRAG==1: Fragment outgoing IPv6 packets that are too big.

2.1.7.10.4.8 LWIP_IPV6_NUM_ADDRESSES

```
#define LWIP_IPV6_NUM_ADDRESSES   3
```

LWIP_IPV6_NUM_ADDRESSES: Number of IPv6 addresses per netif.

2.1.7.10.4.9 LWIP_IPV6_REASS

```
#define LWIP_IPV6_REASS   LWIP_IPV6
```

LWIP_IPV6_REASS==1: reassemble incoming IPv6 packets that fragmented

2.1.7.10.4.10 LWIP_IPV6_SCOPES

```
#define LWIP_IPV6_SCOPES   (LWIP_IPV6 && !LWIP_SINGLE_NETIF)
```

LWIP_IPV6_SCOPES==1: Enable support for IPv6 address scopes, ensuring that e.g. link-local addresses are really treated as link-local. Disable this setting only for single-interface configurations. All addresses that have a scope according to the default policy (link-local unicast addresses, interface-local and link-local multicast addresses) should now have a zone set on them before being passed to the core API, although lwIP will currently attempt to select a zone on the caller's behalf when necessary. Applications that directly assign IPv6 addresses to interfaces (which is NOT recommended) must now ensure that link-local addresses carry the netif's zone. See the new ip6_zone.h header file for more information and relevant macros. For now it is still possible to turn off scopes support through the new LWIP_IPV6_SCOPES option. When upgrading an implementation that uses the core API directly, it is highly recommended to enable LWIP_IPV6_SCOPES_DEBUG at least for a while, to ensure e.g. proper address initialization.

2.1.7.10.4.11 LWIP_IPV6_SCOPES_DEBUG

```
#define LWIP_IPV6_SCOPES_DEBUG   0
```

LWIP_IPV6_SCOPES_DEBUG==1: Perform run-time checks to verify that addresses are properly zoned (see ip6_zone.h on what that means) where it matters. Enabling this setting is highly recommended when upgrading from an existing installation that is not yet scope-aware; otherwise it may be too expensive.

2.1.7.10.4.12 LWIP_IPV6_SEND_ROUTER_SOLICIT

```
#define LWIP_IPV6_SEND_ROUTER_SOLICIT   LWIP_IPV6
```

LWIP_IPV6_SEND_ROUTER_SOLICIT==1: Send router solicitation messages during network startup.

2.1.7.10.5 ICMP6

2.1.7.10.5.1 Macros

- #define LWIP_ICMP6 LWIP_IPV6
- #define LWIP_ICMP6_DATASIZE 0
- #define LWIP_ICMP6_HL 255

2.1.7.10.5.2 Detailed Description

2.1.7.10.5.3 Macro Definition Documentation

2.1.7.10.5.4 LWIP_ICMP6

```
#define LWIP_ICMP6   LWIP_IPV6
```

LWIP_ICMP6==1: Enable ICMPv6 (mandatory per RFC)

2.1.7.10.5.5 LWIP_ICMP6_DATASIZE

```
#define LWIP_ICMP6_DATASIZE   0
```

LWIP_ICMP6_DATASIZE: bytes from original packet to send back in ICMPv6 error messages (0 = default of IP6_MIN_MTU_LENGT
ATTENTION: RFC4443 section 2.4 says IP6_MIN_MTU_LENGTH is a MUST, so override this only if you absolutely have to!

2.1.7.10.5.6 LWIP_ICMP6_HL

```
#define LWIP_ICMP6_HL   255
```

LWIP_ICMP6_HL: default hop limit for ICMPv6 messages

2.1.7.10.6 Multicast listener discovery

2.1.7.10.6.1 Macros

- #define LWIP_IPV6_MLD LWIP_IPV6
- #define MEMP_NUM_MLD6_GROUP 4

2.1.7.10.6.2 Detailed Description

2.1.7.10.6.3 Macro Definition Documentation

2.1.7.10.6.4 LWIP_IPV6_MLD

```
#define LWIP_IPV6_MLD   LWIP_IPV6
```

LWIP_IPV6_MLD==1: Enable multicast listener discovery protocol. If LWIP_IPV6 is enabled but this setting is disabled, the MAC layer must indiscriminately pass all inbound IPv6 multicast traffic to lwIP.

2.1.7.10.6.5 MEMP_NUM_MLD6_GROUP

```
#define MEMP_NUM_MLD6_GROUP   4
```

MEMP_NUM_MLD6_GROUP: Max number of IPv6 multicast groups that can be joined. There must be enough groups so that each netif can join the solicited-node multicast group for each of its local addresses, plus one for MDNS if applicable, plus any number of groups to be joined on UDP sockets.

2.1.7.10.7 Neighbor discovery

2.1.7.10.7.1 Macros

- #define LWIP_ND6_QUEUEING LWIP_IPV6
- #define MEMP_NUM_ND6_QUEUE 20
- #define LWIP_ND6_NUM_NEIGHBORS 10
- #define LWIP_ND6_NUM_DESTINATIONS 10
- #define LWIP_ND6_NUM_PREFIXES 5
- #define LWIP_ND6_NUM_ROUTERS 3
- #define LWIP_ND6_MAX_MULTICAST_SOLICIT 3
- #define LWIP_ND6_MAX_UNICAST_SOLICIT 3
- #define LWIP_ND6_MAX_ANYCAST_DELAY_TIME 1000
- #define LWIP_ND6_MAX_NEIGHBOR_ADVERTISEMENT 3
- #define LWIP_ND6_REACHABLE_TIME 30000
- #define LWIP_ND6_RETRANS_TIMER 1000
- #define LWIP_ND6_DELAY_FIRST_PROBE_TIME 5000
- #define LWIP_ND6_ALLOW_RA_UPDATES 1
- #define LWIP_ND6_TCP_REACHABILITY_HINTS 1
- #define LWIP_ND6_RDNSS_MAX_DNS_SERVERS 0

2.1.7.10.7.2 Detailed Description

2.1.7.10.7.3 Macro Definition Documentation

2.1.7.10.7.4 LWIP_ND6_ALLOW_RA_UPDATES

```
#define LWIP_ND6_ALLOW_RA_UPDATES   1
```

LWIP_ND6_ALLOW_RA_UPDATES==1: Allow Router Advertisement messages to update Reachable time and retransmission timers, and netif MTU.

2.1.7.10.7.5 LWIP_ND6_DELAY_FIRST_PROBE_TIME

```
#define LWIP_ND6_DELAY_FIRST_PROBE_TIME   5000
```

LWIP_ND6_DELAY_FIRST_PROBE_TIME: Delay before first unicast neighbor solicitation message is sent, during neighbor reachability detection.

2.1.7.10.7.6 LWIP_ND6_MAX_ANYCAST_DELAY_TIME

```
#define LWIP_ND6_MAX_ANYCAST_DELAY_TIME   1000
```

Unused: See ND RFC (time in milliseconds).

2.1.7.10.7.7 LWIP_ND6_MAX_MULTICAST_SOLICIT

```
#define LWIP_ND6_MAX_MULTICAST_SOLICIT   3
```

LWIP_ND6_MAX_MULTICAST_SOLICIT: max number of multicast solicit messages to send (neighbor solicit and router solicit)

2.1.7.10.7.8 LWIP_ND6_MAX_NEIGHBOR_ADVERTISEMENT

```
#define LWIP_ND6_MAX_NEIGHBOR_ADVERTISEMENT   3
```

Unused: See ND RFC

2.1.7.10.7.9 LWIP_ND6_MAX_UNICAST_SOLICIT

```
#define LWIP_ND6_MAX_UNICAST_SOLICIT   3
```

LWIP_ND6_MAX_UNICAST_SOLICIT: max number of unicast neighbor solicitation messages to send during neighbor reachability detection.

2.1.7.10.7.10 LWIP_ND6_NUM_DESTINATIONS

```
#define LWIP_ND6_NUM_DESTINATIONS   10
```

LWIP_ND6_NUM_DESTINATIONS: number of entries in IPv6 destination cache

2.1.7.10.7.11 LWIP_ND6_NUM_NEIGHBORS

```
#define LWIP_ND6_NUM_NEIGHBORS   10
```

LWIP_ND6_NUM_NEIGHBORS: Number of entries in IPv6 neighbor cache

2.1.7.10.7.12 LWIP_ND6_NUM_PREFIXES

```
#define LWIP_ND6_NUM_PREFIXES   5
```

LWIP_ND6_NUM_PREFIXES: number of entries in IPv6 on-link prefixes cache

2.1.7.10.7.13 LWIP_ND6_NUM_ROUTERS

```
#define LWIP_ND6_NUM_ROUTERS   3
```

LWIP_ND6_NUM_ROUTERS: number of entries in IPv6 default router cache

2.1.7.10.7.14 LWIP_ND6_QUEUEING

```
#define LWIP_ND6_QUEUEING   LWIP_IPV6
```

LWIP_ND6_QUEUEING==1: queue outgoing IPv6 packets while MAC address is being resolved.

2.1.7.10.7.15 LWIP_ND6_RDNSS_MAX_DNS_SERVERS

```
#define LWIP_ND6_RDNSS_MAX_DNS_SERVERS   0
```

LWIP_ND6_RDNSS_MAX_DNS_SERVERS > 0: Use IPv6 Router Advertisement Recursive DNS Server Option (as per RFC 6106) to copy a defined maximum number of DNS servers to the DNS module.

2.1.7.10.7.16 LWIP_ND6_REACHABLE_TIME

```
#define LWIP_ND6_REACHABLE_TIME   30000
```

LWIP_ND6_REACHABLE_TIME: default neighbor reachable time (in milliseconds). May be updated by router advertisement messages.

2.1.7.10.7.17 LWIP_ND6_RETRANS_TIMER

```
#define LWIP_ND6_RETRANS_TIMER   1000
```

LWIP_ND6_RETRANS_TIMER: default retransmission timer for solicitation messages

2.1.7.10.7.18 LWIP_ND6_TCP_REACHABILITY_HINTS

```
#define LWIP_ND6_TCP_REACHABILITY_HINTS   1
```

LWIP_ND6_TCP_REACHABILITY_HINTS==1: Allow TCP to provide Neighbor Discovery with reachability hints for connected destinations. This helps avoid sending unicast neighbor solicitation messages.

2.1.7.10.7.19 MEMP_NUM_ND6_QUEUE

```
#define MEMP_NUM_ND6_QUEUE   20
```

MEMP_NUM_ND6_QUEUE: Max number of IPv6 packets to queue during MAC resolution.

2.1.7.10.8 DHCPv6

2.1.7.10.8.1 Macros

- #define LWIP_IPV6_DHCP6 0
- #define LWIP_IPV6_DHCP6_STATEFUL 0
- #define LWIP_IPV6_DHCP6_STATELESS LWIP_IPV6_DHCP6
- #define LWIP_DHCP6_GET_NTP_SRV 0
- #define LWIP_DHCP6_MAX_NTP_SERVERS 1
- #define LWIP_DHCP6_MAX_DNS_SERVERS DNS_MAX_SERVERS

2.1.7.10.8.2 Detailed Description

2.1.7.10.8.3 Macro Definition Documentation

2.1.7.10.8.4 LWIP_DHCP6_GET_NTP_SRV

```
#define LWIP_DHCP6_GET_NTP_SRV   0
```

LWIP_DHCP6_GETS_NTP==1: Request NTP servers via DHCPv6. For each response packet, a callback is called, which has to be provided by the port: void dhcp6_set_ntp_servers(u8_t num_ntp_servers, ip_addr_t* ntp_server_addrs);

2.1.7.10.8.5 LWIP_DHCP6_MAX_DNS_SERVERS

```
#define LWIP_DHCP6_MAX_DNS_SERVERS   DNS_MAX_SERVERS
```

LWIP_DHCP6_MAX_DNS_SERVERS > 0: Request DNS servers via DHCPv6. DNS servers received in the response are passed to DNS via dns_setserver() (up to the maximum limit defined here).

2.1.7.10.8.6 LWIP_DHCP6_MAX_NTP_SERVERS

```
#define LWIP_DHCP6_MAX_NTP_SERVERS   1
```

The maximum of NTP servers requested

2.1.7.10.8.7 LWIP_IPV6_DHCP6

```
#define LWIP_IPV6_DHCP6   0
```

LWIP_IPV6_DHCP6==1: enable DHCPv6 stateful/stateless address autoconfiguration.

2.1.7.10.8.8 LWIP_IPV6_DHCP6_STATEFUL

```
#define LWIP_IPV6_DHCP6_STATEFUL   0
```

LWIP_IPV6_DHCP6_STATEFUL==1: enable DHCPv6 stateful address autoconfiguration. (not supported, yet!)

2.1.7.10.8.9 LWIP_IPV6_DHCP6_STATELESS

```
#define LWIP_IPV6_DHCP6_STATELESS   LWIP_IPV6_DHCP6
```

LWIP_IPV6_DHCP6_STATELESS==1: enable DHCPv6 stateless address autoconfiguration.

2.2 Infrastructure

2.2.1 Modules

- IP address handling
- Memory pools
- Packet buffers (PBUF)
- Error codes
- IANA assigned numbers
- IEEE assigned numbers

2.2.2 Detailed Description

2.2.3 IP address handling

2.2.3.1 Modules

- IPv4 only
- IPv6 only

2.2.3.2 Data Structures

- struct ip_addr

2.2.3.3 Macros

- #define ip_addr_netcmp(addr1, addr2, mask)   ip_addr_net_eq((addr1), (addr2), (mask))
- #define ip_addr_net_eq(addr1, addr2, mask)
- #define ip_addr_cmp(addr1, addr2)   ip_addr_eq((addr1), (addr2))
- #define ip_addr_eq(addr1, addr2)
- #define ip_addr_cmp_zoneless(addr1, addr2)   ip_addr_zoneless_eq((addr1), (addr2))
- #define ip_addr_zoneless_eq(addr1, addr2)
- #define ip_addr_isany(ipaddr)
- #define ip_addr_isany_val(ipaddr)
- #define ip_addr_isbroadcast(ipaddr, netif)
- #define ip_addr_ismulticast(ipaddr)
- #define ip_addr_isloopback(ipaddr)
- #define ip_addr_islinklocal(ipaddr)
- #define IP_ANY_TYPE   (&ip_addr_any_type)

2.2.3.4 Typedefs

- typedef struct ip_addr ip_addr_t

2.2.3.5 Enumerations

- enum lwip_ip_addr_type { IPADDR_TYPE_V4 = 0U , IPADDR_TYPE_V6 = 6U , IPADDR_TYPE_ANY = 46U }

2.2.3.6 Functions

- char * ipaddr_ntoa (const ip_addr_t *addr)
- char * ipaddr_ntoa_r (const ip_addr_t *addr, char *buf, int buflen)
- int ipaddr_aton (const char *cp, ip_addr_t *addr)

2.2.3.7 Detailed Description

2.2.3.8 Macro Definition Documentation

2.2.3.8.1 ip_addr_cmp

`#define ip_addr_cmp( addr1, addr2)   ip_addr_eq((addr1), (addr2))`

Deprecated Renamed to ip_addr_eq

2.2.3.8.2 ip_addr_cmp_zoneless

`#define ip_addr_cmp_zoneless( addr1, addr2)   ip_addr_zoneless_eq((addr1), (addr2))`

Deprecated Renamed to ip_addr_zoneless_eq

2.2.3.8.3 ip_addr_eq

`#define ip_addr_eq( addr1, addr2)` **Value:**

```
((IP_GET_TYPE(addr1) != IP_GET_TYPE(addr2)) ? 0 : (IP_IS_V6_VAL(*(addr1)) ? \
ip6_addr_eq(ip_2_ip6(addr1), ip_2_ip6(addr2)) : \
ip4_addr_eq(ip_2_ip4(addr1), ip_2_ip4(addr2))))
```

Check if two ip addresses are equal.

2.2.3.8.4 ip_addr_isany

`#define ip_addr_isany( ipaddr)` **Value:**

```
(((ipaddr) == NULL) ? 1 : ((IP_IS_V6(ipaddr)) ? \
ip6_addr_isany(ip_2_ip6(ipaddr)) : \
ip4_addr_isany(ip_2_ip4(ipaddr))))
```

Check if an ip address is the ’any’ address.

2.2.3.8.5 ip_addr_isany_val

`#define ip_addr_isany_val( ipaddr)` **Value:**

```
((IP_IS_V6_VAL(ipaddr)) ? \
ip6_addr_isany_val(*ip_2_ip6(&(ipaddr))) : \
ip4_addr_isany_val(*ip_2_ip4(&(ipaddr))))
```

Check if an ip address is the 'any' address, by value.

2.2.3.8.6 ip_addr_isbroadcast

`#define ip_addr_isbroadcast( ipaddr, netif)` **Value:**

```
((IP_IS_V6(ipaddr)) ? \
0 : \
ip4_addr_isbroadcast(ip_2_ip4(ipaddr), netif))
```

Check if an ip address is a broadcast address.

2.2.3.8.7 ip_addr_islinklocal

`#define ip_addr_islinklocal( ipaddr)` **Value:**

```
((IP_IS_V6(ipaddr)) ? \
ip6_addr_islinklocal(ip_2_ip6(ipaddr)) : \
ip4_addr_islinklocal(ip_2_ip4(ipaddr)))
```

Check inf an ip address is a link-local address.

2.2.3.8.8 ip_addr_isloopback

`#define ip_addr_isloopback( ipaddr)` **Value:**

```
((IP_IS_V6(ipaddr)) ? \
ip6_addr_isloopback(ip_2_ip6(ipaddr)) : \
ip4_addr_isloopback(ip_2_ip4(ipaddr)))
```

Check inf an ip address is a loopback address.

2.2.3.8.9 ip_addr_ismulticast

`#define ip_addr_ismulticast( ipaddr)` **Value:**

```
((IP_IS_V6(ipaddr)) ? \
ip6_addr_ismulticast(ip_2_ip6(ipaddr)) : \
ip4_addr_ismulticast(ip_2_ip4(ipaddr)))
```

Check inf an ip address is a multicast address.

2.2.3.8.10 ip_addr_net_eq

`#define ip_addr_net_eq( addr1, addr2, mask)` **Value:**

```
((IP_IS_V6(addr1) && IP_IS_V6(addr2)) ? \
0 : \
ip4_addr_net_eq(ip_2_ip4(addr1), ip_2_ip4(addr2), mask))
```

Check if two ip addresses are share the same network, for a specific netmask.

2.2.3.8.11 ip_addr_netcmp

`#define ip_addr_netcmp( addr1, addr2, mask) ip_addr_net_eq((addr1), (addr2), (mask))`

Deprecated Renamed to ip_addr_net_eq

2.2.3.8.12 ip_addr_zoneless_eq

`#define ip_addr_zoneless_eq( addr1, addr2)` **Value:**

```
((IP_GET_TYPE(addr1) != IP_GET_TYPE(addr2)) ? 0 : (IP_IS_V6_VAL(*(addr1)) ? \
ip6_addr_zoneless_eq(ip_2_ip6(addr1), ip_2_ip6(addr2)) : \
ip4_addr_eq(ip_2_ip4(addr1), ip_2_ip4(addr2))))
```

Check if two ip addresses are equal, ignoring the zone.

2.2.3.8.13 IP_ANY_TYPE

`#define IP_ANY_TYPE (&ip_addr_any_type)`

Macro representing the 'any' address.

2.2.3.9 Typedef Documentation

2.2.3.9.1 ip_addr_t

`typedef struct ip_addr ip_addr_t`

A union struct for both IP version's addresses. ATTENTION: watch out for its size when adding IPv6 address scope!

2.2.3.10 Enumeration Type Documentation

2.2.3.10.1 lwip_ip_addr_type

`enum lwip_ip_addr_type`

IP address types for use in ip_addr_t.type member. **See also** tcp_new_ip_type(), udp_new_ip_type(), raw_new_ip_type().

IPADDR_TYPE_V4	IPv4
IPADDR_TYPE_V6	IPv6
IPADDR_TYPE_ANY	IPv4+IPv6 ("dual-stack")

2.2.3.11 Function Documentation

2.2.3.11.1 ipaddr_aton()

`int ipaddr_aton (const char * cp, ip_addr_t * addr)`

Convert IP address string (both versions) to numeric. The version is auto-detected from the string.

Parameters

cp	IP address string to convert
addr	conversion result is stored here

Returns 1 on success, 0 on error

2.2.3.11.2 ipaddr_ntoa()

```
char * ipaddr_ntoa (const ip_addr_t * addr)
```

Convert numeric IP address (both versions) into ASCII representation. returns ptr to static buffer; not reentrant!

Parameters

addr	ip address in network order to convert

Returns pointer to a global static (!) buffer that holds the ASCII representation of addr

2.2.3.11.3 ipaddr_ntoa_r()

```
char * ipaddr_ntoa_r (const ip_addr_t * addr, char * buf, int buflen)
```

Same as ipaddr_ntoa, but reentrant since a user-supplied buffer is used.

Parameters

addr	ip address in network order to convert
buf	target buffer where the string is stored
buflen	length of buf

Returns either pointer to buf which now holds the ASCII representation of addr or NULL if buf was too small

2.2.3.12 IPv4 only

2.2.3.12.1 Macros

- #define ip_2_ip4(ipaddr)   (&((ipaddr)->u_addr.ip4))
- #define IP_ADDR_ANY IP4_ADDR_ANY
- #define IP4_ADDR_ANY   (&ip_addr_any)
- #define IP4_ADDR_ANY4   (ip_2_ip4(&ip_addr_any))

2.2.3.12.2 Detailed Description

2.2.3.12.3 Macro Definition Documentation

2.2.3.12.3.1 IP4_ADDR_ANY

```
#define IP4_ADDR_ANY   (&ip_addr_any)
```

Can be used as a fixed/const ip_addr_t for the IPv4 wildcard and the broadcast address

2.2.3.12.3.2 IP4_ADDR_ANY4

```
#define IP4_ADDR_ANY4   (ip_2_ip4(&ip_addr_any))
```

Can be used as a fixed/const ip4_addr_t for the wildcard and the broadcast address

2.2.3.12.3.3 ip_2_ip4

```
#define ip_2_ip4( ipaddr)   (&((ipaddr)->u_addr.ip4))
```

Convert generic ip address to specific protocol version

2.2.3.12.3.4 IP_ADDR_ANY

```
#define IP_ADDR_ANY   IP4_ADDR_ANY
```

Can be used as a fixed/const ip_addr_t for the IP wildcard. Defined to IP4_ADDR_ANY when IPv4 is enabled. Defined to IP6_ADDR_ANY in IPv6 only systems. Use this if you can handle IPv4 *AND* IPv6 addresses. Use IP4_ADDR_ANY or IP6_ADDR_ANY when the IP type matters.

2.2.3.13 IPv6 only

2.2.3.13.1 Macros

- #define ip_2_ip6(ipaddr)   (&((ipaddr)->u_addr.ip6))
- #define IP6_ADDR_ANY   (&ip6_addr_any)
- #define IP6_ADDR_ANY6   (ip_2_ip6(&ip6_addr_any))

2.2.3.13.2 Detailed Description

2.2.3.13.3 Macro Definition Documentation

2.2.3.13.3.1 IP6_ADDR_ANY

```
#define IP6_ADDR_ANY   (&ip6_addr_any)
```

IP6_ADDR_ANY can be used as a fixed ip_addr_t for the IPv6 wildcard address

2.2.3.13.3.2 IP6_ADDR_ANY6

```
#define IP6_ADDR_ANY6   (ip_2_ip6(&ip6_addr_any))
```

IP6_ADDR_ANY6 can be used as a fixed ip6_addr_t for the IPv6 wildcard address

2.2.3.13.3.3 ip_2_ip6

```
#define ip_2_ip6( ipaddr)   (&((ipaddr)->u_addr.ip6))
```

Convert generic ip address to specific protocol version

2.2.4 Memory pools

2.2.4.1 Macros

- #define LWIP_MEMPOOL_PROTOTYPE(name)   extern const struct memp_desc memp_ ## name
- #define LWIP_MEMPOOL_DECLARE(name, num, size, desc)
- #define LWIP_MEMPOOL_INIT(name)   memp_init_pool(&memp_ ## name)
- #define LWIP_MEMPOOL_ALLOC(name)   memp_malloc_pool(&memp_ ## name)
- #define LWIP_MEMPOOL_FREE(name, x)   memp_free_pool(&memp_ ## name, (x))

2.2.4.2 Detailed Description

Custom memory pools

2.2.4.3 Macro Definition Documentation

2.2.4.3.1 LWIP_MEMPOOL_ALLOC

`#define LWIP_MEMPOOL_ALLOC( name)    memp_malloc_pool(&memp_ ## name)`

Allocate from a private memory pool

2.2.4.3.2 LWIP_MEMPOOL_DECLARE

`#define LWIP_MEMPOOL_DECLARE( name, num, size, desc)` **Value:**

```
LWIP_DECLARE_MEMORY_ALIGNED(memp_memory_ ## name ## _base, ((num) * (MEMP_SIZE + ↩
    MEMP_ALIGN_SIZE(size)))); \
  \
LWIP_MEMPOOL_DECLARE_STATS_INSTANCE(memp_stats_ ## name) \
  \
static struct memp *memp_tab_ ## name; \
  \
const struct memp_desc memp_ ## name = { \
  DECLARE_LWIP_MEMPOOL_DESC(desc) \
  LWIP_MEMPOOL_DECLARE_STATS_REFERENCE(memp_stats_ ## name) \
  LWIP_MEM_ALIGN_SIZE(size), \
  (num), \
  memp_memory_ ## name ## _base, \
  &memp_tab_ ## name \
};
```

Declare a private memory pool Private mempools example: .h: only when pool is used in multiple .c files: LWIP_MEMPOOL_PROTOTY
.c:

- in global variables section: LWIP_MEMPOOL_DECLARE(my_private_pool, 10, sizeof(foo), "Some description")
- call ONCE before using pool (e.g. in some init() function): LWIP_MEMPOOL_INIT(my_private_pool);
- allocate: void* my_new_mem = LWIP_MEMPOOL_ALLOC(my_private_pool);
- free: LWIP_MEMPOOL_FREE(my_private_pool, my_new_mem);

To relocate a pool, declare it as extern in cc.h. Example for GCC: extern u8_t __attribute__((section(".onchip_mem"))) memp_memory_

2.2.4.3.3 LWIP_MEMPOOL_FREE

`#define LWIP_MEMPOOL_FREE( name, x)    memp_free_pool(&memp_ ## name, (x))`

Free element from a private memory pool

2.2.4.3.4 LWIP_MEMPOOL_INIT

`#define LWIP_MEMPOOL_INIT( name)    memp_init_pool(&memp_ ## name)`

Initialize a private memory pool

2.2.4.3.5 LWIP_MEMPOOL_PROTOTYPE

`#define LWIP_MEMPOOL_PROTOTYPE( name)    extern const struct memp_desc memp_ ## name`

Declare prototype for private memory pool if it is used in multiple files

2.2.5 Packet buffers (PBUF)

2.2.5.1 Macros

- #define PBUF_NEEDS_COPY(p)   ((p)->type_internal & PBUF_TYPE_FLAG_DATA_VOLATILE)

2.2.5.2 Enumerations

- enum pbuf_layer { PBUF_TRANSPORT = 0 + (14 + 0) + 40 + 20 , PBUF_IP = 0 + (14 + 0) + 40 , PBUF_LINK = 0 + (14 + 0) , PBUF_RAW_TX = 0 , PBUF_RAW = 0 }
- enum pbuf_type { PBUF_RAM = (0x0200 | 0x80 | 0x00) , PBUF_ROM = 0x01 , PBUF_REF = (0x40 | 0x01) , PBUF_POOL = (0x0100 | 0x80 | 0x02) }

2.2.5.3 Functions

- struct pbuf * pbuf_alloc (pbuf_layer layer, u16_t length, pbuf_type type)
- struct pbuf * pbuf_alloc_reference (void *payload, u16_t length, pbuf_type type)
- struct pbuf * pbuf_alloced_custom (pbuf_layer l, u16_t length, pbuf_type type, struct pbuf_custom *p, void *payload_mem, u16_t payload_mem_len)
- void pbuf_realloc (struct pbuf *p, u16_t new_len)
- u8_t pbuf_free (struct pbuf *p)
- void pbuf_ref (struct pbuf *p)
- void pbuf_cat (struct pbuf *h, struct pbuf *t)
- void pbuf_chain (struct pbuf *h, struct pbuf *t)
- err_t pbuf_copy (struct pbuf *p_to, const struct pbuf *p_from)
- err_t pbuf_copy_partial_pbuf (struct pbuf *p_to, const struct pbuf *p_from, u16_t copy_len, u16_t offset)
- u16_t pbuf_copy_partial (const struct pbuf *buf, void *dataptr, u16_t len, u16_t offset)
- void * pbuf_get_contiguous (const struct pbuf *p, void *buffer, size_t bufsize, u16_t len, u16_t offset)
- struct pbuf * pbuf_skip (struct pbuf *in, u16_t in_offset, u16_t *out_offset)
- err_t pbuf_take (struct pbuf *buf, const void *dataptr, u16_t len)
- err_t pbuf_take_at (struct pbuf *buf, const void *dataptr, u16_t len, u16_t offset)
- struct pbuf * pbuf_coalesce (struct pbuf *p, pbuf_layer layer)
- struct pbuf * pbuf_clone (pbuf_layer layer, pbuf_type type, struct pbuf *p)
- u8_t pbuf_get_at (const struct pbuf *p, u16_t offset)
- int pbuf_try_get_at (const struct pbuf *p, u16_t offset)
- void pbuf_put_at (struct pbuf *p, u16_t offset, u8_t data)
- u16_t pbuf_memcmp (const struct pbuf *p, u16_t offset, const void *s2, u16_t n)
- u16_t pbuf_memfind (const struct pbuf *p, const void *mem, u16_t mem_len, u16_t start_offset)

2.2.5.4 Detailed Description

Packets are built from the pbuf data structure. It supports dynamic memory allocation for packet contents or can reference externally managed packet contents both in RAM and ROM. Quick allocation for incoming packets is provided through pools with fixed sized pbufs.

A packet may span over multiple pbufs, chained as a singly linked list. This is called a "pbuf chain".

Multiple packets may be queued, also using this singly linked list. This is called a "packet queue".

So, a packet queue consists of one or more pbuf chains, each of which consist of one or more pbufs. CURRENTLY, PACKET QUEUES ARE NOT SUPPORTED!!! Use helper structs to queue multiple packets.

The differences between a pbuf chain and a packet queue are very precise but subtle.

The last pbuf of a packet has a ->tot_len field that equals the ->len field. It can be found by traversing the list. If the last pbuf of a packet has a ->next field other than NULL, more packets are on the queue.

Therefore, looping through a pbuf of a single packet, has an loop end condition (tot_len == p->len), NOT (next == NULL).

Example of custom pbuf usage: Zero-copy RX

2.2.5.5 Macro Definition Documentation

2.2.5.5.1 PBUF_NEEDS_COPY

```
#define PBUF_NEEDS_COPY( p)   ((p)->type_internal & PBUF_TYPE_FLAG_DATA_VOLATILE)
```

PBUF_NEEDS_COPY(p): return a boolean value indicating whether the given pbuf needs to be copied in order to be kept around beyond the current call stack without risking being corrupted. The default setting provides safety: it will make a copy iof any pbuf chain that does not consist entirely of PBUF_ROM type pbufs. For setups with zero-copy support, it may be redefined to evaluate to true in all cases, for example. However, doing so also has an effect on the application side: any buffers that are *not* copied must also *not* be reused by the application after passing them to lwIP. For example, when setting PBUF_NEEDS_COPY to (0), after using udp_send() with a PBUF_RAM pbuf, the application must free the pbuf immediately, rather than reusing it for other purposes. For more background information on this, see tasks #6735 and #7896, and bugs #11400 and #49914.

2.2.5.6 Enumeration Type Documentation

2.2.5.6.1 pbuf_layer

```
enum pbuf_layer
```

Enumeration of pbuf layers

PBUF_TRANSPORT	Includes spare room for transport layer header, e.g. UDP header. Use this if you intend to pass the pbuf to functions like udp_send().
PBUF_IP	Includes spare room for IP header. Use this if you intend to pass the pbuf to functions like raw_send().
PBUF_LINK	Includes spare room for link layer header (ethernet header). Use this if you intend to pass the pbuf to functions like ethernet_output(). **See also** PBUF_LINK_HLEN
PBUF_RAW_TX	Includes spare room for additional encapsulation header before ethernet headers (e.g. 802.11). Use this if you intend to pass the pbuf to functions like netif->linkoutput(). **See also** PBUF_LINK_ENCAPSULATION_HLEN

PBUF_RAW	Use this for input packets in a netif driver when calling netif->input() in the most common case - ethernet-layer netif driver.

2.2.5.6.2 pbuf_type

`enum pbuf_type`

Enumeration of pbuf types

PBUF_RAM	pbuf data is stored in RAM, used for TX mostly, struct pbuf and its payload are allocated in one piece of contiguous memory (so the first payload byte can be calculated from struct pbuf). pbuf_alloc() allocates PBUF_RAM pbufs as unchained pbufs (although that might change in future versions). This should be used for all OUTGOING packets (TX).
PBUF_ROM	pbuf data is stored in ROM, i.e. struct pbuf and its payload are located in totally different memory areas. Since it points to ROM, payload does not have to be copied when queued for transmission.
PBUF_REF	pbuf comes from the pbuf pool. Much like PBUF_ROM but payload might change so it has to be duplicated when queued before transmitting, depending on who has a 'ref' to it.
PBUF_POOL	pbuf payload refers to RAM. This one comes from a pool and should be used for RX. Payload can be chained (scatter-gather RX) but like PBUF_RAM, struct pbuf and its payload are allocated in one piece of contiguous memory (so the first payload byte can be calculated from struct pbuf). Don't use this for TX, if the pool becomes empty e.g. because of TCP queuing, you are unable to receive TCP acks!

2.2.5.7 Function Documentation

2.2.5.7.1 pbuf_alloc()

`struct pbuf * pbuf_alloc (pbuf_layer layer, u16_t length, pbuf_type type)`

Allocates a pbuf of the given type (possibly a chain for PBUF_POOL type).

The actual memory allocated for the pbuf is determined by the layer at which the pbuf is allocated and the requested size (from the size parameter).

Parameters

layer	header size
length	size of the pbuf's payload
type	this parameter decides how and where the pbuf should be allocated as follows:

- PBUF_RAM: buffer memory for pbuf is allocated as one large chunk. This includes protocol headers as well.
- PBUF_ROM: no buffer memory is allocated for the pbuf, even for protocol headers. Additional headers must be prepended by allocating another pbuf and chain in to the front of the ROM pbuf. It is assumed that the memory used is really similar to ROM in that it is immutable and will not be changed. Memory which is dynamic should generally not be attached to PBUF_ROM pbufs. Use PBUF_REF instead.

- PBUF_REF: no buffer memory is allocated for the pbuf, even for protocol headers. It is assumed that the pbuf is only being used in a single thread. If the pbuf gets queued, then pbuf_take should be called to copy the buffer.
- PBUF_POOL: the pbuf is allocated as a pbuf chain, with pbufs from the pbuf pool that is allocated during pbuf_init().

Returns the allocated pbuf. If multiple pbufs where allocated, this is the first pbuf of a pbuf chain.

2.2.5.7.2 pbuf_alloc_reference()

```
struct pbuf * pbuf_alloc_reference (void * payload, u16_t length, pbuf_type type)
```

Allocates a pbuf for referenced data. Referenced data can be volatile (PBUF_REF) or long-lived (PBUF_ROM).

The actual memory allocated for the pbuf is determined by the layer at which the pbuf is allocated and the requested size (from the size parameter).

Parameters

payload	referenced payload
length	size of the pbuf's payload
type	this parameter decides how and where the pbuf should be allocated as follows:

- PBUF_ROM: It is assumed that the memory used is really similar to ROM in that it is immutable and will not be changed. Memory which is dynamic should generally not be attached to PBUF_ROM pbufs. Use PBUF_REF instead.
- PBUF_REF: It is assumed that the pbuf is only being used in a single thread. If the pbuf gets queued, then pbuf_take should be called to copy the buffer.

Returns the allocated pbuf.

2.2.5.7.3 pbuf_alloced_custom()

```
struct pbuf * pbuf_alloced_custom (pbuf_layer l, u16_t length, pbuf_type type, struct pbuf
* p, void * payload_mem, u16_t payload_mem_len)
```

Initialize a custom pbuf (already allocated). Example of custom pbuf usage: Zero-copy RX

Parameters

l	header size
length	size of the pbuf's payload
type	type of the pbuf (only used to treat the pbuf accordingly, as this function allocates no memory)
p	pointer to the custom pbuf to initialize (already allocated)
payload_mem	pointer to the buffer that is used for payload and headers, must be at least big enough to hold ’length’ plus the header size, may be NULL if set later. ATTENTION: The caller is responsible for correct alignment of this buffer!!
payload_mem_len	the size of the ’payload_mem’ buffer, must be at least big enough to hold ’length’ plus the header size

2.2.5.7.4 pbuf_cat()

```
void pbuf_cat (struct pbuf * h, struct pbuf * t)
```

Concatenate two pbufs (each may be a pbuf chain) and take over the caller's reference of the tail pbuf.

> **Note**
> The caller MAY NOT reference the tail pbuf afterwards. Use pbuf_chain() for that purpose.

This function explicitly does not check for tot_len overflow to prevent failing to queue too long pbufs. This can produce invalid pbufs, so handle with care!

See also pbuf_chain()

2.2.5.7.5 pbuf_chain()

```
void pbuf_chain (struct pbuf * h, struct pbuf * t)
```

Chain two pbufs (or pbuf chains) together.

The caller MUST call pbuf_free(t) once it has stopped using it. Use pbuf_cat() instead if you no longer use t.

Parameters

h	head pbuf (chain)
t	tail pbuf (chain)

> **Note**
> The pbufs MUST belong to the same packet.
> MAY NOT be called on a packet queue.

The ->tot_len fields of all pbufs of the head chain are adjusted. The ->next field of the last pbuf of the head chain is adjusted. The ->ref field of the first pbuf of the tail chain is adjusted.

2.2.5.7.6 pbuf_clone()

```
struct pbuf * pbuf_clone (pbuf_layer layer, pbuf_type type, struct pbuf * p)
```

Allocates a new pbuf of same length (via pbuf_alloc()) and copies the source pbuf into this new pbuf (using pbuf_copy()).

Parameters

layer	pbuf_layer of the new pbuf
type	this parameter decides how and where the pbuf should be allocated (

See also pbuf_alloc())

Parameters

p	the source pbuf

Returns a new pbuf or NULL if allocation fails

2.2.5.7.7 pbuf_coalesce()

```
struct pbuf * pbuf_coalesce (struct pbuf * p, pbuf_layer layer)
```

Creates a single pbuf out of a queue of pbufs.

Remarks : Either the source pbuf 'p' is freed by this function or the original pbuf 'p' is returned, therefore the caller has to check the result!

Parameters

p	the source pbuf
layer	pbuf_layer of the new pbuf

Returns a new, single pbuf (p->next is NULL) or the old pbuf if allocation fails

2.2.5.7.8 pbuf_copy()

```
err_t pbuf_copy (struct pbuf * p_to, const struct pbuf * p_from)
```

Copy the contents of one packet buffer into another.

Note
Only one packet is copied, no packet queue!

Parameters

p_to	pbuf destination of the copy
p_from	pbuf source of the copy

Returns ERR_OK if pbuf was copied ERR_ARG if one of the pbufs is NULL or p_to is not big enough to hold p_from ERR_VAL if any of the pbufs are part of a queue

2.2.5.7.9 pbuf_copy_partial()

```
u16_t pbuf_copy_partial (const struct pbuf * buf, void * dataptr, u16_t len, u16_t offset)
```

Copy (part of) the contents of a packet buffer to an application supplied buffer.

Parameters

buf	the pbuf from which to copy data
dataptr	the application supplied buffer
len	length of data to copy (dataptr must be big enough). No more than buf->tot_len will be copied, irrespective of len
offset	offset into the packet buffer from where to begin copying len bytes

Returns the number of bytes copied, or 0 on failure

2.2.5.7.10 pbuf_copy_partial_pbuf()

```
err_t pbuf_copy_partial_pbuf (struct pbuf * p_to, const struct pbuf * p_from, u16_t copy_l
u16_t offset)
```

Copy part or all of one packet buffer into another, to a specified offset.

Note
Only data in one packet is copied, no packet queue!
Argument order is shared with pbuf_copy, but different than pbuf_copy_partial.

Parameters

p_to	pbuf destination of the copy
p_from	pbuf source of the copy
copy_len	number of bytes to copy
offset	offset in destination pbuf where to copy to

Returns ERR_OK if copy_len bytes were copied ERR_ARG if one of the pbufs is NULL or p_from is shorter than copy_len or p_to is not big enough to hold copy_len at offset ERR_VAL if any of the pbufs are part of a queue

2.2.5.7.11 pbuf_free()

```
u8_t pbuf_free (struct pbuf * p)
```

Dereference a pbuf chain or queue and deallocate any no-longer-used pbufs at the head of this chain or queue.

Decrements the pbuf reference count. If it reaches zero, the pbuf is deallocated.

For a pbuf chain, this is repeated for each pbuf in the chain, up to the first pbuf which has a non-zero reference count after decrementing. So, when all reference counts are one, the whole chain is free'd.

Parameters

p	The pbuf (chain) to be dereferenced.

Returns the number of pbufs that were de-allocated from the head of the chain.

Note
the reference counter of a pbuf equals the number of pointers that refer to the pbuf (or into the pbuf).

2.2.5.7.12 pbuf_get_at()

```
u8_t pbuf_get_at (const struct pbuf * p, u16_t offset)
```

Get one byte from the specified position in a pbuf WARNING: returns zero for offset >= p->tot_len

Parameters

p	pbuf to parse
offset	offset into p of the byte to return

Returns byte at an offset into p OR ZERO IF 'offset' >= p->tot_len

2.2.5.7.13 pbuf_get_contiguous()

```
void * pbuf_get_contiguous (const struct pbuf * p, void * buffer, size_t bufsize, u16_t
len, u16_t offset)
```

Get part of a pbuf's payload as contiguous memory. The returned memory is either a pointer into the pbuf's payload or, if split over multiple pbufs, a copy into the user-supplied buffer.

Parameters

p	the pbuf from which to copy data
buffer	the application supplied buffer. May be NULL if the caller does not want to copy. In this case, offset + len should be checked against p->tot_len, since there's no way for the caller to know why NULL is returned.
bufsize	size of the application supplied buffer (when buffer is != NULL)
len	length of data to copy (p and buffer must be big enough)
offset	offset into the packet buffer from where to begin copying len bytes

Returns - pointer into pbuf payload if that is already contiguous (no copy needed)

- pointer to 'buffer' if data was not contiguous and had to be copied
- NULL on error

2.2.5.7.14 pbuf_memcmp()

```
u16_t pbuf_memcmp (const struct pbuf * p, u16_t offset, const void * s2, u16_t n)
```

Compare pbuf contents at specified offset with memory s2, both of length n

Parameters

p	pbuf to compare
offset	offset into p at which to start comparing
s2	buffer to compare
n	length of buffer to compare

Returns zero if equal, nonzero otherwise (0xffff if p is too short, diffoffset+1 otherwise)

2.2.5.7.15 pbuf_memfind()

```
u16_t pbuf_memfind (const struct pbuf * p, const void * mem, u16_t mem_len, u16_t start_of
```

Find occurrence of mem (with length mem_len) in pbuf p, starting at offset start_offset.

Parameters

p	pbuf to search, maximum length is 0xFFFE since 0xFFFF is used as return value 'not found'
mem	search for the contents of this buffer
mem_len	length of 'mem'
start_offset	offset into p at which to start searching

Returns 0xFFFF if substr was not found in p or the index where it was found

2.2.5.7.16 pbuf_put_at()

```
void pbuf_put_at (struct pbuf * p, u16_t offset, u8_t data)
```

Put one byte to the specified position in a pbuf WARNING: silently ignores offset >= p->tot_len

Parameters

p	pbuf to fill
offset	offset into p of the byte to write
data	byte to write at an offset into p

2.2.5.7.17 pbuf_realloc()

```
void pbuf_realloc (struct pbuf * p, u16_t new_len)
```

Shrink a pbuf chain to a desired length.

Parameters

p	pbuf to shrink.
new_len	desired new length of pbuf chain

Depending on the desired length, the first few pbufs in a chain might be skipped and left unchanged. The new last pbuf in the chain will be resized, and any remaining pbufs will be freed.

Note
If the pbuf is ROM/REF, only the ->tot_len and ->len fields are adjusted.
May not be called on a packet queue.
Despite its name, pbuf_realloc cannot grow the size of a pbuf (chain).

2.2.5.7.18 pbuf_ref()

```
void pbuf_ref (struct pbuf * p)
```

Increment the reference count of the pbuf.

Parameters

p	pbuf to increase reference counter of

2.2.5.7.19 pbuf_skip()

```
struct pbuf * pbuf_skip (struct pbuf * in, u16_t in_offset, u16_t * out_offset)
```

Skip a number of bytes at the start of a pbuf

Parameters

Returns the pbuf in the queue where the offset is or NULL when the offset is too high

in	input pbuf
in_offset	offset to skip
out_offset	resulting offset in the returned pbuf

buf	pbuf to fill with data
dataptr	application supplied data buffer
len	length of the application supplied data buffer

2.2.5.7.20 pbuf_take()

`err_t pbuf_take (struct pbuf * buf, const void * dataptr, u16_t len)`

Copy application supplied data into a pbuf. This function can only be used to copy the equivalent of buf->tot_len data.

Parameters

Returns ERR_OK if successful, ERR_MEM if the pbuf is not big enough

2.2.5.7.21 pbuf_take_at()

`err_t pbuf_take_at (struct pbuf * buf, const void * dataptr, u16_t len, u16_t offset)`

Same as pbuf_take() but puts data at an offset

Parameters

buf	pbuf to fill with data
dataptr	application supplied data buffer
len	length of the application supplied data buffer
offset	offset in pbuf where to copy dataptr to

Returns ERR_OK if successful, ERR_MEM if the pbuf is not big enough

2.2.5.7.22 pbuf_try_get_at()

`int pbuf_try_get_at (const struct pbuf * p, u16_t offset)`

Get one byte from the specified position in a pbuf

Parameters

Returns byte at an offset into p [0..0xFF] OR negative if 'offset' >= p->tot_len

2.2.6 Error codes

2.2.6.1 Typedefs

- typedef s8_t err_t

2.2.6.2 Enumerations

- enum err_enum_t { ERR_OK = 0 , ERR_MEM = -1 , ERR_BUF = -2 , ERR_TIMEOUT = -3 , ERR_RTE = -4 , ERR_INPROGRESS = -5 , ERR_VAL = -6 , ERR_WOULDBLOCK = -7 , ERR_USE = -8 , ERR_ALREADY = -9 , ERR_ISCONN = -10 , ERR_CONN = -11 , ERR_IF = -12 , ERR_ABRT = -13 , ERR_RST = -14 , ERR_CLSD = -15 , ERR_ARG = -16 }

p	pbuf to parse
offset	offset into p of the byte to return

2.2.6.3 Detailed Description

2.2.6.4 Typedef Documentation

2.2.6.4.1 err_t

```
typedef s8_t err_t
```

Define LWIP_ERR_T in cc.h if you want to use a different type for your platform (must be signed).

2.2.6.5 Enumeration Type Documentation

2.2.6.5.1 err_enum_t

```
enum err_enum_t
```

Definitions for error constants.

ERR_OK	No error, everything OK.
ERR_MEM	Out of memory error.
ERR_BUF	Buffer error.
ERR_TIMEOUT	Timeout.
ERR_RTE	Routing problem.
ERR_INPROGRESS	Operation in progress
ERR_VAL	Illegal value.
ERR_WOULDBLOCK	Operation would block.
ERR_USE	Address in use.
ERR_ALREADY	Already connecting.
ERR_ISCONN	Conn already established.
ERR_CONN	Not connected.
ERR_IF	Low-level netif error
ERR_ABRT	Connection aborted.
ERR_RST	Connection reset.
ERR_CLSD	Connection closed.
ERR_ARG	Illegal argument.

2.2.7 IANA assigned numbers

2.2.7.1 Enumerations

- enum lwip_iana_hwtype { LWIP_IANA_HWTYPE_ETHERNET = 1 }
- enum lwip_iana_port_number { LWIP_IANA_PORT_SMTP = 25 , LWIP_IANA_PORT_DHCP_SERVER = 67 , LWIP_IANA_PORT
= 68 , LWIP_IANA_PORT_TFTP = 69 , LWIP_IANA_PORT_HTTP = 80 , LWIP_IANA_PORT_SNTP = 123 , LWIP_IANA_PORT_
= 137 , LWIP_IANA_PORT_SNMP = 161 , LWIP_IANA_PORT_SNMP_TRAP = 162 , LWIP_IANA_PORT_HTTPS =
443 , LWIP_IANA_PORT_SMTPS = 465 , LWIP_IANA_PORT_MQTT = 1883 , LWIP_IANA_PORT_MDNS = 5353 ,
LWIP_IANA_PORT_SECURE_MQTT = 8883 }

2.2.7.2 Detailed Description

2.2.7.3 Enumeration Type Documentation

2.2.7.3.1 lwip_iana_hwtype

`enum lwip_iana_hwtype`

Hardware types

LWIP_IANA_HWTYPE_ETHERNET	Ethernet

2.2.7.3.2 lwip_iana_port_number

`enum lwip_iana_port_number`

Port numbers https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt

LWIP_IANA_PORT_SMTP	SMTP
LWIP_IANA_PORT_DHCP_SERVER	DHCP server
LWIP_IANA_PORT_DHCP_CLIENT	DHCP client
LWIP_IANA_PORT_TFTP	TFTP
LWIP_IANA_PORT_HTTP	HTTP
LWIP_IANA_PORT_SNTP	SNTP
LWIP_IANA_PORT_NETBIOS	NETBIOS
LWIP_IANA_PORT_SNMP	SNMP
LWIP_IANA_PORT_SNMP_TRAP	SNMP traps
LWIP_IANA_PORT_HTTPS	HTTPS
LWIP_IANA_PORT_SMTPS	SMTPS
LWIP_IANA_PORT_MQTT	MQTT
LWIP_IANA_PORT_MDNS	MDNS
LWIP_IANA_PORT_SECURE_MQTT	Secure MQTT

2.2.8 IEEE assigned numbers

2.2.8.1 Enumerations

- enum lwip_ieee_eth_type { ETHTYPE_IP = 0x0800U , ETHTYPE_ARP = 0x0806U , ETHTYPE_WOL = 0x0842U , ETHTYPE_RA = 0x8035U , ETHTYPE_VLAN = 0x8100U , ETHTYPE_IPV6 = 0x86DDU , ETHTYPE_PPPOEDISC = 0x8863U , ETHTYPE_PPP = 0x8864U , ETHTYPE_JUMBO = 0x8870U , ETHTYPE_PROFINET = 0x8892U , ETHTYPE_ETHERCAT = 0x88A4U , ETHTYPE_LLDP = 0x88CCU , ETHTYPE_SERCOS = 0x88CDU , ETHTYPE_MRP = 0x88E3U , ETHTYPE_PTP = 0x88F7U , ETHTYPE_QINQ = 0x9100U }

2.2.8.2 Detailed Description

2.2.8.3 Enumeration Type Documentation

2.2.8.3.1 lwip_ieee_eth_type

`enum lwip_ieee_eth_type`

A list of often ethtypes (although lwIP does not use all of them).

ETHTYPE_IP	Internet protocol v4
ETHTYPE_ARP	Address resolution protocol
ETHTYPE_WOL	Wake on lan
ETHTYPE_RARP	RARP
ETHTYPE_VLAN	Virtual local area network
ETHTYPE_IPV6	Internet protocol v6
ETHTYPE_PPPOEDISC	PPP Over Ethernet Discovery Stage
ETHTYPE_PPPOE	PPP Over Ethernet Session Stage
ETHTYPE_JUMBO	Jumbo Frames
ETHTYPE_PROFINET	Process field network
ETHTYPE_ETHERCAT	Ethernet for control automation technology
ETHTYPE_LLDP	Link layer discovery protocol
ETHTYPE_SERCOS	Serial real-time communication system
ETHTYPE_MRP	Media redundancy protocol
ETHTYPE_PTP	Precision time protocol
ETHTYPE_QINQ	Q-in-Q, 802.1ad

2.3 APIs

2.3.1 Modules

- "raw" APIs
- Sequential-style APIs
- Socket API

2.3.2 Detailed Description

lwIP provides three Application Program's Interfaces (APIs) for programs to use for communication with the TCP/IP code:

- low-level "core" / "callback" or "raw" APIs.
- higher-level Sequential-style APIs.
- BSD-style Socket API.

The raw TCP/IP interface allows the application program to integrate better with the TCP/IP code. Program execution is event based by having callback functions being called from within the TCP/IP code. The TCP/IP code and the application program both run in the same thread. The sequential API has a much higher overhead and is not very well suited for small systems since it forces a multithreaded paradigm on the application.

The raw TCP/IP interface is not only faster in terms of code execution time but is also less memory intensive. The drawback is that program development is somewhat harder and application programs written for the raw TCP/IP interface are more difficult to understand. Still, this is the preferred way of writing applications that should be small in code size and memory usage.

All APIs can be used simultaneously by different application programs. In fact, the sequential API is implemented as an application program using the raw TCP/IP interface.

Do not confuse the lwIP raw API with raw Ethernet or IP sockets. The former is a way of interfacing the lwIP network stack (including TCP and UDP), the latter refers to processing raw Ethernet or IP data instead of TCP connections or UDP packets.

Raw API applications may never block since all packet processing (input and output) as well as timer processing (TCP mainly) is done in a single execution context.

2.3.3 "raw" APIs

2.3.3.1 Modules

- Application layered TCP Introduction
- DNS
- IP
- Network interface (NETIF)
- RAW
- TCP
- UDP
- Ethernet

2.3.3.2 Detailed Description

Non thread-safe APIs, callback style for maximum performance and minimum memory footprint. Program execution is driven by callbacks functions, which are then invoked by the lwIP core when activity related to that application occurs. A particular application may register to be notified via a callback function for events such as incoming data available, outgoing data sent, error notifications, poll timer expiration, connection closed, etc. An application can provide a callback function to perform processing for any or all of these events. Each callback is an ordinary C function that is called from within the TCP/IP code. Every callback function is passed the current TCP or UDP connection state as an argument. Also, in order to be able to keep program specific state, the callback functions are called with a program specified argument that is independent of the TCP/IP state. The raw API (sometimes called native API) is an event-driven API designed to be used without an operating system that implements zero-copy send and receive. This API is also used by the core stack for interaction between the various protocols. It is the only API available when running lwIP without an operating system.

2.3.3.3 Application layered TCP Introduction

2.3.3.3.1 Modules

- Application layered TCP Functions

2.3.3.3.2 Detailed Description

Overview

altcp (application layered TCP connection API; to be used from TCPIP thread) is an abstraction layer that prevents applications linking hard against the tcp.h functions while providing the same functionality. It is used to e.g. add SSL/TLS (see LWIP_ALTCP_TLS) or proxy-connect support to an application written for the tcp callback API without that application knowing the protocol details.

- This interface mimics the tcp callback API to the application while preventing direct linking (much like virtual functions).
- This way, an application can make use of other application layer protocols on top of TCP without knowing the details (e.g. TLS, proxy connection).
- This is achieved by simply including "lwip/altcp.h" instead of "lwip/tcp.h", replacing "struct tcp_pcb" with "struct altcp_pcb" and prefixing all functions with "altcp_" instead of "tcp_".

With altcp support disabled (LWIP_ALTCP==0), applications written against the altcp API can still be compiled but are directly linked against the tcp.h callback API and then cannot use layered protocols. To minimize code changes in this case, the use of altcp_allocators is strongly suggested.

Usage

To make use of this API from an existing tcp raw API application:

- Include "lwip/altcp.h" instead of "lwip/tcp.h"
- Replace "struct tcp_pcb" with "struct altcp_pcb"
- Prefix all called tcp API functions with "altcp_" instead of "tcp_" to link against the altcp functions
- altcp_new (and altcp_new_ip_type / altcp_new_ip6) take an altcp_allocator_t as an argument, whereas the original tcp API functions take no arguments.
- An altcp_allocator_t allocator is an object that holds a pointer to an allocator object and a corresponding state (e.g. for TLS, the corresponding state may hold certificates or keys). This way, the application does not even need to know if it uses TLS or pure TCP, this is handled at runtime by passing a specific allocator.
- An application can alternatively bind hard to the altcp_tls API by calling altcp_tls_new or altcp_tls_wrap.
- The TLS layer is not directly implemented by lwIP, but a port to mbedTLS is provided.
- Another altcp layer is proxy-connect to use TLS behind a HTTP proxy (see altcp_proxyconnect.h)

altcp_allocator_t

An altcp allocator is created by the application by combining an allocator callback function and a corresponding state, e.g.:

```
static const unsigned char cert[] = {0x2D, ... (see mbedTLS doc for how to create ↩
    this)};
struct altcp_tls_config * conf = altcp_tls_create_config_client(cert, sizeof(cert) ↩
    );
altcp_allocator_t tls_allocator = {
  altcp_tls_alloc, conf
};
```

struct altcp_tls_config

The struct altcp_tls_config holds state that is needed to create new TLS client or server connections (e.g. certificates and private keys).

It is not defined by lwIP itself but by the TLS port (e.g. altcp_tls to mbedTLS adaption). However, the parameters used to create it are defined in altcp_tls.h (see altcp_tls_create_config_server_privkey_cert for servers and altcp_tls_create_config_client / altcp_tls_create_config_client_2wayauth for clients).

For mbedTLS, ensure that certificates can be parsed by 'mbedtls_x509_crt_parse()' and private keys can be parsed by 'mbedtls_pk_parse_

2.3.3.3.3 Application layered TCP Functions

2.3.3.3.3.1 Modules

- TLS layer

2.3.3.3.3.2 Data Structures

- struct altcp_allocator_s

2.3.3.3.3.3 Typedefs

- typedef struct altcp_allocator_s altcp_allocator_t

2.3.3.3.3.4 Functions

- struct altcp_pcb * altcp_new_ip6 (altcp_allocator_t *allocator)
- struct altcp_pcb * altcp_new (altcp_allocator_t *allocator)
- struct altcp_pcb * altcp_new_ip_type (altcp_allocator_t *allocator, u8_t ip_type)
- void altcp_arg (struct altcp_pcb *conn, void *arg)
- void altcp_accept (struct altcp_pcb *conn, altcp_accept_fn accept)
- void altcp_recv (struct altcp_pcb *conn, altcp_recv_fn recv)
- void altcp_sent (struct altcp_pcb *conn, altcp_sent_fn sent)
- void altcp_poll (struct altcp_pcb *conn, altcp_poll_fn poll, u8_t interval)
- void altcp_err (struct altcp_pcb *conn, altcp_err_fn err)
- void altcp_recved (struct altcp_pcb *conn, u16_t len)
- err_t altcp_bind (struct altcp_pcb *conn, const ip_addr_t *ipaddr, u16_t port)
- err_t altcp_connect (struct altcp_pcb *conn, const ip_addr_t *ipaddr, u16_t port, altcp_connected_fn connected)
- struct altcp_pcb * altcp_listen_with_backlog_and_err (struct altcp_pcb *conn, u8_t backlog, err_t *err)
- void altcp_abort (struct altcp_pcb *conn)
- err_t altcp_close (struct altcp_pcb *conn)
- err_t altcp_shutdown (struct altcp_pcb *conn, int shut_rx, int shut_tx)
- err_t altcp_write (struct altcp_pcb *conn, const void *dataptr, u16_t len, u8_t apiflags)
- err_t altcp_output (struct altcp_pcb *conn)
- u16_t altcp_mss (struct altcp_pcb *conn)
- u16_t altcp_sndbuf (struct altcp_pcb *conn)
- u16_t altcp_sndqueuelen (struct altcp_pcb *conn)
- void altcp_setprio (struct altcp_pcb *conn, u8_t prio)

2.3.3.3.3.5 Detailed Description

This file contains the common functions for altcp to work. For more details see Application layered TCP Introduction.

2.3.3.3.3.6 Typedef Documentation

2.3.3.3.3.7 altcp_allocator_t

```
typedef struct altcp_allocator_s altcp_allocator_t
```

Struct containing an allocator and its state.

2.3.3.3.3.8 Function Documentation

2.3.3.3.3.9 altcp_abort()

```
void altcp_abort (struct altcp_pcb * conn)
```

See also tcp_abort()

2.3.3.3.3.10 altcp_accept()

```
void altcp_accept (struct altcp_pcb * conn, altcp_accept_fn accept)
```

See also tcp_accept()

2.3.3.3.3.11 altcp_arg()

```
void altcp_arg (struct altcp_pcb * conn, void * arg)
```

See also tcp_arg()

2.3.3.3.3.12 altcp_bind()

```
err_t altcp_bind (struct altcp_pcb * conn, const ip_addr_t * ipaddr, u16_t port)
```

See also tcp_bind()

2.3.3.3.3.13 altcp_close()

```
err_t altcp_close (struct altcp_pcb * conn)
```

See also tcp_close()

2.3.3.3.3.14 altcp_connect()

```
err_t altcp_connect (struct altcp_pcb * conn, const ip_addr_t * ipaddr, u16_t port, altcp_
connected)
```

See also tcp_connect()

2.3.3.3.3.15 altcp_err()

```
void altcp_err (struct altcp_pcb * conn, altcp_err_fn err)
```

See also tcp_err()

2.3.3.3.3.16 altcp_listen_with_backlog_and_err()

```
struct altcp_pcb * altcp_listen_with_backlog_and_err (struct altcp_pcb * conn, u8_t backlo
err_t * err)
```

See also tcp_listen_with_backlog_and_err()

2.3.3.3.3.17 altcp_mss()

```
u16_t altcp_mss (struct altcp_pcb * conn)
```

See also tcp_mss()

2.3.3.3.3.18 altcp_new()

```
struct altcp_pcb * altcp_new (altcp_allocator_t * allocator)
```

altcp_new: altcp_new for IPv4

2.3.3.3.3.19 altcp_new_ip6()

```
struct altcp_pcb * altcp_new_ip6 (altcp_allocator_t * allocator)
```

altcp_new_ip6: altcp_new for IPv6

2.3.3.3.3.20 altcp_new_ip_type()

```
struct altcp_pcb * altcp_new_ip_type (altcp_allocator_t * allocator, u8_t ip_type)
```

altcp_new_ip_type: called by applications to allocate a new pcb with the help of an allocator function.

Parameters

allocator	allocator function and argument
ip_type	IP version of the pcb (lwip_ip_addr_type)

Returns a new altcp_pcb or NULL on error

2.3.3.3.3.21 altcp_output()

```
err_t altcp_output (struct altcp_pcb * conn)
```

See also tcp_output()

2.3.3.3.3.22 altcp_poll()

```
void altcp_poll (struct altcp_pcb * conn, altcp_poll_fn poll, u8_t interval)
```

See also tcp_poll()

2.3.3.3.3.23 altcp_recv()

```
void altcp_recv (struct altcp_pcb * conn, altcp_recv_fn recv)
```

See also tcp_recv()

2.3.3.3.3.24 altcp_recved()

```
void altcp_recved (struct altcp_pcb * conn, u16_t len)
```

See also tcp_recved()

2.3.3.3.3.25 altcp_sent()

```
void altcp_sent (struct altcp_pcb * conn, altcp_sent_fn sent)
```

See also tcp_sent()

2.3.3.3.3.26 altcp_setprio()

```
void altcp_setprio (struct altcp_pcb * conn, u8_t prio)
```

See also tcp_setprio()

2.3.3.3.3.27 altcp_shutdown()

```
err_t altcp_shutdown (struct altcp_pcb * conn, int shut_rx, int shut_tx)
```

See also tcp_shutdown()

2.3.3.3.3.28 altcp_sndbuf()

```
u16_t altcp_sndbuf (struct altcp_pcb * conn)
```

See also tcp_sndbuf()

2.3.3.3.3.29 altcp_sndqueuelen()

```
u16_t altcp_sndqueuelen (struct altcp_pcb * conn)
```

See also tcp_sndqueuelen()

2.3.3.3.3.30 altcp_write()

```
err_t altcp_write (struct altcp_pcb * conn, const void * dataptr, u16_t len, u8_t apiflags
```

See also tcp_write()

2.3.3.3.3.31 TLS layer

2.3.3.3.3.32 Functions

- struct altcp_tls_config * altcp_tls_create_config_server (u8_t cert_count)
- err_t altcp_tls_config_server_add_privkey_cert (struct altcp_tls_config *config, const u8_t *privkey, size_t privkey_len, const u8_t *privkey_pass, size_t privkey_pass_len, const u8_t *cert, size_t cert_len)
- struct altcp_tls_config * altcp_tls_create_config_server_privkey_cert (const u8_t *privkey, size_t privkey_len, const u8_t *privkey_pass, size_t privkey_pass_len, const u8_t *cert, size_t cert_len)
- struct altcp_tls_config * altcp_tls_create_config_client (const u8_t *cert, size_t cert_len)
- struct altcp_tls_config * altcp_tls_create_config_client_2wayauth (const u8_t *ca, size_t ca_len, const u8_t *privkey, size_t privkey_len, const u8_t *privkey_pass, size_t privkey_pass_len, const u8_t *cert, size_t cert_len)
- int altcp_tls_configure_alpn_protocols (struct altcp_tls_config *conf, const char **protos)
- void altcp_tls_free_config (struct altcp_tls_config *conf)
- void altcp_tls_free_entropy (void)
- struct altcp_pcb * altcp_tls_wrap (struct altcp_tls_config *config, struct altcp_pcb *inner_pcb)
- struct altcp_pcb * altcp_tls_new (struct altcp_tls_config *config, u8_t ip_type)
- struct altcp_pcb * altcp_tls_alloc (void *arg, u8_t ip_type)
- void * altcp_tls_context (struct altcp_pcb *conn)
- void altcp_tls_init_session (struct altcp_tls_session *dest)

- err_t altcp_tls_get_session (struct altcp_pcb *conn, struct altcp_tls_session *dest)
- err_t altcp_tls_set_session (struct altcp_pcb *conn, struct altcp_tls_session *from)
- void altcp_tls_free_session (struct altcp_tls_session *dest)

2.3.3.3.3.33 Detailed Description

This file contains function prototypes for a TLS layer. A port to ARM mbedtls is provided in the apps/ tree (LWIP_ALTCP_TLS_MBEDT
option).

2.3.3.3.3.34 Function Documentation

2.3.3.3.3.35 altcp_tls_alloc()

```
struct altcp_pcb * altcp_tls_alloc (void * arg, u8_t ip_type)
```

Create new ALTCP_TLS layer pcb and its inner tcp pcb. Same as altcp_tls_new but this allocator function fits to altcp_allocator_t / altcp_new. 'arg' must contain a struct altcp_tls_config *.

This standard allocator function creates an altcp pcb for TLS over TCP

2.3.3.3.3.36 altcp_tls_config_server_add_privkey_cert()

```
err_t altcp_tls_config_server_add_privkey_cert (struct altcp_tls_config * config, const
u8_t * privkey, size_t privkey_len, const u8_t * privkey_pass, size_t privkey_pass_len,
const u8_t * cert, size_t cert_len)
```

Add a certificate to an ALTCP_TLS server configuration handle

2.3.3.3.3.37 altcp_tls_configure_alpn_protocols()

```
int altcp_tls_configure_alpn_protocols (struct altcp_tls_config * conf, const char ** prot
```

Configure ALPN TLS extension Example: static const char *g_alpn_protocols[] = { "x-amzn-mqtt-ca", NULL }; tls_config = altcp_tls_create_config_client(ca, ca_len); altcp_tls_conf_alpn_protocols(tls_config, g_alpn_protocols);

2.3.3.3.3.38 altcp_tls_context()

```
void * altcp_tls_context (struct altcp_pcb * conn)
```

Return pointer to internal TLS context so application can tweak it. Real type depends on port (e.g. mbedtls)

2.3.3.3.3.39 altcp_tls_create_config_client()

```
struct altcp_tls_config * altcp_tls_create_config_client (const u8_t * cert, size_t cert_l
```

Create an ALTCP_TLS client configuration handle

2.3.3.3.3.40 altcp_tls_create_config_client_2wayauth()

```
struct altcp_tls_config * altcp_tls_create_config_client_2wayauth (const u8_t * ca, size_t
ca_len, const u8_t * privkey, size_t privkey_len, const u8_t * privkey_pass, size_t privke
const u8_t * cert, size_t cert_len)
```

Create an ALTCP_TLS client configuration handle with two-way server/client authentication

2.3.3.3.3.41 altcp_tls_create_config_server()

```
struct altcp_tls_config * altcp_tls_create_config_server (u8_t cert_count)
```

Create an ALTCP_TLS server configuration handle prepared for multiple certificates

2.3.3.3.3.42 altcp_tls_create_config_server_privkey_cert()

```
struct altcp_tls_config * altcp_tls_create_config_server_privkey_cert (const u8_t * privke
size_t privkey_len, const u8_t * privkey_pass, size_t privkey_pass_len, const u8_t * cert,
size_t cert_len)
```

Create an ALTCP_TLS server configuration handle with one certificate (short version of calling altcp_tls_create_config_server and altcp_tls_config_server_add_privkey_cert)

2.3.3.3.3.43 altcp_tls_free_config()

```
void altcp_tls_free_config (struct altcp_tls_config * conf)
```

Free an ALTCP_TLS configuration handle

2.3.3.3.3.44 altcp_tls_free_entropy()

```
void altcp_tls_free_entropy (void )
```

Free an ALTCP_TLS global entropy instance. All ALTCP_TLS configuration are linked to one altcp_tls_entropy_rng structure that handle an unique system entropy & ctr_drbg instance. This function allow application to free this altcp_tls_entropy_rng structure when all configuration referencing it were destroyed. This function does nothing if some ALTCP_TLS configuration handle are still active.

2.3.3.3.3.45 altcp_tls_free_session()

```
void altcp_tls_free_session (struct altcp_tls_session * dest)
```

Free allocated data inside a TLS session buffer. Real type depends on port (e.g. mbedtls use mbedtls_ssl_session)

2.3.3.3.3.46 altcp_tls_get_session()

```
err_t altcp_tls_get_session (struct altcp_pcb * conn, struct altcp_tls_session * dest)
```

Save current connected session to reuse it later. Should be called after altcp_connect() succeeded. Return error if saving session fail. Real type depends on port (e.g. mbedtls use mbedtls_ssl_session)

2.3.3.3.3.47 altcp_tls_init_session()

```
void altcp_tls_init_session (struct altcp_tls_session * dest)
```

Initialise a TLS session buffer. Real type depends on port (e.g. mbedtls use mbedtls_ssl_session)

2.3.3.3.3.48 altcp_tls_new()

```
struct altcp_pcb * altcp_tls_new (struct altcp_tls_config * config, u8_t ip_type)
```

Create new ALTCP_TLS pcb and its inner tcp pcb

This standard allocator function creates an altcp pcb for TLS over TCP

2.3.3.3.3.49 altcp_tls_set_session()

```
err_t altcp_tls_set_session (struct altcp_pcb * conn, struct altcp_tls_session * from)
```

Restore a previously saved session. Must be called before altcp_connect(). Return error if cannot restore session. Real type depends on port (e.g. mbedtls use mbedtls_ssl_session)

2.3.3.3.3.50 altcp_tls_wrap()

```
struct altcp_pcb * altcp_tls_wrap (struct altcp_tls_config * config, struct altcp_pcb *
inner_pcb)
```

Create new ALTCP_TLS layer wrapping an existing pcb as inner connection (e.g. TLS over TCP)

2.3.3.4 DNS

2.3.3.4.1 Functions

- void dns_setserver (u8_t numdns, const ip_addr_t *dnsserver)
- const ip_addr_t * dns_getserver (u8_t numdns)
- err_t dns_gethostbyname (const char *hostname, ip_addr_t *addr, dns_found_callback found, void *callback_arg)
- err_t dns_gethostbyname_addrtype (const char *hostname, ip_addr_t *addr, dns_found_callback found, void *callback_arg, u8_t dns_addrtype)

2.3.3.4.2 Detailed Description

Implements a DNS host name to IP address resolver.

The lwIP DNS resolver functions are used to lookup a host name and map it to a numerical IP address. It maintains a list of resolved hostnames that can be queried with the dns_lookup() function. New hostnames can be resolved using the dns_query() function.

The lwIP version of the resolver also adds a non-blocking version of gethostbyname() that will work with a raw API application. This function checks for an IP address string first and converts it if it is valid. gethostbyname() then does a dns_lookup() to see if the name is already in the table. If so, the IP is returned. If not, a query is issued and the function returns with a ERR_INPROGRESS status. The app using the dns client must then go into a waiting state.

Once a hostname has been resolved (or found to be non-existent), the resolver code calls a specified callback function (which must be implemented by the module that uses the resolver).

Multicast DNS queries are supported for names ending on ".local". However, only "One-Shot Multicast DNS Queries" are supported (RFC 6762 chapter 5.1), this is not a fully compliant implementation of continuous mDNS querying!

All functions must be called from TCPIP thread.

See also DNS_MAX_SERVERS

LWIP_DHCP_MAX_DNS_SERVERS

Common functions for thread-safe access.

2.3.3.4.3 Function Documentation

2.3.3.4.3.1 dns_gethostbyname()

```
err_t dns_gethostbyname (const char * hostname, ip_addr_t * addr, dns_found_callback found
void * callback_arg)
```

Resolve a hostname (string) into an IP address. NON-BLOCKING callback version for use with raw API!!!

Returns immediately with one of err_t return codes:

- ERR_OK if hostname is a valid IP address string or the host name is already in the local names table.
- ERR_INPROGRESS enqueue a request to be sent to the DNS server for resolution if no errors are present.
- ERR_ARG: dns client not initialized or invalid hostname

Parameters

hostname	the hostname that is to be queried
addr	pointer to a ip_addr_t where to store the address if it is already cached in the dns_table (only valid if ERR_OK is returned!)
found	a callback function to be called on success, failure or timeout (only if ERR_INPROGRESS is returned!)
callback_arg	argument to pass to the callback function

Returns a err_t return code.

2.3.3.4.3.2 dns_gethostbyname_addrtype()

```
err_t dns_gethostbyname_addrtype (const char * hostname, ip_addr_t * addr, dns_found_callb
found, void * callback_arg, u8_t dns_addrtype)
```

Like dns_gethostbyname, but returned address type can be controlled:

Parameters

hostname	the hostname that is to be queried
addr	pointer to a ip_addr_t where to store the address if it is already cached in the dns_table (only valid if ERR_OK is returned!)
found	a callback function to be called on success, failure or timeout (only if ERR_INPROGRESS is returned!)
callback_arg	argument to pass to the callback function
dns_addrtype	- LWIP_DNS_ADDRTYPE_IPV4_IPV6: try to resolve IPv4 first, try IPv6 if IPv4 fails only • LWIP_DNS_ADDRTYPE_IPV6_IPV4: try to resolve IPv6 first, try IPv4 if IPv6 fails only • LWIP_DNS_ADDRTYPE_IPV4: try to resolve IPv4 only • LWIP_DNS_ADDRTYPE_IPV6: try to resolve IPv6 only

2.3.3.4.3.3 dns_getserver()

```
const ip_addr_t * dns_getserver (u8_t numdns)
```

Obtain one of the currently configured DNS server.

Parameters

numdns	the index of the DNS server

Returns IP address of the indexed DNS server or "ip_addr_any" if the DNS server has not been configured.

2.3.3.4.3.4 dns_setserver()

`void dns_setserver (u8_t numdns, const ip_addr_t * dnsserver)`

Initialize one of the DNS servers.

Parameters

numdns	the index of the DNS server to set must be < DNS_MAX_SERVERS
dnsserver	IP address of the DNS server to set

2.3.3.5 IP

2.3.3.5.1 Modules

- IPv4
- IPv6

2.3.3.5.2 Macros

- #define ip_output(p, src, dest, ttl, tos, proto)
- #define ip_output_if(p, src, dest, ttl, tos, proto, netif)
- #define ip_output_if_src(p, src, dest, ttl, tos, proto, netif)
- #define ip_route(src, dest)
- #define ip_netif_get_local_ip(netif, dest)

2.3.3.5.3 Detailed Description

2.3.3.5.4 Macro Definition Documentation

2.3.3.5.4.1 ip_netif_get_local_ip

`#define ip_netif_get_local_ip( netif, dest)` **Value:**

```
        (IP_IS_V6(dest) ? \
        ip6_netif_get_local_ip(netif, ip_2_ip6(dest)) : \
        ip4_netif_get_local_ip(netif))
```

Get netif for IP.

2.3.3.5.4.2 ip_output

`#define ip_output( p, src, dest, ttl, tos, proto)` **Value:**

```
        (IP_IS_V6(dest) ? \
        ip6_output(p, ip_2_ip6(src), ip_2_ip6(dest), ttl, tos, proto) : \
        ip4_output(p, ip_2_ip4(src), ip_2_ip4(dest), ttl, tos, proto))
```

Output IP packet, netif is selected by source address

2.3.3.5.4.3 ip_output_if

`#define ip_output_if( p, src, dest, ttl, tos, proto, netif)` **Value:**

```
        (IP_IS_V6(dest) ? \
        ip6_output_if(p, ip_2_ip6(src), ip_2_ip6(dest), ttl, tos, proto, netif) : \
        ip4_output_if(p, ip_2_ip4(src), ip_2_ip4(dest), ttl, tos, proto, netif))
```

Output IP packet to specified interface

2.3.3.5.4.4 ip_output_if_src

`#define ip_output_if_src( p, src, dest, ttl, tos, proto, netif)` **Value:**

```
        (IP_IS_V6(dest) ? \
        ip6_output_if_src(p, ip_2_ip6(src), ip_2_ip6(dest), ttl, tos, proto, netif) : \
        ip4_output_if_src(p, ip_2_ip4(src), ip_2_ip4(dest), ttl, tos, proto, netif))
```

Output IP packet to interface specifying source address

2.3.3.5.4.5 ip_route

`#define ip_route( src, dest)` **Value:**

```
        (IP_IS_V6(dest) ? \
        ip6_route(ip_2_ip6(src), ip_2_ip6(dest)) : \
        ip4_route_src(ip_2_ip4(src), ip_2_ip4(dest)))
```

Get netif for address combination. See ip6_route and ip4_route

2.3.3.5.5 IPv4

2.3.3.5.5.1 Modules

- ACD
- AUTOIP
- DHCPv4
- IGMP

2.3.3.5.5.2 Functions

- void ip4_set_default_multicast_netif (struct netif *default_multicast_netif)

2.3.3.5.5.3 Detailed Description

2.3.3.5.5.4 Function Documentation

2.3.3.5.5.5 ip4_set_default_multicast_netif()

`void ip4_set_default_multicast_netif (struct netif * default_multicast_netif)`

Set a default netif for IPv4 multicast.

2.3.3.5.5.6 ACD

2.3.3.5.5.7 Functions

- err_t acd_add (struct netif *netif, struct acd *acd, acd_conflict_callback_t acd_conflict_callback)
- void acd_remove (struct netif *netif, struct acd *acd)
- err_t acd_start (struct netif *netif, struct acd *acd, ip4_addr_t ipaddr)
- err_t acd_stop (struct acd *acd)
- void acd_network_changed_link_down (struct netif *netif)
- void acd_netif_ip_addr_changed (struct netif *netif, const ip_addr_t *old_addr, const ip_addr_t *new_addr)

2.3.3.5.5.8 Detailed Description

ACD related functions USAGE:

define LWIP_ACD 1 in your lwipopts.h Options: ACD_TMR_INTERVAL msecs, I recommend a value of 100. The value must divide 1000 with a remainder almost 0. Possible values are 1000, 500, 333, 250, 200, 166, 142, 125, 111, 100 ....

For fixed IP:

- call acd_start after selecting an IP address. The caller will be informed on conflict status via the callback function.

With AUTOIP:

- will be called from the autoip module. No extra's needed.

With DHCP:

- enable LWIP_DHCP_DOES_ACD_CHECK. Then it will be called from the dhcp module. No extra's needed.

2.3.3.5.5.9 Function Documentation

2.3.3.5.5.10 acd_add()

```
err_t acd_add (struct netif * netif, struct acd * acd, acd_conflict_callback_t acd_conflic
```

Add ACD client to the client list and initialize callback function

Parameters

netif	network interface on which to start the acd client
acd	acd module to be added to the list
acd_conflict_callback	callback to be called when conflict information is available

2.3.3.5.5.11 acd_netif_ip_addr_changed()

```
void acd_netif_ip_addr_changed (struct netif * netif, const ip_addr_t * old_addr, const
ip_addr_t * new_addr)
```

Inform the ACD modules of address changes

Parameters

netif	network interface on which the address is changing
old_addr	old ip address
new_addr	new ip address

netif	network interface on which to inform the ACD clients

2.3.3.5.5.12 acd_network_changed_link_down()

```
void acd_network_changed_link_down (struct netif * netif)
```

Inform the ACD modules when the link goes down

Parameters

2.3.3.5.5.13 acd_remove()

```
void acd_remove (struct netif * netif, struct acd * acd)
```

Remove ACD client from the client list

Parameters

netif	network interface from which to remove the acd client
acd	acd module to be removed from the list

2.3.3.5.5.14 acd_start()

```
err_t acd_start (struct netif * netif, struct acd * acd, ip4_addr_t ipaddr)
```

Start ACD client

Parameters

2.3.3.5.5.15 acd_stop()

```
err_t acd_stop (struct acd * acd)
```

Stop ACD client

Parameters

2.3.3.5.5.16 AUTOIP

2.3.3.5.5.17 Functions

- void autoip_set_struct (struct netif *netif, struct autoip *autoip)
- void autoip_remove_struct (struct netif *netif)
- err_t autoip_start (struct netif *netif)
- err_t autoip_stop (struct netif *netif)

netif	network interface on which to start the acd client
acd	acd module to start
ipaddr	ip address to perform acd on

acd	acd module to stop

2.3.3.5.5.18 Detailed Description

AUTOIP related functions USAGE:

define LWIP_AUTOIP 1 in your lwipopts.h

Without DHCP:

- Call autoip_start() after netif_add().

With DHCP:

- define LWIP_DHCP_AUTOIP_COOP 1 in your lwipopts.h.
- Configure your DHCP Client.

See also AUTOIP

2.3.3.5.5.19 Function Documentation

2.3.3.5.5.20 autoip_remove_struct()

`void autoip_remove_struct (struct netif * netif)`

Remove a struct autoip previously set to the netif using autoip_set_struct()

Parameters

netif	the netif for which to set the struct autoip

2.3.3.5.5.21 autoip_set_struct()

`void autoip_set_struct (struct netif * netif, struct autoip * autoip)`

Set a statically allocated struct autoip to work with. Using this prevents autoip_start to allocate it using mem_malloc.

Parameters

2.3.3.5.5.22 autoip_start()

`err_t autoip_start (struct netif * netif)`

Start AutoIP client

Parameters

2.3.3.5.5.23 autoip_stop()

`err_t autoip_stop (struct netif * netif)`

Stop AutoIP client

Parameters

netif	the netif for which to set the struct autoip
autoip	(uninitialised) autoip struct allocated by the application

netif	network interface on which start the AutoIP client

2.3.3.5.5.24 DHCPv4

2.3.3.5.5.25 Functions

- void dhcp_set_struct (struct netif *netif, struct dhcp *dhcp)
- void dhcp_cleanup (struct netif *netif)
- err_t dhcp_start (struct netif *netif)
- void dhcp_inform (struct netif *netif)
- err_t dhcp_renew (struct netif *netif)
- void dhcp_release_and_stop (struct netif *netif)
- err_t dhcp_release (struct netif *netif)
- void dhcp_stop (struct netif *netif)

2.3.3.5.5.26 Detailed Description

DHCP (IPv4) related functions This is a DHCP client for the lwIP TCP/IP stack. It aims to conform with RFC 2131 and RFC 2132.

Options: DHCP_COARSE_TIMER_SECS (recommended 60 which is a minute) DHCP_FINE_TIMER_MSECS (recommended 500 which equals TCP coarse timer)

dhcp_start() starts a DHCP client instance which configures the interface by obtaining an IP address lease and maintaining it.

Use dhcp_release() to end the lease and use dhcp_stop() to remove the DHCP client.

See also LWIP_HOOK_DHCP_APPEND_OPTIONS

LWIP_HOOK_DHCP_PARSE_OPTION

DHCPv4

2.3.3.5.5.27 Function Documentation

2.3.3.5.5.28 dhcp_cleanup()

```
void dhcp_cleanup (struct netif * netif)
```

Removes a struct dhcp from a netif.

Parameters

netif	network interface on which stop the AutoIP client

netif	the netif from which to remove the struct dhcp

2.3.3.5.5.29 dhcp_inform()

```
void dhcp_inform (struct netif * netif)
```

Inform a DHCP server of our manual configuration.

This informs DHCP servers of our fixed IP address configuration by sending an INFORM message. It does not involve DHCP address configuration, it is just here to be nice to the network.

Parameters

netif	The lwIP network interface

2.3.3.5.5.30 dhcp_release()

```
err_t dhcp_release (struct netif * netif)
```

This function calls dhcp_release_and_stop() internally. Deprecated Use dhcp_release_and_stop() instead.

2.3.3.5.5.31 dhcp_release_and_stop()

```
void dhcp_release_and_stop (struct netif * netif)
```

Release a DHCP lease and stop DHCP statemachine (and AUTOIP if LWIP_DHCP_AUTOIP_COOP).

Parameters

netif	network interface

2.3.3.5.5.32 dhcp_renew()

```
err_t dhcp_renew (struct netif * netif)
```

Renew an existing DHCP lease at the involved DHCP server.

Parameters

2.3.3.5.5.33 dhcp_set_struct()

```
void dhcp_set_struct (struct netif * netif, struct dhcp * dhcp)
```

Set a statically allocated struct dhcp to work with. Using this prevents dhcp_start to allocate it using mem_malloc.

Parameters

2.3.3.5.5.34 dhcp_start()

```
err_t dhcp_start (struct netif * netif)
```

Start DHCP negotiation for a network interface.

If no DHCP client instance was attached to this interface, a new client is created first. If a DHCP client instance was already present, it restarts negotiation.

Parameters

Returns lwIP error code

netif	network interface which must renew its lease

netif	the netif for which to set the struct dhcp
dhcp	(uninitialised) dhcp struct allocated by the application

- ERR_OK - No error
- ERR_MEM - Out of memory

2.3.3.5.5.35 dhcp_stop()

```
void dhcp_stop (struct netif * netif)
```

This function calls dhcp_release_and_stop() internally. Deprecated Use dhcp_release_and_stop() instead.

2.3.3.5.5.36 IGMP

2.3.3.5.5.37 Macros

- #define netif_igmp_data(netif) ((struct igmp_group *)netif_get_client_data(netif, LWIP_NETIF_CLIENT_DATA_INDEX_IGMP))

2.3.3.5.5.38 Functions

- err_t igmp_joingroup (const ip4_addr_t *ifaddr, const ip4_addr_t *groupaddr)
- err_t igmp_joingroup_netif (struct netif *netif, const ip4_addr_t *groupaddr)
- err_t igmp_leavegroup (const ip4_addr_t *ifaddr, const ip4_addr_t *groupaddr)
- err_t igmp_leavegroup_netif (struct netif *netif, const ip4_addr_t *groupaddr)

2.3.3.5.5.39 Detailed Description

To be called from TCPIP thread

2.3.3.5.5.40 Macro Definition Documentation

2.3.3.5.5.41 netif_igmp_data

```
#define netif_igmp_data( netif)   ((struct igmp_group *)netif_get_client_data(netif, LWIP_
```

Get list head of IGMP groups for netif. Note: The allsystems group IP is contained in the list as first entry. **See also** netif_set_igmp_mac_filter()

2.3.3.5.5.42 Function Documentation

2.3.3.5.5.43 igmp_joingroup()

```
err_t igmp_joingroup (const ip4_addr_t * ifaddr, const ip4_addr_t * groupaddr)
```

Join a group on one network interface.

Parameters

Returns ERR_OK if group was joined on the netif(s), an err_t otherwise

netif	The lwIP network interface

ifaddr	ip address of the network interface which should join a new group
groupaddr	the ip address of the group which to join

2.3.3.5.5.44 igmp_joingroup_netif()

`err_t igmp_joingroup_netif (struct netif * netif, const ip4_addr_t * groupaddr)`

Join a group on one network interface.

Parameters

netif	the network interface which should join a new group
groupaddr	the ip address of the group which to join

Returns ERR_OK if group was joined on the netif, an err_t otherwise

2.3.3.5.5.45 igmp_leavegroup()

`err_t igmp_leavegroup (const ip4_addr_t * ifaddr, const ip4_addr_t * groupaddr)`

Leave a group on one network interface.

Parameters

ifaddr	ip address of the network interface which should leave a group
groupaddr	the ip address of the group which to leave

Returns ERR_OK if group was left on the netif(s), an err_t otherwise

2.3.3.5.5.46 igmp_leavegroup_netif()

`err_t igmp_leavegroup_netif (struct netif * netif, const ip4_addr_t * groupaddr)`

Leave a group on one network interface.

Parameters

Returns ERR_OK if group was left on the netif, an err_t otherwise

2.3.3.5.6 IPv6

2.3.3.5.6.1 Modules

- DHCPv6
- MLD6
- IPv6 Zones

2.3.3.5.6.2 Functions

- const ip_addr_t * ip6_select_source_address (struct netif *netif, const ip6_addr_t *dest)

netif	the network interface which should leave a group
groupaddr	the ip address of the group which to leave

2.3.3.5.6.3 Detailed Description

2.3.3.5.6.4 Function Documentation

2.3.3.5.6.5 ip6_select_source_address()

```
const ip_addr_t * ip6_select_source_address (struct netif * netif, const ip6_addr_t * dest
```

Select the best IPv6 source address for a given destination IPv6 address.

This implementation follows RFC 6724 Sec. 5 to the following extent:

- Rules 1, 2, 3: fully implemented
- Rules 4, 5, 5.5: not applicable
- Rule 6: not implemented
- Rule 7: not applicable
- Rule 8: limited to "prefer /64 subnet match over non-match"

For Rule 2, we deliberately deviate from RFC 6724 Sec. 3.1 by considering ULAs to be of smaller scope than global addresses, to avoid that a preferred ULA is picked over a deprecated global address when given a global address as destination, as that would likely result in broken two-way communication.

As long as temporary addresses are not supported (as used in Rule 7), a proper implementation of Rule 8 would obviate the need to implement Rule 6.

Parameters

netif	the netif on which to send a packet
dest	the destination we are trying to reach (possibly not properly zoned)

Returns the most suitable source address to use, or NULL if no suitable source address is found

2.3.3.5.6.6 DHCPv6

2.3.3.5.6.7 Functions

- void dhcp6_set_struct (struct netif *netif, struct dhcp6 *dhcp6)
- void dhcp6_cleanup (struct netif *netif)
- err_t dhcp6_enable_stateful (struct netif *netif)
- err_t dhcp6_enable_stateless (struct netif *netif)
- void dhcp6_disable (struct netif *netif)

2.3.3.5.6.8 Detailed Description

DHCPv6 client: IPv6 address autoconfiguration as per RFC 3315 (stateful DHCPv6) and RFC 3736 (stateless DHCPv6).

For now, only stateless DHCPv6 is implemented!

TODO:

- enable/disable API to not always start when RA is received
- stateful DHCPv6 (for now, only stateless DHCPv6 for DNS and NTP servers works)
- create Client Identifier?
- only start requests if a valid local address is available on the netif
- only start information requests if required (not for every RA)

dhcp6_enable_stateful() enables stateful DHCPv6 for a netif (stateless disabled) dhcp6_enable_stateless() enables stateless DHCPv6 for a netif (stateful disabled) dhcp6_disable() disable DHCPv6 for a netif

When enabled, requests are only issued after receipt of RA with the corresponding bits set.

2.3.3.5.6.9 Function Documentation

2.3.3.5.6.10 dhcp6_cleanup()

```
void dhcp6_cleanup (struct netif * netif)
```

Removes a struct dhcp6 from a netif.

ATTENTION: Only use this when not using dhcp6_set_struct() to allocate the struct dhcp6 since the memory is passed back to the heap.

Parameters

netif	the netif from which to remove the struct dhcp

2.3.3.5.6.11 dhcp6_disable()

```
void dhcp6_disable (struct netif * netif)
```

Disable stateful or stateless DHCPv6 on this netif Requests are sent on receipt of an RA message with the ND6_RA_FLAG_OTHER_CO flag set.

2.3.3.5.6.12 dhcp6_enable_stateful()

```
err_t dhcp6_enable_stateful (struct netif * netif)
```

Enable stateful DHCPv6 on this netif Requests are sent on receipt of an RA message with the ND6_RA_FLAG_MANAGED_ADDR_CC flag set.

A struct dhcp6 will be allocated for this netif if not set via dhcp6_set_struct before.

2.3.3.5.6.13 dhcp6_enable_stateless()

```
err_t dhcp6_enable_stateless (struct netif * netif)
```

Enable stateless DHCPv6 on this netif Requests are sent on receipt of an RA message with the ND6_RA_FLAG_OTHER_CONFIG flag set.

A struct dhcp6 will be allocated for this netif if not set via dhcp6_set_struct before.

2.3.3.5.6.14 dhcp6_set_struct()

```
void dhcp6_set_struct (struct netif * netif, struct dhcp6 * dhcp6)
```

Set a statically allocated struct dhcp6 to work with. Using this prevents dhcp6_start to allocate it using mem_malloc.

Parameters

netif	the netif for which to set the struct dhcp
dhcp6	(uninitialised) dhcp6 struct allocated by the application

2.3.3.5.6.15 MLD6

2.3.3.5.6.16 Macros

- #define netif_mld6_data(netif) ((struct mld_group *)netif_get_client_data(netif, LWIP_NETIF_CLIENT_DATA_INDEX_MLD6))

2.3.3.5.6.17 Functions

- err_t mld6_joingroup (const ip6_addr_t *srcaddr, const ip6_addr_t *groupaddr)
- err_t mld6_joingroup_netif (struct netif *netif, const ip6_addr_t *groupaddr)
- err_t mld6_leavegroup (const ip6_addr_t *srcaddr, const ip6_addr_t *groupaddr)
- err_t mld6_leavegroup_netif (struct netif *netif, const ip6_addr_t *groupaddr)

2.3.3.5.6.18 Detailed Description

Multicast listener discovery for IPv6. Aims to be compliant with RFC 2710. No support for MLDv2. Note: The allnodes (ff01::1, ff02::1) group is assumed be received by your netif since it must always be received for correct IPv6 operation (e.g. SLAAC). Ensure the netif filters are configured accordingly! The netif flags also need NETIF_FLAG_MLD6 flag set to enable MLD6 on a netif ("netif->flags |= NETIF_FLAG_MLD6;"). To be called from TCPIP thread.

2.3.3.5.6.19 Macro Definition Documentation

2.3.3.5.6.20 netif_mld6_data

```
#define netif_mld6_data( netif)   ((struct mld_group *)netif_get_client_data(netif, LWIP_N
```

Get list head of MLD6 groups for netif. Note: The allnodes group IP is NOT in the list, since it must always be received for correct IPv6 operation. **See also** netif_set_mld_mac_filter()

2.3.3.5.6.21 Function Documentation

2.3.3.5.6.22 mld6_joingroup()

```
err_t mld6_joingroup (const ip6_addr_t * srcaddr, const ip6_addr_t * groupaddr)
```

Join a group on one or all network interfaces.

If the group is to be joined on all interfaces, the given group address must not have a zone set (i.e., it must have its zone index set to IP6_NO_ZONE). If the group is to be joined on one particular interface, the given group address may or may not have a zone set.

Parameters

Returns ERR_OK if group was joined on the netif(s), an err_t otherwise

srcaddr	ipv6 address (zoned) of the network interface which should join a new group. If IP6_ADDR_ANY6, join on all netifs
groupaddr	the ipv6 address of the group to join (possibly but not necessarily zoned)

2.3.3.5.6.23 mld6_joingroup_netif()

```
err_t mld6_joingroup_netif (struct netif * netif, const ip6_addr_t * groupaddr)
```

Join a group on a network interface.

Parameters

netif	the network interface which should join a new group.
groupaddr	the ipv6 address of the group to join (possibly but not necessarily zoned)

Returns ERR_OK if group was joined on the netif, an err_t otherwise

2.3.3.5.6.24 mld6_leavegroup()

```
err_t mld6_leavegroup (const ip6_addr_t * srcaddr, const ip6_addr_t * groupaddr)
```

Leave a group on a network interface.

Zoning of address follows the same rules as mld6_joingroup.

Parameters

srcaddr	ipv6 address (zoned) of the network interface which should leave the group. If IP6_ADDR_ANY6, leave on all netifs
groupaddr	the ipv6 address of the group to leave (possibly, but not necessarily zoned)

Returns ERR_OK if group was left on the netif(s), an err_t otherwise

2.3.3.5.6.25 mld6_leavegroup_netif()

```
err_t mld6_leavegroup_netif (struct netif * netif, const ip6_addr_t * groupaddr)
```

Leave a group on a network interface.

Parameters

Returns ERR_OK if group was left on the netif, an err_t otherwise

2.3.3.5.6.26 IPv6 Zones

2.3.3.5.6.27 Macros

- #define IP6_NO_ZONE 0
- #define IPADDR6_ZONE_INIT , IP6_NO_ZONE
- #define ip6_addr_zone(ip6addr) ((ip6addr)->zone)
- #define ip6_addr_has_zone(ip6addr) (ip6_addr_zone(ip6addr) != IP6_NO_ZONE)
- #define ip6_addr_set_zone(ip6addr, zone_idx) ((ip6addr)->zone = (zone_idx))

netif	the network interface which should leave the group.
groupaddr	the ipv6 address of the group to leave (possibly, but not necessarily zoned)

- #define ip6_addr_clear_zone(ip6addr)   ((ip6addr)->zone = IP6_NO_ZONE)
- #define ip6_addr_copy_zone(ip6addr1, ip6addr2)   ((ip6addr1).zone = (ip6addr2).zone)
- #define ip6_addr_equals_zone(ip6addr, zone_idx)   ((ip6addr)->zone == (zone_idx))
- #define ip6_addr_cmp_zone(addr1, addr2)   ip6_addr_zone_eq(ip6addr1, ip6addr2)
- #define ip6_addr_zone_eq(ip6addr1, ip6addr2)   ((ip6addr1)->zone == (ip6addr2)->zone)
- #define IPV6_CUSTOM_SCOPES   0
- #define ip6_addr_has_scope(ip6addr, type)
- #define ip6_addr_assign_zone(ip6addr, type, netif)
- #define ip6_addr_test_zone(ip6addr, netif)   (ip6_addr_equals_zone((ip6addr), netif_get_index(netif)))
- #define ip6_addr_lacks_zone(ip6addr, type)   (!ip6_addr_has_zone(ip6addr) && ip6_addr_has_scope((ip6addr), (type)))
- #define ip6_addr_select_zone(dest, src)

2.3.3.5.6.28 Enumerations

- enum lwip_ipv6_scope_type { IP6_UNKNOWN = 0 , IP6_UNICAST = 1 , IP6_MULTICAST = 2 }

2.3.3.5.6.29 Detailed Description

2.3.3.5.6.30 Macro Definition Documentation

2.3.3.5.6.31 ip6_addr_assign_zone

`#define ip6_addr_assign_zone( ip6addr, type, netif)` **Value:**

```
    (ip6_addr_set_zone((ip6addr), \
      ip6_addr_has_scope((ip6addr), (type)) ? netif_get_index(netif) : 0))
```

Assign a zone index to an IPv6 address, based on a network interface. If the given address has a scope, the assigned zone index is that scope's zone of the given netif; otherwise, the assigned zone index is "no zone".

This default implementation follows the default model of RFC 4007, where only interface-local and link-local scopes are defined, and the zone index of both of those scopes always equals the index of the network interface. As such, this default implementation need not distinguish between different constrained scopes when assigning the zone.

Parameters

ip6addr	the IPv6 address; its address part is examined, and its zone index is assigned.
type	address type; see lwip_ipv6_scope_type.
netif	the network interface (const).

2.3.3.5.6.32 ip6_addr_clear_zone

`#define ip6_addr_clear_zone( ip6addr)    ((ip6addr)->zone = IP6_NO_ZONE)`

Clear the zone field of an IPv6 address, setting it to "no zone".

2.3.3.5.6.33 ip6_addr_cmp_zone

```
#define ip6_addr_cmp_zone( addr1, addr2)   ip6_addr_zone_eq(ip6addr1, ip6addr2)
```

Deprecated Renamed to ip6_addr_zone_eq

2.3.3.5.6.34 ip6_addr_copy_zone

```
#define ip6_addr_copy_zone( ip6addr1, ip6addr2)   ((ip6addr1).zone = (ip6addr2).zone)
```

Copy the zone field from the second IPv6 address to the first one.

2.3.3.5.6.35 ip6_addr_equals_zone

```
#define ip6_addr_equals_zone( ip6addr, zone_idx)   ((ip6addr)->zone == (zone_idx))
```

Is the zone field of the given IPv6 address equal to the given zone index? (0/1)

2.3.3.5.6.36 ip6_addr_has_scope

`#define ip6_addr_has_scope( ip6addr, type)` **Value:**

```
(ip6_addr_islinklocal(ip6addr) || (((type) != IP6_UNICAST) && \
 (ip6_addr_ismulticast_iflocal(ip6addr) || \
  ip6_addr_ismulticast_linklocal(ip6addr))))
```

Determine whether an IPv6 address has a constrained scope, and as such is meaningful only if accompanied by a zone index to identify the scope's zone. The given address type may be used to eliminate at compile time certain checks that will evaluate to false at run time anyway.

This default implementation follows the default model of RFC 4007, where only interface-local and link-local scopes are defined.

Even though the unicast loopback address does have an implied link-local scope, in this implementation it does not have an explicitly assigned zone index. As such it should not be tested for in this macro.

Parameters

ip6addr	the IPv6 address (const); only its address part is examined.
type	address type; see lwip_ipv6_scope_type.

Returns 1 if the address has a constrained scope, 0 if it does not.

2.3.3.5.6.37 ip6_addr_has_zone

```
#define ip6_addr_has_zone( ip6addr)   (ip6_addr_zone(ip6addr) != IP6_NO_ZONE)
```

Does the given IPv6 address have a zone set? (0/1)

2.3.3.5.6.38 ip6_addr_lacks_zone

```
#define ip6_addr_lacks_zone( ip6addr, type)   (!ip6_addr_has_zone(ip6addr) && ip6_addr_ha
(type)))
```

Does the given IPv6 address have a scope, and as such should also have a zone to be meaningful, but does not actually have a zone? (0/1)

2.3.3.5.6.39 ip6_addr_select_zone

`#define ip6_addr_select_zone( dest, src)` **Value:**

```
do { struct netif *selected_netif; \
selected_netif = ip6_route((src), (dest)); \
if (selected_netif != NULL) { \
  ip6_addr_assign_zone((dest), IP6_UNKNOWN, selected_netif); \
} } while (0)
```

Try to select a zone for a scoped address that does not yet have a zone. Called from PCB bind and connect routines, for two reasons: 1) to save on this (relatively expensive) selection for every individual packet route operation and 2) to allow the application to obtain the selected zone from the PCB as is customary for e.g. getsockname/getpeername BSD socket calls.

Ideally, callers would always supply a properly zoned address, in which case this function would not be needed. It exists both for compatibility with the BSD socket API (which accepts zoneless destination addresses) and for backward compatibility with pre-scoping lwIP code.

It may be impossible to select a zone, e.g. if there are no netifs. In that case, the address's zone field will be left as is.

Parameters

dest	the IPv6 address for which to select and set a zone.
src	source IPv6 address (const); may be equal to dest.

2.3.3.5.6.40 ip6_addr_set_zone

`#define ip6_addr_set_zone( ip6addr, zone_idx)   ((ip6addr)->zone = (zone_idx))`

Set the zone field of an IPv6 address to a particular value.

2.3.3.5.6.41 ip6_addr_test_zone

`#define ip6_addr_test_zone( ip6addr, netif)   (ip6_addr_equals_zone((ip6addr), netif_get_`

Test whether an IPv6 address is "zone-compatible" with a network interface. That is, test whether the network interface is part of the zone associated with the address. For efficiency, this macro is only ever called if the given address is either scoped or zoned, and thus, it need not test this. If an address is scoped but not zoned, or zoned and not scoped, it is considered not zone-compatible with any netif.

This default implementation follows the default model of RFC 4007, where only interface-local and link-local scopes are defined, and the zone index of both of those scopes always equals the index of the network interface. As such, there is always only one matching netif for a specific zone index, but all call sites of this macro currently support multiple matching netifs as well (at no additional expense in the common case).

Parameters

ip6addr	the IPv6 address (const).
netif	the network interface (const).

Returns 1 if the address is scope-compatible with the netif, 0 if not.

2.3.3.5.6.42 ip6_addr_zone

`#define ip6_addr_zone( ip6addr)   ((ip6addr)->zone)`

Return the zone index of the given IPv6 address; possibly "no zone".

2.3.3.5.6.43 ip6_addr_zone_eq

```
#define ip6_addr_zone_eq( ip6addr1, ip6addr2)   ((ip6addr1)->zone == (ip6addr2)->zone)
```

Are the zone fields of the given IPv6 addresses equal? (0/1) This macro must only be used on IPv6 addresses of the same scope.

2.3.3.5.6.44 IP6_NO_ZONE

```
#define IP6_NO_ZONE   0
```

Identifier for "no zone".

2.3.3.5.6.45 IPADDR6_ZONE_INIT

```
#define IPADDR6_ZONE_INIT   , IP6_NO_ZONE
```

Zone initializer for static IPv6 address initialization, including comma.

2.3.3.5.6.46 IPV6_CUSTOM_SCOPES

```
#define IPV6_CUSTOM_SCOPES   0
```

IPV6_CUSTOM_SCOPES: together, the following three macro definitions, ip6_addr_has_scope, ip6_addr_assign_zone, and ip6_addr_test_zone, completely define the lwIP scoping policy. The definitions below implement the default policy from RFC 4007 Sec. 6. Should an implementation desire to implement a different policy, it can define IPV6_CUSTOM_SCOPES to 1 and supply its own definitions for the three macros instead.

2.3.3.5.6.47 Enumeration Type Documentation

2.3.3.5.6.48 lwip_ipv6_scope_type

```
enum lwip_ipv6_scope_type
```

Symbolic constants for the 'type' parameters in some of the macros. These exist for efficiency only, allowing the macros to avoid certain tests when the address is known not to be of a certain type. Dead code elimination will do the rest. IP6_MULTICAST is supported but currently not optimized. **See also** ip6_addr_has_scope, ip6_addr_assign_zone, ip6_addr_lacks_zone.

IP6_UNKNOWN	Unknown
IP6_UNICAST	Unicast
IP6_MULTICAST	Multicast

2.3.3.6 Network interface (NETIF)

2.3.3.6.1 Modules

- IPv4 address handling
- IPv6 address handling
- Client data handling
- Flags
- MIB2 statistics

2.3.3.6.2 Data Structures

- union netif_ext_callback_args_t

2.3.3.6.3 Macros

- #define netif_is_up(netif) (((netif)->flags & NETIF_FLAG_UP) ? (u8_t)1 : (u8_t)0)
- #define netif_set_igmp_mac_filter(netif, function) do { if((netif) != NULL) { (netif)->igmp_mac_filter = function; }}while(0)
- #define netif_set_mld_mac_filter(netif, function) do { if((netif) != NULL) { (netif)->mld_mac_filter = function; }}while(0)

2.3.3.6.4 Typedefs

- typedef u16_t netif_nsc_reason_t
- typedef void(* netif_ext_callback_fn) (struct netif *netif, netif_nsc_reason_t reason, const netif_ext_callback_args_t *args)

2.3.3.6.5 Functions

- struct netif * netif_add_noaddr (struct netif *netif, void *state, netif_init_fn init, netif_input_fn input)
- struct netif * netif_add (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw, void *state, netif_init_fn init, netif_input_fn input)
- void netif_remove (struct netif *netif)
- void netif_set_default (struct netif *netif)
- void netif_set_up (struct netif *netif)
- void netif_set_down (struct netif *netif)
- void netif_set_status_callback (struct netif *netif, netif_status_callback_fn status_callback)
- void netif_set_remove_callback (struct netif *netif, netif_status_callback_fn remove_callback)
- void netif_set_link_up (struct netif *netif)
- void netif_set_link_down (struct netif *netif)
- void netif_set_link_callback (struct netif *netif, netif_status_callback_fn link_callback)
- err_t netif_loop_output (struct netif *netif, struct pbuf *p)
- u8_t netif_name_to_index (const char *name)
- char * netif_index_to_name (u8_t idx, char *name)
- struct netif * netif_get_by_index (u8_t idx)
- struct netif * netif_find (const char *name)
- void netif_add_ext_callback (netif_ext_callback_t *callback, netif_ext_callback_fn fn)
- void netif_remove_ext_callback (netif_ext_callback_t *callback)

2.3.3.6.6 Detailed Description

2.3.3.6.7 Macro Definition Documentation

2.3.3.6.7.1 netif_is_up

```
#define netif_is_up( netif)   (((netif)->flags & NETIF_FLAG_UP) ? (u8_t)1 :  (u8_t)0)
```

Ask if an interface is up

2.3.3.6.7.2 netif_set_igmp_mac_filter

```
#define netif_set_igmp_mac_filter( netif, function)   do { if((netif) != NULL) { (netif)->
= function; }}while(0)
```

Set igmp mac filter function for a netif.

2.3.3.6.7.3 netif_set_mld_mac_filter

```
#define netif_set_mld_mac_filter( netif, function)   do { if((netif) != NULL) { (netif)->m
= function; }}while(0)
```

Set mld mac filter function for a netif.

2.3.3.6.8 Typedef Documentation

2.3.3.6.8.1 netif_ext_callback_fn

```
typedef void(* netif_ext_callback_fn) (struct netif *netif, netif_nsc_reason_t reason,
const netif_ext_callback_args_t *args)
```

Function used for extended netif status callbacks Note: When parsing reason argument, keep in mind that more reasons may be added in the future!

Parameters

netif	netif that is affected by change
reason	change reason
args	depends on reason, see reason description

2.3.3.6.8.2 netif_nsc_reason_t

```
typedef u16_t netif_nsc_reason_t
```

Extended netif status callback (NSC) reasons flags. May be extended in the future!

2.3.3.6.9 Function Documentation

2.3.3.6.9.1 netif_add()

```
struct netif * netif_add (struct netif * netif, const ip4_addr_t * ipaddr, const ip4_addr_
* netmask, const ip4_addr_t * gw, void * state, netif_init_fn init, netif_input_fn input)
```

Add a network interface to the list of lwIP netifs.

Parameters

Returns netif, or NULL if failed.

2.3.3.6.9.2 netif_add_ext_callback()

```
void netif_add_ext_callback (netif_ext_callback_t * callback, netif_ext_callback_fn fn)
```

Add extended netif events listener

Parameters

netif	a pre-allocated netif structure
ipaddr	IP address for the new netif
netmask	network mask for the new netif
gw	default gateway IP address for the new netif
state	opaque data passed to the new netif
init	callback function that initializes the interface
input	callback function that is called to pass ingress packets up in the protocol layer stack. It is recommended to use a function that passes the input directly to the stack (netif_input(), NO_SYS=1 mode) or via sending a message to TCPIP thread (tcpip_input(), NO_SYS=0 mode). These functions use netif flags NETIF_FLAG_ETHARP and NETIF_FLAG_ETHERNET to decide whether to forward to ethernet_input() or ip_input(). In other words, the functions only work when the netif driver is implemented correctly! Most members of struct netif should be be initialized by the netif init function = netif driver (init parameter of this function). IPv6: Don't forget to call netif_create_ip6_linklocal_address() after setting the MAC address in struct netif.hwaddr (IPv6 requires a link-local address).

callback	pointer to listener structure
fn	callback function

2.3.3.6.9.3 netif_add_noaddr()

```
struct netif * netif_add_noaddr (struct netif * netif, void * state, netif_init_fn init,
netif_input_fn input)
```

Add a network interface to the list of lwIP netifs.

Same as netif_add but without IPv4 addresses

2.3.3.6.9.4 netif_find()

```
struct netif * netif_find (const char * name)
```

Find a network interface by searching for its name

Parameters

name	the name of the netif (like netif->name) plus concatenated number in ascii representation (e.g. 'en0')

2.3.3.6.9.5 netif_get_by_index()

```
struct netif * netif_get_by_index (u8_t idx)
```

Return the interface for the netif index

Parameters

2.3.3.6.9.6 netif_index_to_name()

```
char * netif_index_to_name (u8_t idx, char * name)
```

Return the interface name for the netif matching index or NULL if not found/on error

Parameters

idx	index of netif to find

idx	the interface index of the netif
name	char buffer of at least NETIF_NAMESIZE bytes

2.3.3.6.9.7 netif_loop_output()

```
err_t netif_loop_output (struct netif * netif, struct pbuf * p)
```

Send an IP packet to be received on the same netif (loopif-like). The pbuf is copied and added to an internal queue which is fed to netif->input by netif_poll(). In multithreaded mode, the call to netif_poll() is queued to be done on the TCP/IP thread. In callback mode, the user has the responsibility to call netif_poll() in the main loop of their application.

Parameters

netif	the lwip network interface structure
p	the (IP) packet to 'send'

Returns ERR_OK if the packet has been sent ERR_MEM if the pbuf used to copy the packet couldn't be allocated

2.3.3.6.9.8 netif_name_to_index()

```
u8_t netif_name_to_index (const char * name)
```

Return the interface index for the netif with name or NETIF_NO_INDEX if not found/on error

Parameters

2.3.3.6.9.9 netif_remove()

```
void netif_remove (struct netif * netif)
```

Remove a network interface from the list of lwIP netifs.

Parameters

2.3.3.6.9.10 netif_remove_ext_callback()

```
void netif_remove_ext_callback (netif_ext_callback_t * callback)
```

Remove extended netif events listener

Parameters

2.3.3.6.9.11 netif_set_default()

```
void netif_set_default (struct netif * netif)
```

Set a network interface as the default network interface (used to output all packets for which no specific route is found)

Parameters

2.3.3.6.9.12 netif_set_down()

```
void netif_set_down (struct netif * netif)
```

Bring an interface down, disabling any traffic processing.

name	the name of the netif

netif	the network interface to remove

2.3.3.6.9.13 netif_set_link_callback()

```
void netif_set_link_callback (struct netif * netif, netif_status_callback_fn link_callback
```

Set callback to be called when link is brought up/down

2.3.3.6.9.14 netif_set_link_down()

```
void netif_set_link_down (struct netif * netif)
```

Called by a driver when its link goes down

2.3.3.6.9.15 netif_set_link_up()

```
void netif_set_link_up (struct netif * netif)
```

Called by a driver when its link goes up

2.3.3.6.9.16 netif_set_remove_callback()

```
void netif_set_remove_callback (struct netif * netif, netif_status_callback_fn remove_call
```

Set callback to be called when the interface has been removed

2.3.3.6.9.17 netif_set_status_callback()

```
void netif_set_status_callback (struct netif * netif, netif_status_callback_fn status_call
```

Set callback to be called when interface is brought up/down or address is changed while up

2.3.3.6.9.18 netif_set_up()

```
void netif_set_up (struct netif * netif)
```

Bring an interface up, available for processing traffic.

2.3.3.6.10 IPv4 address handling

2.3.3.6.10.1 Functions

- void netif_set_ipaddr (struct netif *netif, const ip4_addr_t *ipaddr)
- void netif_set_netmask (struct netif *netif, const ip4_addr_t *netmask)
- void netif_set_gw (struct netif *netif, const ip4_addr_t *gw)
- void netif_set_addr (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw)

callback	pointer to listener structure

netif	the default network interface

2.3.3.6.10.2 Detailed Description

2.3.3.6.10.3 Function Documentation

2.3.3.6.10.4 netif_set_addr()

```
void netif_set_addr (struct netif * netif, const ip4_addr_t * ipaddr, const ip4_addr_t
* netmask, const ip4_addr_t * gw)
```

Change IP address configuration for a network interface (including netmask and default gateway).

Parameters

netif	the network interface to change
ipaddr	the new IP address
netmask	the new netmask
gw	the new default gateway

2.3.3.6.10.5 netif_set_gw()

```
void netif_set_gw (struct netif * netif, const ip4_addr_t * gw)
```

Change the default gateway for a network interface

Parameters

netif	the network interface to change
gw	the new default gateway

Note
call netif_set_addr() if you also want to change ip address and netmask

2.3.3.6.10.6 netif_set_ipaddr()

```
void netif_set_ipaddr (struct netif * netif, const ip4_addr_t * ipaddr)
```

Change the IP address of a network interface

Parameters

Note
call netif_set_addr() if you also want to change netmask and default gateway

netif	the network interface to change
ipaddr	the new IP address

netif	the network interface to change
netmask	the new netmask

2.3.3.6.10.7 netif_set_netmask()

```
void netif_set_netmask (struct netif * netif, const ip4_addr_t * netmask)
```

Change the netmask of a network interface

Parameters

Note
call netif_set_addr() if you also want to change ip address and default gateway

2.3.3.6.11 IPv6 address handling

2.3.3.6.11.1 Functions

- void netif_ip6_addr_set (struct netif *netif, s8_t addr_idx, const ip6_addr_t *addr6)
- void netif_ip6_addr_set_state (struct netif *netif, s8_t addr_idx, u8_t state)
- void netif_create_ip6_linklocal_address (struct netif *netif, u8_t from_mac_48bit)
- err_t netif_add_ip6_address (struct netif *netif, const ip6_addr_t *ip6addr, s8_t *chosen_idx)

2.3.3.6.11.2 Detailed Description

2.3.3.6.11.3 Function Documentation

2.3.3.6.11.4 netif_add_ip6_address()

```
err_t netif_add_ip6_address (struct netif * netif, const ip6_addr_t * ip6addr, s8_t * chos
```

This function allows for the easy addition of a new IPv6 address to an interface. It takes care of finding an empty slot and then sets the address tentative (to make sure that all the subsequent processing happens).

Parameters

netif	netif to add the address on
ip6addr	address to add
chosen_idx	if != NULL, the chosen IPv6 address index will be stored here

2.3.3.6.11.5 netif_create_ip6_linklocal_address()

```
void netif_create_ip6_linklocal_address (struct netif * netif, u8_t from_mac_48bit)
```

Create a link-local IPv6 address on a netif (stored in slot 0)

Parameters

netif	the netif to create the address on
from_mac_48bit	if != 0, assume hwadr is a 48-bit MAC address (std conversion) if == 0, use hwaddr directly as interface ID

2.3.3.6.11.6 netif_ip6_addr_set()

```
void netif_ip6_addr_set (struct netif * netif, s8_t addr_idx, const ip6_addr_t * addr6)
```

Change an IPv6 address of a network interface

Parameters

netif	the network interface to change
addr_idx	index of the IPv6 address
addr6	the new IPv6 address

Note
call netif_ip6_addr_set_state() to set the address valid/temptative

2.3.3.6.11.7 netif_ip6_addr_set_state()

```
void netif_ip6_addr_set_state (struct netif * netif, s8_t addr_idx, u8_t state)
```

Change the state of an IPv6 address of a network interface (INVALID, TEMPTATIVE, PREFERRED, DEPRECATED, where TEMPTATIVE includes the number of checks done, see ip6_addr.h)

Parameters

netif	the network interface to change
addr_idx	index of the IPv6 address
state	the new IPv6 address state

2.3.3.6.12 Client data handling

2.3.3.6.12.1 Macros

- #define netif_set_client_data(netif, id, data)   netif_get_client_data(netif, id) = (data)
- #define netif_get_client_data(netif, id)   (netif)->client_data[(id)]

2.3.3.6.12.2 Functions

- u8_t netif_alloc_client_data_id (void)

2.3.3.6.12.3 Detailed Description

Store data (void*) on a netif for application usage. **See also** LWIP_NUM_NETIF_CLIENT_DATA

2.3.3.6.12.4 Macro Definition Documentation

2.3.3.6.12.5 netif_get_client_data

```
#define netif_get_client_data( netif, id)   (netif)->client_data[(id)]
```

Get client data. Obtain ID from netif_alloc_client_data_id().

2.3.3.6.12.6 netif_set_client_data

```
#define netif_set_client_data( netif, id, data)   netif_get_client_data(netif, id) = (data
```

Set client data. Obtain ID from netif_alloc_client_data_id().

2.3.3.6.12.7 Function Documentation

2.3.3.6.12.8 netif_alloc_client_data_id()

```
u8_t netif_alloc_client_data_id (void )
```

Allocate an index to store data in client_data member of struct netif. Returned value is an index in mentioned array. **See also** LWIP_NUM_NETIF_CLIENT_DATA

2.3.3.6.13 Flags

2.3.3.6.13.1 Macros

- #define NETIF_FLAG_UP 0x01U
- #define NETIF_FLAG_BROADCAST 0x02U
- #define NETIF_FLAG_LINK_UP 0x04U
- #define NETIF_FLAG_ETHARP 0x08U
- #define NETIF_FLAG_ETHERNET 0x10U
- #define NETIF_FLAG_IGMP 0x20U
- #define NETIF_FLAG_MLD6 0x40U

2.3.3.6.13.2 Detailed Description

2.3.3.6.13.3 Macro Definition Documentation

2.3.3.6.13.4 NETIF_FLAG_BROADCAST

```
#define NETIF_FLAG_BROADCAST   0x02U
```

If set, the netif has broadcast capability. Set by the netif driver in its init function.

2.3.3.6.13.5 NETIF_FLAG_ETHARP

```
#define NETIF_FLAG_ETHARP   0x08U
```

If set, the netif is an ethernet device using ARP. Set by the netif driver in its init function. Used to check input packet types and use of DHCP.

2.3.3.6.13.6 NETIF_FLAG_ETHERNET

```
#define NETIF_FLAG_ETHERNET   0x10U
```

If set, the netif is an ethernet device. It might not use ARP or TCP/IP if it is used for PPPoE only.

2.3.3.6.13.7 NETIF_FLAG_IGMP

```
#define NETIF_FLAG_IGMP   0x20U
```

If set, the netif has IGMP capability. Set by the netif driver in its init function.

2.3.3.6.13.8 NETIF_FLAG_LINK_UP

```
#define NETIF_FLAG_LINK_UP   0x04U
```

If set, the interface has an active link (set by the network interface driver). Either set by the netif driver in its init function (if the link is up at that time) or at a later point once the link comes up (if link detection is supported by the hardware).

2.3.3.6.13.9 NETIF_FLAG_MLD6

```
#define NETIF_FLAG_MLD6   0x40U
```

If set, the netif has MLD6 capability. Set by the netif driver in its init function.

2.3.3.6.13.10 NETIF_FLAG_UP

```
#define NETIF_FLAG_UP   0x01U
```

Whether the network interface is 'up'. This is a software flag used to control whether this network interface is enabled and processes traffic. It must be set by the startup code before this netif can be used (also for dhcp/autoip).

2.3.3.6.14 MIB2 statistics

2.3.3.6.14.1 Data Structures

- struct stats_mib2_netif_ctrs

2.3.3.6.14.2 Macros

- #define MIB2_STATS_NETIF_INC(n, x)   do { ++(n)->mib2_counters.x; } while(0)
- #define MIB2_STATS_NETIF_ADD(n, x, val)   do { (n)->mib2_counters.x += (val); } while(0)
- #define MIB2_INIT_NETIF(netif, type, speed)

2.3.3.6.14.3 Enumerations

- enum snmp_ifType

2.3.3.6.14.4 Detailed Description

2.3.3.6.14.5 Macro Definition Documentation

2.3.3.6.14.6 MIB2_INIT_NETIF

`#define MIB2_INIT_NETIF( netif, type, speed)` **Value:**

```
do { \
(netif)->link_type = (type);  \
(netif)->link_speed = (speed);\
(netif)->ts = 0;              \
(netif)->mib2_counters.ifinoctets = 0;      \
(netif)->mib2_counters.ifinucastpkts = 0;   \
(netif)->mib2_counters.ifinnucastpkts = 0;  \
(netif)->mib2_counters.ifindiscards = 0;    \
(netif)->mib2_counters.ifinerrors = 0;    \
(netif)->mib2_counters.ifinunknownprotos = 0;    \
(netif)->mib2_counters.ifoutoctets = 0;     \
(netif)->mib2_counters.ifoutucastpkts = 0;  \
(netif)->mib2_counters.ifoutnucastpkts = 0; \
(netif)->mib2_counters.ifoutdiscards = 0; \
(netif)->mib2_counters.ifouterrors = 0; } while(0)
```

Init MIB2 statistic counters in netif

Parameters

netif	Netif to init
type	one of enum snmp_ifType
speed	your link speed here (units: bits per second)

2.3.3.6.14.7 MIB2_STATS_NETIF_ADD

`#define MIB2_STATS_NETIF_ADD( n, x, val)   do { (n)->mib2_counters.x += (val); } while(0)`

Add value to stats member for SNMP MIB2 stats (struct stats_mib2_netif_ctrs)

2.3.3.6.14.8 MIB2_STATS_NETIF_INC

`#define MIB2_STATS_NETIF_INC( n, x)   do { ++(n)->mib2_counters.x; } while(0)`

Increment stats member for SNMP MIB2 stats (struct stats_mib2_netif_ctrs)

2.3.3.6.14.9 Enumeration Type Documentation

2.3.3.6.14.10 snmp_ifType

`enum snmp_ifType`

See also RFC1213, "MIB-II, 6. Definitions"

2.3.3.7 RAW

2.3.3.7.1 Functions

- err_t raw_bind (struct raw_pcb *pcb, const ip_addr_t *ipaddr)

- void raw_bind_netif (struct raw_pcb *pcb, const struct netif *netif)
- err_t raw_connect (struct raw_pcb *pcb, const ip_addr_t *ipaddr)
- void raw_disconnect (struct raw_pcb *pcb)
- void raw_recv (struct raw_pcb *pcb, raw_recv_fn recv, void *recv_arg)
- err_t raw_sendto (struct raw_pcb *pcb, struct pbuf *p, const ip_addr_t *ipaddr)
- err_t raw_sendto_if_src (struct raw_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, struct netif *netif, const ip_addr_t *src_ip)
- err_t raw_send (struct raw_pcb *pcb, struct pbuf *p)
- void raw_remove (struct raw_pcb *pcb)
- struct raw_pcb * raw_new (u8_t proto)
- struct raw_pcb * raw_new_ip_type (u8_t type, u8_t proto)

2.3.3.7.2 Detailed Description

Implementation of raw protocol PCBs for low-level handling of different types of protocols besides (or overriding) those already available in lwIP. **See also** APIs

2.3.3.7.3 Function Documentation

2.3.3.7.3.1 raw_bind()

```
err_t raw_bind (struct raw_pcb * pcb, const ip_addr_t * ipaddr)
```

Bind a RAW PCB.

Parameters

pcb	RAW PCB to be bound with a local address ipaddr.
ipaddr	local IP address to bind with. Use IP4_ADDR_ANY to bind to all local interfaces.

Returns lwIP error code.

- ERR_OK. Successful. No error occurred.
- ERR_USE. The specified IP address is already bound to by another RAW PCB.

See also raw_disconnect()

2.3.3.7.3.2 raw_bind_netif()

```
void raw_bind_netif (struct raw_pcb * pcb, const struct netif * netif)
```

Bind an RAW PCB to a specific netif. After calling this function, all packets received via this PCB are guaranteed to have come in via the specified netif, and all outgoing packets will go out via the specified netif.

Parameters

pcb	RAW PCB to be bound with netif.
netif	netif to bind to. Can be NULL.

See also raw_disconnect()

2.3.3.7.3.3 raw_connect()

`err_t raw_connect (struct raw_pcb * pcb, const ip_addr_t * ipaddr)`

Connect an RAW PCB. This function is required by upper layers of lwip. Using the raw api you could use raw_sendto() instead

This will associate the RAW PCB with the remote address.

Parameters

pcb	RAW PCB to be connected with remote address ipaddr and port.
ipaddr	remote IP address to connect with.

Returns lwIP error code

See also raw_disconnect() and raw_sendto()

2.3.3.7.3.4 raw_disconnect()

`void raw_disconnect (struct raw_pcb * pcb)`

Disconnect a RAW PCB.

Parameters

pcb	the raw pcb to disconnect.

2.3.3.7.3.5 raw_new()

`struct raw_pcb * raw_new (u8_t proto)`

Create a RAW PCB.

Returns The RAW PCB which was created. NULL if the PCB data structure could not be allocated.

Parameters

proto	the protocol number of the IPs payload (e.g. IP_PROTO_ICMP)

See also raw_remove()

2.3.3.7.3.6 raw_new_ip_type()

`struct raw_pcb * raw_new_ip_type (u8_t type, u8_t proto)`

Create a RAW PCB for specific IP type.

Returns The RAW PCB which was created. NULL if the PCB data structure could not be allocated.

Parameters

See also raw_remove()

type	IP address type, see lwip_ip_addr_type definitions. If you want to listen to IPv4 and IPv6 (dual-stack) packets, supply IPADDR_TYPE_ANY as argument and bind to IP_ANY_TYPE.
proto	the protocol number (next header) of the IPv6 packet payload (e.g. IP6_NEXTH_ICMP6)

2.3.3.7.3.7 raw_recv()

```
void raw_recv (struct raw_pcb * pcb, raw_recv_fn recv, void * recv_arg)
```

Set the callback function for received packets that match the raw PCB's protocol and binding.

The callback function MUST either

- eat the packet by calling pbuf_free() and returning non-zero. The packet will not be passed to other raw PCBs or other protocol layers.
- not free the packet, and return zero. The packet will be matched against further PCBs and/or forwarded to another protocol layers.

2.3.3.7.3.8 raw_remove()

```
void raw_remove (struct raw_pcb * pcb)
```

Remove an RAW PCB.

Parameters

pcb	RAW PCB to be removed. The PCB is removed from the list of RAW PCB's and the data structure is freed from memory.

See also raw_new()

2.3.3.7.3.9 raw_send()

```
err_t raw_send (struct raw_pcb * pcb, struct pbuf * p)
```

Send the raw IP packet to the address given by raw_connect()

Parameters

pcb	the raw pcb which to send
p	the IP payload to send

2.3.3.7.3.10 raw_sendto()

```
err_t raw_sendto (struct raw_pcb * pcb, struct pbuf * p, const ip_addr_t * ipaddr)
```

Send the raw IP packet to the given address. An IP header will be prepended to the packet, unless the RAW_FLAGS_HDRINCL flag is set on the PCB. In that case, the packet must include an IP header, which will then be sent as is.

Parameters

pcb	the raw pcb which to send
p	the IP payload to send
ipaddr	the destination address of the IP packet

2.3.3.7.3.11 raw_sendto_if_src()

```
err_t raw_sendto_if_src (struct raw_pcb * pcb, struct pbuf * p, const ip_addr_t * dst_ip,
struct netif * netif, const ip_addr_t * src_ip)
```

Send the raw IP packet to the given address, using a particular outgoing netif and source IP address. An IP header will be prepended to the packet, unless the RAW_FLAGS_HDRINCL flag is set on the PCB. In that case, the packet must include an IP header, which will then be sent as is.

Parameters

pcb	RAW PCB used to send the data
p	chain of pbufs to be sent
dst_ip	destination IP address
netif	the netif used for sending
src_ip	source IP address

2.3.3.8 TCP

2.3.3.8.1 Modules

- ext arguments

2.3.3.8.2 Functions

- void tcp_backlog_delayed (struct tcp_pcb *pcb)
- void tcp_backlog_accepted (struct tcp_pcb *pcb)
- err_t tcp_close (struct tcp_pcb *pcb)
- err_t tcp_shutdown (struct tcp_pcb *pcb, int shut_rx, int shut_tx)
- void tcp_abort (struct tcp_pcb *pcb)
- err_t tcp_bind (struct tcp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port)
- void tcp_bind_netif (struct tcp_pcb *pcb, const struct netif *netif)
- struct tcp_pcb * tcp_listen_with_backlog (struct tcp_pcb *pcb, u8_t backlog)
- struct tcp_pcb * tcp_listen_with_backlog_and_err (struct tcp_pcb *pcb, u8_t backlog, err_t *err)
- void tcp_recved (struct tcp_pcb *pcb, u16_t len)
- err_t tcp_connect (struct tcp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port, tcp_connected_fn connected)
- struct tcp_pcb * tcp_new (void)
- struct tcp_pcb * tcp_new_ip_type (u8_t type)
- void tcp_arg (struct tcp_pcb *pcb, void *arg)
- void tcp_recv (struct tcp_pcb *pcb, tcp_recv_fn recv)

- void tcp_sent (struct tcp_pcb *pcb, tcp_sent_fn sent)
- void tcp_err (struct tcp_pcb *pcb, tcp_err_fn err)
- void tcp_accept (struct tcp_pcb *pcb, tcp_accept_fn accept)
- void tcp_poll (struct tcp_pcb *pcb, tcp_poll_fn poll, u8_t interval)
- err_t tcp_write (struct tcp_pcb *pcb, const void *arg, u16_t len, u8_t apiflags)
- err_t tcp_output (struct tcp_pcb *pcb)

2.3.3.8.3 Detailed Description

Transmission Control Protocol for IP **See also** APIs

Common functions for the TCP implementation, such as functions for manipulating the data structures and the TCP timer functions. TCP functions related to input and output is found in tcp_in.c and tcp_out.c respectively.

TCP connection setup

The functions used for setting up connections is similar to that of the sequential API and of the BSD socket API. A new TCP connection identifier (i.e., a protocol control block - PCB) is created with the tcp_new() function. This PCB can then be either set to listen for new incoming connections or be explicitly connected to another host.

- tcp_new()
- tcp_bind()
- tcp_listen() and tcp_listen_with_backlog()
- tcp_accept()
- tcp_connect()

Sending TCP data

TCP data is sent by enqueueing the data with a call to tcp_write() and triggering to send by calling tcp_output(). When the data is successfully transmitted to the remote host, the application will be notified with a call to a specified callback function.

- tcp_write()
- tcp_output()
- tcp_sent()

Receiving TCP data

TCP data reception is callback based - an application specified callback function is called when new data arrives. When the application has taken the data, it has to call the tcp_recved() function to indicate that TCP can advertise increase the receive window.

- tcp_recv()
- tcp_recved()

Application polling

When a connection is idle (i.e., no data is either transmitted or received), lwIP will repeatedly poll the application by calling a specified callback function. This can be used either as a watchdog timer for killing connections that have stayed idle for too long, or as a method of waiting for memory to become available. For instance, if a call to tcp_write() has failed because memory wasn't available, the application may use the polling functionality to call tcp_write() again when the connection has been idle for a while.

- tcp_poll()

Closing and aborting connections

- tcp_close()
- tcp_abort()
- tcp_err()

2.3.3.8.4 Function Documentation

2.3.3.8.4.1 tcp_abort()

```
void tcp_abort (struct tcp_pcb * pcb)
```

Aborts the connection by sending a RST (reset) segment to the remote host. The pcb is deallocated. This function never fails.

ATTENTION: When calling this from one of the TCP callbacks, make sure you always return ERR_ABRT (and never return ERR_ABRT otherwise or you will risk accessing deallocated memory or memory leaks!

Parameters

pcb	the tcp pcb to abort

2.3.3.8.4.2 tcp_accept()

```
void tcp_accept (struct tcp_pcb * pcb, tcp_accept_fn accept)
```

Used for specifying the function that should be called when a LISTENing connection has been connected to another host. **See also** MEMP_NUM_TCP_PCB_LISTEN and MEMP_NUM_TCP_PCB

Parameters

pcb	tcp_pcb to set the accept callback
accept	callback function to call for this pcb when LISTENing connection has been connected to another host

2.3.3.8.4.3 tcp_arg()

```
void tcp_arg (struct tcp_pcb * pcb, void * arg)
```

Specifies the program specific state that should be passed to all other callback functions. The "pcb" argument is the current TCP connection control block, and the "arg" argument is the argument that will be passed to the callbacks.

Parameters

pcb	tcp_pcb to set the callback argument
arg	void pointer argument to pass to callback functions

pcb	the connection pcb which is now fully accepted (or closed/aborted)

2.3.3.8.4.4 tcp_backlog_accepted()

```
void tcp_backlog_accepted (struct tcp_pcb * pcb)
```

A delayed-accept a connection is accepted (or closed/aborted): decreases the number of outstanding connections after calling tcp_backlog_delayed().

ATTENTION: the caller is responsible for calling tcp_backlog_accepted() or else the backlog feature will get out of sync!

Parameters

2.3.3.8.4.5 tcp_backlog_delayed()

```
void tcp_backlog_delayed (struct tcp_pcb * pcb)
```

Delay accepting a connection in respect to the listen backlog: the number of outstanding connections is increased until tcp_backlog_accep
is called.

ATTENTION: the caller is responsible for calling tcp_backlog_accepted() or else the backlog feature will get out of sync!

Parameters

pcb	the connection pcb which is not fully accepted yet

2.3.3.8.4.6 tcp_bind()

```
err_t tcp_bind (struct tcp_pcb * pcb, const ip_addr_t * ipaddr, u16_t port)
```

Binds the connection to a local port number and IP address. If the IP address is not given (i.e., ipaddr == IP_ANY_TYPE), the connection is bound to all local IP addresses. If another connection is bound to the same port, the function will return ERR_USE, otherwise ERR_OK is returned. **See also** MEMP_NUM_TCP_PCB_LISTEN and MEMP_NUM_TCP_PCB

Parameters

pcb	the tcp_pcb to bind (no check is done whether this pcb is already bound!)
ipaddr	the local ip address to bind to (use IPx_ADDR_ANY to bind to any local address
port	the local port to bind to

Returns ERR_USE if the port is already in use ERR_VAL if bind failed because the PCB is not in a valid state ERR_OK if bound

2.3.3.8.4.7 tcp_bind_netif()

```
void tcp_bind_netif (struct tcp_pcb * pcb, const struct netif * netif)
```

Binds the connection to a netif and IP address. After calling this function, all packets received via this PCB are guaranteed to have come in via the specified netif, and all outgoing packets will go out via the specified netif.

Parameters

pcb	the tcp_pcb to bind.
netif	the netif to bind to. Can be NULL.

2.3.3.8.4.8 tcp_close()

```
err_t tcp_close (struct tcp_pcb * pcb)
```

Closes the connection held by the PCB.

Listening pcbs are freed and may not be referenced any more. Connection pcbs are freed if not yet connected and may not be referenced any more. If a connection is established (at least SYN received or in a closing state), the connection is closed, and put in a closing state. The pcb is then automatically freed in tcp_slowtmr(). It is therefore unsafe to reference it (unless an error is returned).

The function may return ERR_MEM if no memory was available for closing the connection. If so, the application should wait and try again either by using the acknowledgment callback or the polling functionality. If the close succeeds, the function returns ERR_OK.

Parameters

pcb	the tcp_pcb to close

Returns ERR_OK if connection has been closed another err_t if closing failed and pcb is not freed

2.3.3.8.4.9 tcp_connect()

```
err_t tcp_connect (struct tcp_pcb * pcb, const ip_addr_t * ipaddr, u16_t port, tcp_connect
connected)
```

Connects to another host. The function given as the "connected" argument will be called when the connection has been established. Sets up the pcb to connect to the remote host and sends the initial SYN segment which opens the connection.

The tcp_connect() function returns immediately; it does not wait for the connection to be properly setup. Instead, it will call the function specified as the fourth argument (the "connected" argument) when the connection is established. If the connection could not be properly established, either because the other host refused the connection or because the other host didn't answer, the "err" callback function of this pcb (registered with tcp_err, see below) will be called.

The tcp_connect() function can return ERR_MEM if no memory is available for enqueueing the SYN segment. If the SYN indeed was enqueued successfully, the tcp_connect() function returns ERR_OK.

Parameters

pcb	the tcp_pcb used to establish the connection
ipaddr	the remote ip address to connect to
port	the remote tcp port to connect to
connected	callback function to call when connected (on error, the err callback will be called)

Returns ERR_VAL if invalid arguments are given ERR_OK if connect request has been sent other err_t values if connect request couldn't be sent

2.3.3.8.4.10 tcp_err()

```
void tcp_err (struct tcp_pcb * pcb, tcp_err_fn err)
```

Used to specify the function that should be called when a fatal error has occurred on the connection.

If a connection is aborted because of an error, the application is alerted of this event by the err callback. Errors that might abort a connection are when there is a shortage of memory. The callback function to be called is set using the tcp_err() function.

Note
The corresponding pcb is already freed when this callback is called!

Parameters

pcb	tcp_pcb to set the err callback
err	callback function to call for this pcb when a fatal error has occurred on the connection

2.3.3.8.4.11 tcp_listen_with_backlog()

`struct tcp_pcb * tcp_listen_with_backlog (struct tcp_pcb * pcb, u8_t backlog)`

Set the state of the connection to be LISTEN, which means that it is able to accept incoming connections. The protocol control block is reallocated in order to consume less memory. Setting the connection to LISTEN is an irreversible process. When an incoming connection is accepted, the function specified with the tcp_accept() function will be called. The pcb has to be bound to a local port with the tcp_bind() function.

The tcp_listen() function returns a new connection identifier, and the one passed as an argument to the function will be deallocated. The reason for this behavior is that less memory is needed for a connection that is listening, so tcp_listen() will reclaim the memory needed for the original connection and allocate a new smaller memory block for the listening connection.

tcp_listen() may return NULL if no memory was available for the listening connection. If so, the memory associated with the pcb passed as an argument to tcp_listen() will not be deallocated.

The backlog limits the number of outstanding connections in the listen queue to the value specified by the backlog argument. To use it, your need to set TCP_LISTEN_BACKLOG=1 in your lwipopts.h.

Parameters

pcb	the original tcp_pcb
backlog	the incoming connections queue limit

Returns tcp_pcb used for listening, consumes less memory.

Note
The original tcp_pcb is freed. This function therefore has to be called like this: tpcb = tcp_listen_with_backlog(tpcb, backlog);

2.3.3.8.4.12 tcp_listen_with_backlog_and_err()

`struct tcp_pcb * tcp_listen_with_backlog_and_err (struct tcp_pcb * pcb, u8_t backlog, err_`
`* err)`

Set the state of the connection to be LISTEN, which means that it is able to accept incoming connections. The protocol control block is reallocated in order to consume less memory. Setting the connection to LISTEN is an irreversible process.

Parameters

pcb	the original tcp_pcb
backlog	the incoming connections queue limit
err	when NULL is returned, this contains the error reason

Returns tcp_pcb used for listening, consumes less memory.

Note
The original tcp_pcb is freed. This function therefore has to be called like this: tpcb = tcp_listen_with_backlog_and_err(tpcb, backlog, &err);

2.3.3.8.4.13 tcp_new()

```
struct tcp_pcb * tcp_new (void )
```

Creates a new TCP protocol control block but doesn't place it on any of the TCP PCB lists. The pcb is not put on any list until binding using tcp_bind(). If memory is not available for creating the new pcb, NULL is returned. **See also** MEMP_NUM_TCP_PCB_LIS and MEMP_NUM_TCP_PCB

2.3.3.8.4.14 tcp_new_ip_type()

```
struct tcp_pcb * tcp_new_ip_type (u8_t type)
```

Creates a new TCP protocol control block but doesn't place it on any of the TCP PCB lists. The pcb is not put on any list until binding using tcp_bind(). **See also** MEMP_NUM_TCP_PCB_LISTEN and MEMP_NUM_TCP_PCB

Parameters

type	IP address type, see lwip_ip_addr_type definitions. If you want to listen to IPv4 and IPv6 (dual-stack) connections, supply IPADDR_TYPE_ANY as argument and bind to IP_ANY_TYPE.

Returns a new tcp_pcb that initially is in state CLOSED

2.3.3.8.4.15 tcp_output()

```
err_t tcp_output (struct tcp_pcb * pcb)
```

Find out what we can send and send it

Parameters

pcb	Protocol control block for the TCP connection to send data

Returns ERR_OK if data has been sent or nothing to send another err_t on error

2.3.3.8.4.16 tcp_poll()

```
void tcp_poll (struct tcp_pcb * pcb, tcp_poll_fn poll, u8_t interval)
```

Specifies the polling interval and the callback function that should be called to poll the application. The interval is specified in number of TCP coarse grained timer shots, which typically occurs twice a second. An interval of 10 means that the application would be polled every 5 seconds.

When a connection is idle (i.e., no data is either transmitted or received), lwIP will repeatedly poll the application by calling a specified callback function. This can be used either as a watchdog timer for killing connections that have stayed idle for too long, or as a method of waiting for memory to become available. For instance, if a call to tcp_write() has failed because memory wasn't available, the application may use the polling functionality to call tcp_write() again when the connection has been idle for a while.

2.3.3.8.4.17 tcp_recv()

```
void tcp_recv (struct tcp_pcb * pcb, tcp_recv_fn recv)
```

Sets the callback function that will be called when new data arrives. The callback function will be passed a NULL pbuf to indicate that the remote host has closed the connection. If the callback function returns ERR_OK or ERR_ABRT it must have freed the pbuf, otherwise it must not have freed it.

Parameters

pcb	tcp_pcb to set the recv callback
recv	callback function to call for this pcb when data is received

2.3.3.8.4.18 tcp_recved()

```
void tcp_recved (struct tcp_pcb * pcb, u16_t len)
```

This function should be called by the application when it has processed the data. The purpose is to advertise a larger window when the data has been processed.

Parameters

pcb	the tcp_pcb for which data is read
len	the amount of bytes that have been read by the application

2.3.3.8.4.19 tcp_sent()

```
void tcp_sent (struct tcp_pcb * pcb, tcp_sent_fn sent)
```

Specifies the callback function that should be called when data has successfully been received (i.e., acknowledged) by the remote host. The len argument passed to the callback function gives the amount bytes that was acknowledged by the last acknowledgment.

Parameters

pcb	tcp_pcb to set the sent callback
sent	callback function to call for this pcb when data is successfully sent

2.3.3.8.4.20 tcp_shutdown()

```
err_t tcp_shutdown (struct tcp_pcb * pcb, int shut_rx, int shut_tx)
```

Causes all or part of a full-duplex connection of this PCB to be shut down. This doesn't deallocate the PCB unless shutting down both sides! Shutting down both sides is the same as calling tcp_close, so if it succeeds (i.e. returns ER_OK), the PCB must not be referenced any more!

Parameters

Returns ERR_OK if shutdown succeeded (or the PCB has already been shut down) another err_t on error.

pcb	PCB to shutdown
shut_rx	shut down receive side if this is != 0
shut_tx	shut down send side if this is != 0

2.3.3.8.4.21 tcp_write()

`err_t tcp_write (struct tcp_pcb * pcb, const void * arg, u16_t len, u8_t apiflags)`

Write data for sending (but does not send it immediately).

It waits in the expectation of more data being sent soon (as it can send them more efficiently by combining them together). To prompt the system to send data now, call tcp_output() after calling tcp_write().

This function enqueues the data pointed to by the argument dataptr. The length of the data is passed as the len parameter. The apiflags can be one or more of:

- TCP_WRITE_FLAG_COPY: indicates whether the new memory should be allocated for the data to be copied into. If this flag is not given, no new memory should be allocated and the data should only be referenced by pointer. This also means that the memory behind dataptr must not change until the data is ACKed by the remote host
- TCP_WRITE_FLAG_MORE: indicates that more data follows. If this is omitted, the PSH flag is set in the last segment created by this call to tcp_write. If this flag is given, the PSH flag is not set.

The tcp_write() function will fail and return ERR_MEM if the length of the data exceeds the current send buffer size or if the length of the queue of outgoing segment is larger than the upper limit defined in lwipopts.h. The number of bytes available in the output queue can be retrieved with the tcp_sndbuf() function.

The proper way to use this function is to call the function with at most tcp_sndbuf() bytes of data. If the function returns ERR_MEM, the application should wait until some of the currently enqueued data has been successfully received by the other host and try again.

Parameters

pcb	Protocol control block for the TCP connection to enqueue data for.
arg	Pointer to the data to be enqueued for sending.
len	Data length in bytes
apiflags	combination of following flags : • TCP_WRITE_FLAG_COPY (0x01) data will be copied into memory belonging to the stack • TCP_WRITE_FLAG_MORE (0x02) for TCP connection, PSH flag will not be set on last segment sent,

Returns ERR_OK if enqueued, another err_t on error

2.3.3.8.5 ext arguments

2.3.3.8.5.1 Functions

- u8_t tcp_ext_arg_alloc_id (void)
- void tcp_ext_arg_set_callbacks (struct tcp_pcb *pcb, u8_t id, const struct tcp_ext_arg_callbacks *const callbacks)
- void tcp_ext_arg_set (struct tcp_pcb *pcb, u8_t id, void *arg)
- void * tcp_ext_arg_get (const struct tcp_pcb *pcb, u8_t id)

2.3.3.8.5.2 Detailed Description

Additional data storage per tcp pcb **See also** TCP

When LWIP_TCP_PCB_NUM_EXT_ARGS is > 0, every tcp pcb (including listen pcb) includes a number of additional argument entries in an array.

To support memory management, in addition to a 'void *', callbacks can be provided to manage transition from listening pcbs to connections and to deallocate memory when a pcb is deallocated (see struct tcp_ext_arg_callbacks).

After allocating this index, use tcp_ext_arg_set and tcp_ext_arg_get to store and load arguments from this index for a given pcb.

2.3.3.8.5.3 Function Documentation

2.3.3.8.5.4 tcp_ext_arg_alloc_id()

```
u8_t tcp_ext_arg_alloc_id (void )
```

Allocate an index to store data in ext_args member of struct tcp_pcb. Returned value is an index in mentioned array. The index is *global* over all pcbs!

When LWIP_TCP_PCB_NUM_EXT_ARGS is > 0, every tcp pcb (including listen pcb) includes a number of additional argument entries in an array.

To support memory management, in addition to a 'void *', callbacks can be provided to manage transition from listening pcbs to connections and to deallocate memory when a pcb is deallocated (see struct tcp_ext_arg_callbacks).

After allocating this index, use tcp_ext_arg_set and tcp_ext_arg_get to store and load arguments from this index for a given pcb.

Returns a unique index into struct tcp_pcb.ext_args

2.3.3.8.5.5 tcp_ext_arg_get()

```
void * tcp_ext_arg_get (const struct tcp_pcb * pcb, u8_t id)
```

Set data for a given index of ext_args on the specified pcb.

Parameters

pcb	tcp_pcb for which to set the data
id	ext_args index to set (allocated via tcp_ext_arg_alloc_id)

Returns data pointer at the given index

2.3.3.8.5.6 tcp_ext_arg_set()

```
void tcp_ext_arg_set (struct tcp_pcb * pcb, u8_t id, void * arg)
```

Set data for a given index of ext_args on the specified pcb.

Parameters

pcb	tcp_pcb for which to set the data
id	ext_args index to set (allocated via tcp_ext_arg_alloc_id)
arg	data pointer to set

pcb	tcp_pcb for which to set the callback
id	ext_args index to set (allocated via tcp_ext_arg_alloc_id)
callbacks	callback table (const since it is referenced, not copied!)

2.3.3.8.5.7 tcp_ext_arg_set_callbacks()

```
void tcp_ext_arg_set_callbacks (struct tcp_pcb * pcb, u8_t id, const struct tcp_ext_arg_ca
*const callbacks)
```

Set callbacks for a given index of ext_args on the specified pcb.

Parameters

2.3.3.9 UDP

2.3.3.9.1 Functions

- err_t udp_send (struct udp_pcb *pcb, struct pbuf *p)
- err_t udp_sendto (struct udp_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, u16_t dst_port)
- err_t udp_sendto_if (struct udp_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, u16_t dst_port, struct netif *netif)
- err_t udp_sendto_if_src (struct udp_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, u16_t dst_port, struct netif *netif, const ip_addr_t *src_ip)
- err_t udp_bind (struct udp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port)
- void udp_bind_netif (struct udp_pcb *pcb, const struct netif *netif)
- err_t udp_connect (struct udp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port)
- void udp_disconnect (struct udp_pcb *pcb)
- void udp_recv (struct udp_pcb *pcb, udp_recv_fn recv, void *recv_arg)
- void udp_remove (struct udp_pcb *pcb)
- struct udp_pcb * udp_new (void)
- struct udp_pcb * udp_new_ip_type (u8_t type)

2.3.3.9.2 Detailed Description

User Datagram Protocol module **See also** APIs

2.3.3.9.3 Function Documentation

2.3.3.9.3.1 udp_bind()

```
err_t udp_bind (struct udp_pcb * pcb, const ip_addr_t * ipaddr, u16_t port)
```

Bind an UDP PCB.

Parameters

ipaddr & port are expected to be in the same byte order as in the pcb.

Returns lwIP error code.

- ERR_OK. Successful. No error occurred.
- ERR_USE. The specified ipaddr and port are already bound to by another UDP PCB.

See also udp_disconnect()

pcb	UDP PCB to be bound with a local address ipaddr and port.
ipaddr	local IP address to bind with. Use IP_ANY_TYPE to bind to all local interfaces.
port	local UDP port to bind with. Use 0 to automatically bind to a random port between UDP_LOCAL_PORT_RANGE_START and UDP_LOCAL_PORT_RANGE_END.

2.3.3.9.3.2 udp_bind_netif()

```
void udp_bind_netif (struct udp_pcb * pcb, const struct netif * netif)
```

Bind an UDP PCB to a specific netif. After calling this function, all packets received via this PCB are guaranteed to have come in via the specified netif, and all outgoing packets will go out via the specified netif.

Parameters

pcb	UDP PCB to be bound.
netif	netif to bind udp pcb to. Can be NULL.

See also udp_disconnect()

2.3.3.9.3.3 udp_connect()

```
err_t udp_connect (struct udp_pcb * pcb, const ip_addr_t * ipaddr, u16_t port)
```

Sets the remote end of the pcb. This function does not generate any network traffic, but only sets the remote address of the pcb.

Parameters

pcb	UDP PCB to be connected with remote address ipaddr and port.
ipaddr	remote IP address to connect with.
port	remote UDP port to connect with.

Returns lwIP error code

ipaddr & port are expected to be in the same byte order as in the pcb.

The udp pcb is bound to a random local port if not already bound.

See also udp_disconnect()

2.3.3.9.3.4 udp_disconnect()

```
void udp_disconnect (struct udp_pcb * pcb)
```

Remove the remote end of the pcb. This function does not generate any network traffic, but only removes the remote address of the pcb.

Parameters

2.3.3.9.3.5 udp_new()

```
struct udp_pcb * udp_new (void )
```

Creates a new UDP pcb which can be used for UDP communication. The pcb is not active until it has either been bound to a local address or connected to a remote address. **See also** MEMP_NUM_UDP_PCB

Returns The UDP PCB which was created. NULL if the PCB data structure could not be allocated.

See also udp_remove()

pcb	the udp pcb to disconnect.

2.3.3.9.3.6 udp_new_ip_type()

```
struct udp_pcb * udp_new_ip_type (u8_t type)
```

Create a UDP PCB for specific IP type. The pcb is not active until it has either been bound to a local address or connected to a remote address. **See also** MEMP_NUM_UDP_PCB

Parameters

type	IP address type, see lwip_ip_addr_type definitions. If you want to listen to IPv4 and IPv6 (dual-stack) packets, supply IPADDR_TYPE_ANY as argument and bind to IP_ANY_TYPE.

Returns The UDP PCB which was created. NULL if the PCB data structure could not be allocated.

See also udp_remove()

2.3.3.9.3.7 udp_recv()

```
void udp_recv (struct udp_pcb * pcb, udp_recv_fn recv, void * recv_arg)
```

Set a receive callback for a UDP PCB. This callback will be called when receiving a datagram for the pcb.

Parameters

pcb	the pcb for which to set the recv callback
recv	function pointer of the callback function
recv_arg	additional argument to pass to the callback function

2.3.3.9.3.8 udp_remove()

```
void udp_remove (struct udp_pcb * pcb)
```

Removes and deallocates the pcb.

Parameters

See also udp_new()

2.3.3.9.3.9 udp_send()

```
err_t udp_send (struct udp_pcb * pcb, struct pbuf * p)
```

Sends the pbuf p using UDP. The pbuf is not deallocated.

Parameters

The datagram will be sent to the current remote_ip & remote_port stored in pcb. If the pcb is not bound to a port, it will automatically be bound to a random port.

Returns lwIP error code.

- ERR_OK. Successful. No error occurred.
- ERR_MEM. Out of memory.
- ERR_RTE. Could not find route to destination address.

pcb	UDP PCB to be removed. The PCB is removed from the list of UDP PCB's and the data structure is freed from memory.

pcb	UDP PCB used to send the data.
p	chain of pbuf's to be sent.

- ERR_VAL. No PCB or PCB is dual-stack
- More errors could be returned by lower protocol layers.

See also udp_disconnect() udp_sendto()

2.3.3.9.3.10 udp_sendto()

```
err_t udp_sendto (struct udp_pcb * pcb, struct pbuf * p, const ip_addr_t * dst_ip, u16_t
dst_port)
```

Send data to a specified address using UDP.

Parameters

pcb	UDP PCB used to send the data.
p	chain of pbuf's to be sent.
dst_ip	Destination IP address.
dst_port	Destination UDP port.

dst_ip & dst_port are expected to be in the same byte order as in the pcb.

If the PCB already has a remote address association, it will be restored after the data is sent.

Returns lwIP error code (

See also udp_send for possible error codes)

udp_disconnect() udp_send()

2.3.3.9.3.11 udp_sendto_if()

```
err_t udp_sendto_if (struct udp_pcb * pcb, struct pbuf * p, const ip_addr_t * dst_ip, u16_
dst_port, struct netif * netif)
```

Send data to a specified address using UDP. The netif used for sending can be specified.

This function exists mainly for DHCP, to be able to send UDP packets on a netif that is still down.

Parameters

dst_ip & dst_port are expected to be in the same byte order as in the pcb.

Returns lwIP error code (

See also udp_send for possible error codes)

udp_disconnect() udp_send()

2.3.3.9.3.12 udp_sendto_if_src()

```
err_t udp_sendto_if_src (struct udp_pcb * pcb, struct pbuf * p, const ip_addr_t * dst_ip,
u16_t dst_port, struct netif * netif, const ip_addr_t * src_ip)
```

Same as udp_sendto_if, but with source address

pcb	UDP PCB used to send the data.
p	chain of pbuf's to be sent.
dst_ip	Destination IP address.
dst_port	Destination UDP port.
netif	the netif used for sending.

2.3.3.10 Ethernet

2.3.3.10.1 Functions

- err_t ethernet_output (struct netif *netif, struct pbuf *p, const struct eth_addr *src, const struct eth_addr *dst, u16_t eth_type)

2.3.3.10.2 Detailed Description

2.3.3.10.3 Function Documentation

2.3.3.10.3.1 ethernet_output()

```
err_t ethernet_output (struct netif * netif, struct pbuf * p, const struct eth_addr * src,
const struct eth_addr * dst, u16_t eth_type)
```

Send an ethernet packet on the network using netif->linkoutput(). The ethernet header is filled in before sending.

See also LWIP_HOOK_VLAN_SET

Parameters

netif	the lwIP network interface on which to send the packet
p	the packet to send. pbuf layer must be PBUF_LINK.
src	the source MAC address to be copied into the ethernet header
dst	the destination MAC address to be copied into the ethernet header
eth_type	ethernet type (lwip_ieee_eth_type)

Returns ERR_OK if the packet was sent, any other err_t on failure

2.3.4 Sequential-style APIs

2.3.4.1 Modules

- Netconn API
- NETIF API

2.3.4.2 Detailed Description

Sequential-style APIs, blocking functions. More overhead, but can be called from any thread except TCPIP thread. The sequential API provides a way for ordinary, sequential, programs to use the lwIP stack. It is quite similar to the BSD socket API. The model of execution is based on the blocking open-read-write-close paradigm. Since the TCP/IP stack is event based by nature, the TCP/IP code and the application program must reside in different execution contexts (threads).

2.3.4.3 Netconn API

2.3.4.3.1 Modules

- Common functions
- TCP only
- UDP only
- Network buffers

2.3.4.3.2 Detailed Description

Thread-safe, to be called from non-TCPIP threads only. TX/RX handling based on Network buffers (containing Packet buffers (PBUF)) to avoid copying data around.

2.3.4.3.3 Common functions

2.3.4.3.3.1 Macros

- #define netconn_new(t) netconn_new_with_proto_and_callback(t, 0, NULL)
- #define netconn_set_ipv6only(conn, val)
- #define netconn_get_ipv6only(conn) (((conn)->flags & NETCONN_FLAG_IPV6_V6ONLY) != 0)

2.3.4.3.3.2 Enumerations

- enum netconn_type { }

2.3.4.3.3.3 Functions

- err_t netconn_prepare_delete (struct netconn *conn)
- err_t netconn_delete (struct netconn *conn)
- err_t netconn_bind (struct netconn *conn, const ip_addr_t *addr, u16_t port)
- err_t netconn_bind_if (struct netconn *conn, u8_t if_idx)
- err_t netconn_connect (struct netconn *conn, const ip_addr_t *addr, u16_t port)
- err_t netconn_recv (struct netconn *conn, struct netbuf **new_buf)
- err_t netconn_err (struct netconn *conn)
- err_t netconn_gethostbyname_addrtype (const char *name, ip_addr_t *addr, u8_t dns_addrtype)

2.3.4.3.3.4 Detailed Description

For use with TCP and UDP

2.3.4.3.3.5 Macro Definition Documentation

2.3.4.3.3.6 netconn_get_ipv6only

```
#define netconn_get_ipv6only( conn)    (((conn)->flags & NETCONN_FLAG_IPV6_V6ONLY) != 0)
```

TCP: Get the IPv6 ONLY status of netconn calls (see NETCONN_FLAG_IPV6_V6ONLY)

2.3.4.3.3.7 netconn_new

`#define netconn_new( t) netconn_new_with_proto_and_callback(t, 0, NULL)`

Create new netconn connection

Parameters

t	netconn_type

2.3.4.3.3.8 netconn_set_ipv6only

`#define netconn_set_ipv6only( conn, val)` **Value:**

```
  do { if(val) { \
  netconn_set_flags(conn, NETCONN_FLAG_IPV6_V6ONLY); \
} else { \
  netconn_clear_flags(conn, NETCONN_FLAG_IPV6_V6ONLY); }} while(0)
```

TCP: Set the IPv6 ONLY status of netconn calls (see NETCONN_FLAG_IPV6_V6ONLY)

2.3.4.3.3.9 Enumeration Type Documentation

2.3.4.3.3.10 netconn_type

`enum netconn_type`

Protocol family and type of the netconn

NETCONN_TCP	TCP IPv4
NETCONN_TCP_IPV6	TCP IPv6
NETCONN_UDP	UDP IPv4
NETCONN_UDPLITE	UDP IPv4 lite
NETCONN_UDPNOCHKSUM	UDP IPv4 no checksum
NETCONN_UDP_IPV6	UDP IPv6 (dual-stack by default, unless you call netconn_set_ipv6only)
NETCONN_UDPLITE_IPV6	UDP IPv6 lite (dual-stack by default, unless you call netconn_set_ipv6only)
NETCONN_UDPNOCHKSUM_IPV6	UDP IPv6 no checksum (dual-stack by default, unless you call netconn_set_ipv6only)
NETCONN_RAW	Raw connection IPv4

2.3.4.3.3.11 Function Documentation

2.3.4.3.3.12 netconn_bind()

`err_t netconn_bind (struct netconn * conn, const ip_addr_t * addr, u16_t port)`

Bind a netconn to a specific local IP address and port. Binding one netconn twice might not always be checked correctly!

Parameters

conn	the netconn to bind
addr	the local IP address to bind the netconn to (use IP4_ADDR_ANY/IP6_ADDR_ANY to bind to all addresses)
port	the local port to bind the netconn to (not used for RAW)

Returns ERR_OK if bound, any other err_t on failure

2.3.4.3.3.13 netconn_bind_if()

```
err_t netconn_bind_if (struct netconn * conn, u8_t if_idx)
```

Bind a netconn to a specific interface and port. Binding one netconn twice might not always be checked correctly!

Parameters

conn	the netconn to bind
if_idx	the local interface index to bind the netconn to

Returns ERR_OK if bound, any other err_t on failure

2.3.4.3.3.14 netconn_connect()

```
err_t netconn_connect (struct netconn * conn, const ip_addr_t * addr, u16_t port)
```

Connect a netconn to a specific remote IP address and port.

Parameters

conn	the netconn to connect
addr	the remote IP address to connect to
port	the remote port to connect to (no used for RAW)

Returns ERR_OK if connected, return value of tcp_/udp_/raw_connect otherwise

2.3.4.3.3.15 netconn_delete()

```
err_t netconn_delete (struct netconn * conn)
```

Close a netconn 'connection' and free its resources. UDP and RAW connection are completely closed, TCP pcbs might still be in a waitstate after this returns.

Parameters

conn	the netconn to delete

Returns ERR_OK if the connection was deleted

2.3.4.3.3.16 netconn_err()

```
err_t netconn_err (struct netconn * conn)
```

Get and reset pending error on a netconn

Parameters

Returns and pending error or ERR_OK if no error was pending

2.3.4.3.3.17 netconn_gethostbyname_addrtype()

```
err_t netconn_gethostbyname_addrtype (const char * name, ip_addr_t * addr, u8_t dns_addrty
```

Execute a DNS query, only one IP address is returned

Parameters

Returns ERR_OK: resolving succeeded ERR_MEM: memory error, try again later ERR_ARG: dns client not initialized or invalid hostname ERR_VAL: dns server response was invalid

Parameters

conn	the netconn to get the error from

name	a string representation of the DNS host name to query
addr	a preallocated ip_addr_t where to store the resolved IP address

2.3.4.3.3.18 netconn_prepare_delete()

```
err_t netconn_prepare_delete (struct netconn * conn)
```

Close a netconn 'connection' and free all its resources but not the netconn itself. UDP and RAW connection are completely closed, TCP pcbs might still be in a waitstate after this returns.

Parameters

Returns ERR_OK if the connection was deleted

2.3.4.3.3.19 netconn_recv()

```
err_t netconn_recv (struct netconn * conn, struct netbuf ** new_buf)
```

Receive data (in form of a netbuf containing a packet buffer) from a netconn

Parameters

Returns ERR_OK if data has been received, an error code otherwise (timeout, memory error or another error)

2.3.4.3.4 TCP only

2.3.4.3.4.1 Functions

- err_t netconn_listen_with_backlog (struct netconn *conn, u8_t backlog)
- err_t netconn_accept (struct netconn *conn, struct netconn **new_conn)
- err_t netconn_recv_tcp_pbuf (struct netconn *conn, struct pbuf **new_buf)
- err_t netconn_recv_tcp_pbuf_flags (struct netconn *conn, struct pbuf **new_buf, u8_t apiflags)
- err_t netconn_write_partly (struct netconn *conn, const void *dataptr, size_t size, u8_t apiflags, size_t *bytes_written)
- err_t netconn_close (struct netconn *conn)
- err_t netconn_shutdown (struct netconn *conn, u8_t shut_rx, u8_t shut_tx)

2.3.4.3.4.2 Detailed Description

TCP only functions

2.3.4.3.4.3 Function Documentation

2.3.4.3.4.4 netconn_accept()

```
err_t netconn_accept (struct netconn * conn, struct netconn ** new_conn)
```

Accept a new connection on a TCP listening netconn.

Parameters

Returns ERR_OK if a new connection has been received or an error code otherwise

dns_addrtype	IP address type (IPv4 / IPv6)

conn	the netconn to delete

conn	the netconn from which to receive data
new_buf	pointer where a new netbuf is stored when received data

2.3.4.3.4.5 netconn_close()

`err_t netconn_close (struct netconn * conn)`

Close a TCP netconn (doesn't delete it).

Parameters

Returns ERR_OK if the netconn was closed, any other err_t on error

2.3.4.3.4.6 netconn_listen_with_backlog()

`err_t netconn_listen_with_backlog (struct netconn * conn, u8_t backlog)`

Set a TCP netconn into listen mode

Parameters

Returns ERR_OK if the netconn was set to listen (UDP and RAW netconns don't return any error (yet?))

2.3.4.3.4.7 netconn_recv_tcp_pbuf()

`err_t netconn_recv_tcp_pbuf (struct netconn * conn, struct pbuf ** new_buf)`

Receive data (in form of a pbuf) from a TCP netconn

Parameters

Returns ERR_OK if data has been received, an error code otherwise (timeout, memory error or another error,

See also netconn_recv_data) ERR_ARG if conn is not a TCP netconn

2.3.4.3.4.8 netconn_recv_tcp_pbuf_flags()

`err_t netconn_recv_tcp_pbuf_flags (struct netconn * conn, struct pbuf ** new_buf, u8_t apiflags)`

Receive data (in form of a pbuf) from a TCP netconn

Parameters

Returns ERR_OK if data has been received, an error code otherwise (timeout, memory error or another error,

See also netconn_recv_data) ERR_ARG if conn is not a TCP netconn

2.3.4.3.4.9 netconn_shutdown()

`err_t netconn_shutdown (struct netconn * conn, u8_t shut_rx, u8_t shut_tx)`

Shut down one or both sides of a TCP netconn (doesn't delete it).

Parameters

Returns ERR_OK if the netconn was closed, any other err_t on error

conn	the TCP listen netconn
new_conn	pointer where the new connection is stored

conn	the TCP netconn to close

conn	the tcp netconn to set to listen mode
backlog	the listen backlog, only used if TCP_LISTEN_BACKLOG==1

2.3.4.3.4.10 netconn_write_partly()

```
err_t netconn_write_partly (struct netconn * conn, const void * dataptr, size_t size, u8_t
apiflags, size_t * bytes_written)
```

Send data over a TCP netconn.

Parameters

Returns ERR_OK if data was sent, any other err_t on error

2.3.4.3.5 UDP only

2.3.4.3.5.1 Functions

- err_t netconn_disconnect (struct netconn *conn)
- err_t netconn_sendto (struct netconn *conn, struct netbuf *buf, const ip_addr_t *addr, u16_t port)
- err_t netconn_send (struct netconn *conn, struct netbuf *buf)
- err_t netconn_join_leave_group (struct netconn *conn, const ip_addr_t *multiaddr, const ip_addr_t *netif_addr, enum netconn_igmp join_or_leave)
- err_t netconn_join_leave_group_netif (struct netconn *conn, const ip_addr_t *multiaddr, u8_t if_idx, enum netconn_igmp join_or_leave)

2.3.4.3.5.2 Detailed Description

UDP only functions

2.3.4.3.5.3 Function Documentation

2.3.4.3.5.4 netconn_disconnect()

```
err_t netconn_disconnect (struct netconn * conn)
```

Disconnect a netconn from its current peer (only valid for UDP netconns).

Parameters

Returns See err_t

2.3.4.3.5.5 netconn_join_leave_group()

```
err_t netconn_join_leave_group (struct netconn * conn, const ip_addr_t * multiaddr, const
ip_addr_t * netif_addr, enum netconn_igmp join_or_leave)
```

Join multicast groups for UDP netconns.

Parameters

Returns ERR_OK if the action was taken, any err_t on error

conn	the netconn from which to receive data
new_buf	pointer where a new pbuf is stored when received data

conn	the netconn from which to receive data
new_buf	pointer where a new pbuf is stored when received data
apiflags	flags that control function behaviour. For now only: • NETCONN_DONTBLOCK: only read data that is available now, don't wait for more data

conn	the TCP netconn to shut down
shut_rx	shut down the RX side (no more read possible after this)
shut_tx	shut down the TX side (no more write possible after this)

conn	the TCP netconn over which to send data
dataptr	pointer to the application buffer that contains the data to send
size	size of the application data to send
apiflags	combination of following flags : • NETCONN_COPY: data will be copied into memory belonging to the stack • NETCONN_MORE: for TCP connection, PSH flag will be set on last segment sent • NETCONN_DONTBLOCK: only write the data if all data can be written at once
bytes_written	pointer to a location that receives the number of written bytes

conn	the netconn to disconnect

conn	the UDP netconn for which to change multicast addresses
multiaddr	IP address of the multicast group to join or leave
netif_addr	the IP address of the network interface on which to send the igmp message
join_or_leave	flag whether to send a join- or leave-message

2.3.4.3.5.6 netconn_join_leave_group_netif()

```
err_t netconn_join_leave_group_netif (struct netconn * conn, const ip_addr_t * multiaddr,
u8_t if_idx, enum netconn_igmp join_or_leave)
```

Join multicast groups for UDP netconns.

Parameters

conn	the UDP netconn for which to change multicast addresses
multiaddr	IP address of the multicast group to join or leave
if_idx	the index of the netif
join_or_leave	flag whether to send a join- or leave-message

Returns ERR_OK if the action was taken, any err_t on error

2.3.4.3.5.7 netconn_send()

```
err_t netconn_send (struct netconn * conn, struct netbuf * buf)
```

Send data over a UDP or RAW netconn (that is already connected).

Parameters

conn	the UDP or RAW netconn over which to send data
buf	a netbuf containing the data to send

Returns ERR_OK if data was sent, any other err_t on error

2.3.4.3.5.8 netconn_sendto()

```
err_t netconn_sendto (struct netconn * conn, struct netbuf * buf, const ip_addr_t * addr,
u16_t port)
```

Send data (in form of a netbuf) to a specific remote IP address and port. Only to be used for UDP and RAW netconns (not TCP).

Parameters

conn	the netconn over which to send data
buf	a netbuf containing the data to send
addr	the remote IP address to which to send the data
port	the remote port to which to send the data

Returns ERR_OK if data was sent, any other err_t on error

2.3.4.3.6 Network buffers

2.3.4.3.6.1 Functions

- struct netbuf * netbuf_new (void)
- void netbuf_delete (struct netbuf *buf)
- void * netbuf_alloc (struct netbuf *buf, u16_t size)
- void netbuf_free (struct netbuf *buf)

- err_t netbuf_ref (struct netbuf *buf, const void *dataptr, u16_t size)
- void netbuf_chain (struct netbuf *head, struct netbuf *tail)
- err_t netbuf_data (struct netbuf *buf, void **dataptr, u16_t *len)
- s8_t netbuf_next (struct netbuf *buf)
- void netbuf_first (struct netbuf *buf)

2.3.4.3.6.2 Detailed Description

Network buffer descriptor for Netconn API. Based on Packet buffers (PBUF) internally to avoid copying data around. Buffers must not be shared across multiple threads, all functions except netbuf_new() and netbuf_delete() are not thread-safe.

2.3.4.3.6.3 Function Documentation

2.3.4.3.6.4 netbuf_alloc()

```
void * netbuf_alloc (struct netbuf * buf, u16_t size)
```

Allocate memory for a packet buffer for a given netbuf.

Parameters

buf	the netbuf for which to allocate a packet buffer
size	the size of the packet buffer to allocate

Returns pointer to the allocated memory NULL if no memory could be allocated

2.3.4.3.6.5 netbuf_chain()

```
void netbuf_chain (struct netbuf * head, struct netbuf * tail)
```

Chain one netbuf to another (**See also** pbuf_chain)

Parameters

head	the first netbuf
tail	netbuf to chain after head, freed by this function, may not be reference after returning

2.3.4.3.6.6 netbuf_data()

```
err_t netbuf_data (struct netbuf * buf, void ** dataptr, u16_t * len)
```

Get the data pointer and length of the data inside a netbuf.

Parameters

buf	netbuf to get the data from
dataptr	pointer to a void pointer where to store the data pointer
len	pointer to an u16_t where the length of the data is stored

Returns ERR_OK if the information was retrieved, ERR_BUF on error.

2.3.4.3.6.7 netbuf_delete()

```
void netbuf_delete (struct netbuf * buf)
```

Deallocate a netbuf allocated by netbuf_new().

Parameters

buf	pointer to a netbuf allocated by netbuf_new()

2.3.4.3.6.8 netbuf_first()

```
void netbuf_first (struct netbuf * buf)
```

Move the current data pointer of a packet buffer contained in a netbuf to the beginning of the packet. The packet buffer itself is not modified.

Parameters

buf	the netbuf to modify

2.3.4.3.6.9 netbuf_free()

```
void netbuf_free (struct netbuf * buf)
```

Free the packet buffer included in a netbuf

Parameters

buf	pointer to the netbuf which contains the packet buffer to free

2.3.4.3.6.10 netbuf_new()

```
struct netbuf * netbuf_new (void )
```

Create (allocate) and initialize a new netbuf. The netbuf doesn't yet contain a packet buffer!

Returns a pointer to a new netbuf NULL on lack of memory

2.3.4.3.6.11 netbuf_next()

```
s8_t netbuf_next (struct netbuf * buf)
```

Move the current data pointer of a packet buffer contained in a netbuf to the next part. The packet buffer itself is not modified.

Parameters

Returns -1 if there is no next part 1 if moved to the next part but now there is no next part 0 if moved to the next part and there are still more parts

2.3.4.3.6.12 netbuf_ref()

```
err_t netbuf_ref (struct netbuf * buf, const void * dataptr, u16_t size)
```

Let a netbuf reference existing (non-volatile) data.

Parameters

Returns ERR_OK if data is referenced ERR_MEM if data couldn't be referenced due to lack of memory

buf	the netbuf to modify

buf	netbuf which should reference the data
dataptr	pointer to the data to reference
size	size of the data

2.3.4.4 NETIF API

2.3.4.4.1 Modules

- NETIF related
- DHCPv4
- AUTOIP

2.3.4.4.2 Detailed Description

Thread-safe functions to be called from non-TCPIP threads

2.3.4.4.3 NETIF related

2.3.4.4.3.1 Macros

- #define netifapi_netif_remove(n) netifapi_netif_common(n, netif_remove, NULL)
- #define netifapi_netif_set_up(n) netifapi_netif_common(n, netif_set_up, NULL)
- #define netifapi_netif_set_down(n) netifapi_netif_common(n, netif_set_down, NULL)
- #define netifapi_netif_set_default(n) netifapi_netif_common(n, netif_set_default, NULL)
- #define netifapi_netif_set_link_up(n) netifapi_netif_common(n, netif_set_link_up, NULL)
- #define netifapi_netif_set_link_down(n) netifapi_netif_common(n, netif_set_link_down, NULL)

2.3.4.4.3.2 Functions

- err_t netifapi_netif_add (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw, void *state, netif_init_fn init, netif_input_fn input)
- err_t netifapi_netif_set_addr (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw)
- err_t netifapi_netif_name_to_index (const char *name, u8_t *idx)
- err_t netifapi_netif_index_to_name (u8_t idx, char *name)

2.3.4.4.3.3 Detailed Description

To be called from non-TCPIP threads

2.3.4.4.3.4 Macro Definition Documentation

2.3.4.4.3.5 netifapi_netif_remove

```
#define netifapi_netif_remove( n)    netifapi_netif_common(n, netif_remove, NULL)
```

See also netif_remove()

2.3.4.4.3.6 netifapi_netif_set_default

```
#define netifapi_netif_set_default( n)   netifapi_netif_common(n, netif_set_default, NULL)
```

See also netif_set_default()

2.3.4.4.3.7 netifapi_netif_set_down

```
#define netifapi_netif_set_down( n)   netifapi_netif_common(n, netif_set_down, NULL)
```

See also netif_set_down()

2.3.4.4.3.8 netifapi_netif_set_link_down

```
#define netifapi_netif_set_link_down( n)   netifapi_netif_common(n, netif_set_link_down,
NULL)
```

See also netif_set_link_down()

2.3.4.4.3.9 netifapi_netif_set_link_up

```
#define netifapi_netif_set_link_up( n)   netifapi_netif_common(n, netif_set_link_up, NULL)
```

See also netif_set_link_up()

2.3.4.4.3.10 netifapi_netif_set_up

```
#define netifapi_netif_set_up( n)   netifapi_netif_common(n, netif_set_up, NULL)
```

See also netif_set_up()

2.3.4.4.3.11 Function Documentation

2.3.4.4.3.12 netifapi_netif_add()

```
err_t netifapi_netif_add (struct netif * netif, const ip4_addr_t * ipaddr, const ip4_addr_
* netmask, const ip4_addr_t * gw, void * state, netif_init_fn init, netif_input_fn input)
```

Call netif_add() in a thread-safe way by running that function inside the tcpip_thread context.

Note
for params

See also netif_add()

2.3.4.4.3.13 netifapi_netif_index_to_name()

```
err_t netifapi_netif_index_to_name (u8_t idx, char * name)
```

Call netif_index_to_name() in a thread-safe way by running that function inside the tcpip_thread context.

Parameters

2.3.4.4.3.14 netifapi_netif_name_to_index()

```
err_t netifapi_netif_name_to_index (const char * name, u8_t * idx)
```

Call netif_name_to_index() in a thread-safe way by running that function inside the tcpip_thread context.

Parameters

idx	the interface index of the netif
name	output name of the found netif, empty '\0' string if netif not found. name should be of at least NETIF_NAMESIZE bytes

name	the interface name of the netif
idx	output index of the found netif

2.3.4.4.3.15 netifapi_netif_set_addr()

```
err_t netifapi_netif_set_addr (struct netif * netif, const ip4_addr_t * ipaddr, const ip4_
* netmask, const ip4_addr_t * gw)
```

Call netif_set_addr() in a thread-safe way by running that function inside the tcpip_thread context.

Note
for params

See also netif_set_addr()

2.3.4.4.4 DHCPv4

2.3.4.4.4.1 Macros

- #define netifapi_dhcp_start(n)   netifapi_netif_common(n, NULL, dhcp_start)
- #define netifapi_dhcp_stop(n)   netifapi_netif_common(n, dhcp_stop, NULL)
- #define netifapi_dhcp_inform(n)   netifapi_netif_common(n, dhcp_inform, NULL)
- #define netifapi_dhcp_renew(n)   netifapi_netif_common(n, NULL, dhcp_renew)
- #define netifapi_dhcp_release(n)   netifapi_netif_common(n, NULL, dhcp_release)
- #define netifapi_dhcp_release_and_stop(n)   netifapi_netif_common(n, dhcp_release_and_stop, NULL)

2.3.4.4.4.2 Detailed Description

To be called from non-TCPIP threads

2.3.4.4.4.3 Macro Definition Documentation

2.3.4.4.4.4 netifapi_dhcp_inform

```
#define netifapi_dhcp_inform( n)    netifapi_netif_common(n, dhcp_inform, NULL)
```

See also dhcp_inform()

2.3.4.4.4.5 netifapi_dhcp_release

```
#define netifapi_dhcp_release( n)    netifapi_netif_common(n, NULL, dhcp_release)
```

Deprecated Use netifapi_dhcp_release_and_stop() instead.

2.3.4.4.4.6 netifapi_dhcp_release_and_stop

```
#define netifapi_dhcp_release_and_stop( n)    netifapi_netif_common(n, dhcp_release_and_sto
NULL)
```

See also dhcp_release_and_stop()

2.3.4.4.4.7 netifapi_dhcp_renew

```
#define netifapi_dhcp_renew( n)    netifapi_netif_common(n, NULL, dhcp_renew)
```

See also dhcp_renew()

2.3.4.4.4.8 netifapi_dhcp_start

```
#define netifapi_dhcp_start( n)    netifapi_netif_common(n, NULL, dhcp_start)
```

See also dhcp_start()

2.3.4.4.4.9 netifapi_dhcp_stop

```
#define netifapi_dhcp_stop( n)    netifapi_netif_common(n, dhcp_stop, NULL)
```

Deprecated Use netifapi_dhcp_release_and_stop() instead.

2.3.4.4.5 AUTOIP

2.3.4.4.5.1 Macros

- #define netifapi_autoip_start(n)   netifapi_netif_common(n, NULL, autoip_start)
- #define netifapi_autoip_stop(n)   netifapi_netif_common(n, NULL, autoip_stop)

2.3.4.4.5.2 Detailed Description

To be called from non-TCPIP threads

2.3.4.4.5.3 Macro Definition Documentation

2.3.4.4.5.4 netifapi_autoip_start

```
#define netifapi_autoip_start( n)    netifapi_netif_common(n, NULL, autoip_start)
```

See also autoip_start()

2.3.4.4.5.5 netifapi_autoip_stop

```
#define netifapi_autoip_stop( n)    netifapi_netif_common(n, NULL, autoip_stop)
```

See also autoip_stop()

2.3.5 Socket API

2.3.5.1 Modules

- Interface Identification API
- NETDB API

2.3.5.2 Detailed Description

BSD-style socket API. Thread-safe, to be called from non-TCPIP threads only. Can be activated by defining LWIP_SOCKET to 1. Header is in posix/sys/socket.h The socket API is a compatibility API for existing applications, currently it is built on top of the sequential API. It is meant to provide all functions needed to run socket API applications running on other platforms (e.g. unix / windows etc.). However, due to limitations in the specification of this API, there might be incompatibilities that require small modifications of existing programs.

2.3.5.3 Interface Identification API

2.3.5.3.1 Functions

- char * lwip_if_indextoname (unsigned int ifindex, char *ifname)
- unsigned int lwip_if_nametoindex (const char *ifname)

2.3.5.3.2 Detailed Description

2.3.5.3.3 Function Documentation

2.3.5.3.3.1 lwip_if_indextoname()

```
char * lwip_if_indextoname (unsigned int ifindex, char * ifname)
```

Maps an interface index to its corresponding name.

Parameters

ifindex	interface index
ifname	shall point to a buffer of at least {IF_NAMESIZE} bytes

Returns If ifindex is an interface index, then the function shall return the value supplied in ifname, which points to a buffer now containing the interface name. Otherwise, the function shall return a NULL pointer.

2.3.5.3.3.2 lwip_if_nametoindex()

```
unsigned int lwip_if_nametoindex (const char * ifname)
```

Returns the interface index corresponding to name ifname.

Parameters

ifname	Interface name

Returns The corresponding index if ifname is the name of an interface; otherwise, zero.

2.3.5.4 NETDB API

2.3.5.4.1 Detailed Description

2.4 NETIFs

2.4.1 Modules

- IEEE 802.1D bridge

- 6LoWPAN (RFC4944)
- 6LoWPAN over BLE (RFC7668)
- PPP
- SLIP
- ZEP - ZigBee Encapsulation Protocol

2.4.2 Detailed Description

2.4.3 IEEE 802.1D bridge

2.4.3.1 Modules

- Options
- FDB example code

2.4.3.2 Data Structures

- struct bridgeif_initdata_s

2.4.3.3 Macros

- #define BRIDGEIF_INITDATA1(max_ports, max_fdb_dynamic_entries, max_fdb_static_entries, ethaddr) {ethaddr, max_ports, max_fdb_dynamic_entries, max_fdb_static_entries}
- #define BRIDGEIF_INITDATA2(max_ports, max_fdb_dynamic_entries, max_fdb_static_entries, e0, e1, e2, e3, e4, e5) {{e0, e1, e2, e3, e4, e5}, max_ports, max_fdb_dynamic_entries, max_fdb_static_entries}

2.4.3.4 Typedefs

- typedef struct bridgeif_initdata_s bridgeif_initdata_t

2.4.3.5 Functions

- err_t bridgeif_fdb_add (struct netif *bridgeif, const struct eth_addr *addr, bridgeif_portmask_t ports)
- err_t bridgeif_fdb_remove (struct netif *bridgeif, const struct eth_addr *addr)
- err_t bridgeif_init (struct netif *netif)
- err_t bridgeif_add_port (struct netif *bridgeif, struct netif *portif)

2.4.3.6 Detailed Description

This file implements an IEEE 802.1D bridge by using a multilayer netif approach (one hardware-independent netif for the bridge that uses hardware netifs for its ports). On transmit, the bridge selects the outgoing port(s). On receive, the port netif calls into the bridge (via its netif->input function) and the bridge selects the port(s) (and/or its netif->input function) to pass the received pbuf to.

Usage:

- add the port netifs just like you would when using them as dedicated netif without a bridge

- only NETIF_FLAG_ETHARP/NETIF_FLAG_ETHERNET netifs are supported as bridge ports
- add the bridge port netifs without IPv4 addresses (i.e. pass 'NULL, NULL, NULL')
- don't add IPv6 addresses to the port netifs!

• set up the bridge configuration in a global variable of type 'bridgeif_initdata_t' that contains

  - the MAC address of the bridge
  - some configuration options controlling the memory consumption (maximum number of ports and FDB entries)
  - e.g. for a bridge MAC address 00-01-02-03-04-05, 2 bridge ports, 1024 FDB entries + 16 static MAC entries: bridgeif_initdata_t mybridge_initdata = BRIDGEIF_INITDATA1(2, 1024, 16, ETH_ADDR(0, 1, 2, 3, 4, 5));

• add the bridge netif (with IPv4 config): struct netif bridge_netif; netif_add(&bridge_netif, &my_ip, &my_netmask, &my_gw, &mybridge_initdata, bridgeif_init, tcpip_input); NOTE: the passed 'input' function depends on BRIDGEIF_PORT_NETIFS_OUTPU setting, which controls where the forwarding is done (netif low level input context vs. tcpip_thread)

• set up all ports netifs and the bridge netif

• When adding a port netif, NETIF_FLAG_ETHARP flag will be removed from a port to prevent ETHARP working on that port netif (we only want one IP per bridge not per port).

• When adding a port netif, its input function is changed to call into the bridge.

2.4.3.7 Macro Definition Documentation

2.4.3.7.1 BRIDGEIF_INITDATA1

```
#define BRIDGEIF_INITDATA1( max_ports, max_fdb_dynamic_entries, max_fdb_static_entries,
ethaddr)   {ethaddr, max_ports, max_fdb_dynamic_entries, max_fdb_static_entries}
```

Use this for constant initialization of a bridgeif_initdat_t (ethaddr must be passed as ETH_ADDR())

2.4.3.7.2 BRIDGEIF_INITDATA2

```
#define BRIDGEIF_INITDATA2( max_ports, max_fdb_dynamic_entries, max_fdb_static_entries,
e0, e1, e2, e3, e4, e5)   {{e0, e1, e2, e3, e4, e5}, max_ports, max_fdb_dynamic_entries,
max_fdb_static_entries}
```

Use this for constant initialization of a bridgeif_initdat_t (each byte of ethaddr must be passed)

2.4.3.8 Typedef Documentation

2.4.3.8.1 bridgeif_initdata_t

```
typedef struct bridgeif_initdata_s bridgeif_initdata_t
```

Initialisation data for bridgeif_init. An instance of this type must be passed as parameter 'state' to netif_add when the bridge is added.

2.4.3.9 Function Documentation

2.4.3.9.1 bridgeif_add_port()

```
err_t bridgeif_add_port (struct netif * bridgeif, struct netif * portif)
```

Add a port to the bridge

2.4.3.9.2 bridgeif_fdb_add()

```
err_t bridgeif_fdb_add (struct netif * bridgeif, const struct eth_addr * addr, bridgeif_po
ports)
```

Add a static entry to the forwarding database. A static entry marks where frames to a specific eth address (unicast or group address) are forwarded. bits [0..(BRIDGEIF_MAX_PORTS-1)]: hw ports bit [BRIDGEIF_MAX_PORTS]: cpu port 0: drop

2.4.3.9.3 bridgeif_fdb_remove()

```
err_t bridgeif_fdb_remove (struct netif * bridgeif, const struct eth_addr * addr)
```

Remove a static entry from the forwarding database

2.4.3.9.4 bridgeif_init()

```
err_t bridgeif_init (struct netif * netif)
```

Initialization function passed to netif_add().

ATTENTION: A pointer to a bridgeif_initdata_t must be passed as 'state' to netif_add when adding the bridge. I supplies MAC address and controls memory allocation (number of ports, FDB size).

Parameters

netif	the lwip network interface structure for this ethernetif

Returns ERR_OK if the loopif is initialized ERR_MEM if private data couldn't be allocated any other err_t on error

2.4.3.10 Options

2.4.3.10.1 Macros

- #define BRIDGEIF_PORT_NETIFS_OUTPUT_DIRECT NO_SYS
- #define BRIDGEIF_MAX_PORTS 7
- #define BRIDGEIF_DEBUG LWIP_DBG_OFF
- #define BRIDGEIF_FDB_DEBUG LWIP_DBG_OFF
- #define BRIDGEIF_FW_DEBUG LWIP_DBG_OFF

2.4.3.10.2 Detailed Description

2.4.3.10.3 Macro Definition Documentation

2.4.3.10.3.1 BRIDGEIF_DEBUG

```
#define BRIDGEIF_DEBUG    LWIP_DBG_OFF
```

BRIDGEIF_DEBUG: Enable generic debugging in bridgeif.c.

2.4.3.10.3.2 BRIDGEIF_FDB_DEBUG

```
#define BRIDGEIF_FDB_DEBUG    LWIP_DBG_OFF
```

BRIDGEIF_DEBUG: Enable FDB debugging in bridgeif.c.

2.4.3.10.3.3 BRIDGEIF_FW_DEBUG

```
#define BRIDGEIF_FW_DEBUG   LWIP_DBG_OFF
```

BRIDGEIF_DEBUG: Enable forwarding debugging in bridgeif.c.

2.4.3.10.3.4 BRIDGEIF_MAX_PORTS

```
#define BRIDGEIF_MAX_PORTS   7
```

BRIDGEIF_MAX_PORTS: this is used to create a typedef used for forwarding bit-fields: the number of bits required is this + 1 (for the internal/cpu port) (63 is the maximum, resulting in an u64_t for the bit mask) ATTENTION: this controls the maximum number of the implementation only! The max. number of ports per bridge must still be passed via netif_add parameter!

2.4.3.10.3.5 BRIDGEIF_PORT_NETIFS_OUTPUT_DIRECT

```
#define BRIDGEIF_PORT_NETIFS_OUTPUT_DIRECT   NO_SYS
```

BRIDGEIF_PORT_NETIFS_OUTPUT_DIRECT==1: set port netif's 'input' function to call directly into bridgeif code and on top of that, directly call into the selected forwarding port's 'linkoutput' function. This means that the bridgeif input/output path is protected from concurrent access but as well, *all* bridge port netif's drivers must correctly handle concurrent access! == 0: get into tcpip_thread for every input packet (no multithreading) ATTENTION: as ==0 relies on tcpip.h, the default depends on NO_SYS setting

2.4.3.11 FDB example code

2.4.3.11.1 Functions

- void bridgeif_fdb_update_src (void *fdb_ptr, struct eth_addr *src_addr, u8_t port_idx)
- bridgeif_portmask_t bridgeif_fdb_get_dst_ports (void *fdb_ptr, struct eth_addr *dst_addr)
- void * bridgeif_fdb_init (u16_t max_fdb_entries)

2.4.3.11.2 Detailed Description

This file implements an example for an FDB (Forwarding DataBase)

2.4.3.11.3 Function Documentation

2.4.3.11.3.1 bridgeif_fdb_get_dst_ports()

```
bridgeif_portmask_t bridgeif_fdb_get_dst_ports (void * fdb_ptr, struct eth_addr * dst_addr
```

Walk our list of auto-learnt fdb entries and return a port to forward or BR_FLOOD if unknown

2.4.3.11.3.2 bridgeif_fdb_init()

```
void * bridgeif_fdb_init (u16_t max_fdb_entries)
```

Init our simple fdb list

2.4.3.11.3.3 bridgeif_fdb_update_src()

```
void bridgeif_fdb_update_src (void * fdb_ptr, struct eth_addr * src_addr, u8_t port_idx)
```

A real simple and slow implementation of an auto-learning forwarding database that remembers known src mac addresses to know which port to send frames destined for that mac address.

ATTENTION: This is meant as an example only, in real-world use, you should provide a better implementation :-)

2.4.4 6LoWPAN (RFC4944)

2.4.4.1 Functions

- err_t lowpan6_set_context (u8_t idx, const ip6_addr_t *context)
- err_t lowpan6_set_short_addr (u8_t addr_high, u8_t addr_low)
- err_t lowpan6_output (struct netif *netif, struct pbuf *q, const ip6_addr_t *ip6addr)
- err_t lowpan6_input (struct pbuf *p, struct netif *netif)
- err_t lowpan6_set_pan_id (u16_t pan_id)
- err_t tcpip_6lowpan_input (struct pbuf *p, struct netif *inp)

2.4.4.2 Detailed Description

6LowPAN netif implementation

2.4.4.3 Function Documentation

2.4.4.3.1 lowpan6_input()

```
err_t lowpan6_input (struct pbuf * p, struct netif * netif)
```

NETIF input function: don't free the input pbuf when returning != ERR_OK!

2.4.4.3.2 lowpan6_output()

```
err_t lowpan6_output (struct netif * netif, struct pbuf * q, const ip6_addr_t * ip6addr)
```

Resolve and fill-in IEEE 802.15.4 address header for outgoing IPv6 packet.

Perform Header Compression and fragment if necessary.

Parameters

netif	The lwIP network interface which the IP packet will be sent on.
q	The pbuf(s) containing the IP packet to be sent.
ip6addr	The IP address of the packet destination.

Returns err_t

2.4.4.3.3 lowpan6_set_context()

```
err_t lowpan6_set_context (u8_t idx, const ip6_addr_t * context)
```

Set context

2.4.4.3.4 lowpan6_set_pan_id()

```
err_t lowpan6_set_pan_id (u16_t pan_id)
```

Set PAN ID

2.4.4.3.5 lowpan6_set_short_addr()

```
err_t lowpan6_set_short_addr (u8_t addr_high, u8_t addr_low)
```

Set short address

2.4.4.3.6 tcpip_6lowpan_input()

```
err_t tcpip_6lowpan_input (struct pbuf * p, struct netif * inp)
```

Pass a received packet to tcpip_thread for input processing

Parameters

p	the received packet, p->payload pointing to the IEEE 802.15.4 header.
inp	the network interface on which the packet was received

2.4.5 6LoWPAN over BLE (RFC7668)

2.4.5.1 Functions

- void ble_addr_to_eui64 (u8_t *dst, const u8_t *src, int public_addr)
- void eui64_to_ble_addr (u8_t *dst, const u8_t *src)
- err_t rfc7668_set_context (u8_t idx, const ip6_addr_t *context)
- err_t rfc7668_output (struct netif *netif, struct pbuf *q, const ip6_addr_t *ip6addr)
- err_t rfc7668_input (struct pbuf *p, struct netif *netif)
- err_t rfc7668_if_init (struct netif *netif)

2.4.5.2 Detailed Description

This file implements a RFC7668 implementation for 6LoWPAN over Bluetooth Low Energy. The specification is very similar to 6LoWPAN, so most of the code is re-used. Compared to 6LoWPAN, much functionality is already implemented in lower BLE layers (fragmenting, session management,...).

Usage:

- add this netif
  - don't add IPv4 addresses (no IPv4 support in RFC7668), pass 'NULL','NULL','NULL'
  - use the BLE to EUI64 conversation util to create an IPv6 link-local address from the BLE MAC (ble_addr_to_eui64)
  - input function: rfc7668_input
- set the link output function, which transmits output data to an established L2CAP channel
- If data arrives (HCI event "L2CAP_DATA_PACKET"):
  - allocate a PBUF_RAW buffer
  - let the pbuf struct point to the incoming data or copy it to the buffer
  - call netif->input

2.4.5.3 Function Documentation

2.4.5.3.1 ble_addr_to_eui64()

```
void ble_addr_to_eui64 (u8_t * dst, const u8_t * src, int public_addr)
```

convert BT address to EUI64 addr

This method converts a Bluetooth MAC address to an EUI64 address, which is used within IPv6 communication

Parameters

dst	IPv6 destination space
src	BLE MAC address source
public_addr	If the LWIP_RFC7668_LINUX_WORKAROUND_PUBLIC_ADDRESS option is set, bit 0x02 will be set if param=0 (no public addr); cleared otherwise

See also LWIP_RFC7668_LINUX_WORKAROUND_PUBLIC_ADDRESS

2.4.5.3.2 eui64_to_ble_addr()

```
void eui64_to_ble_addr (u8_t * dst, const u8_t * src)
```

convert EUI64 address to Bluetooth MAC addr

This method converts an EUI64 address to a Bluetooth MAC address,

Parameters

dst	BLE MAC address destination
src	IPv6 source

2.4.5.3.3 rfc7668_if_init()

```
err_t rfc7668_if_init (struct netif * netif)
```

Initialize the netif

No flags are used (broadcast not possible, not ethernet, ...) The shortname for this netif is "BT"

Parameters

netif	the network interface to be initialized as RFC7668 netif

Returns ERR_OK if everything went fine

2.4.5.3.4 rfc7668_input()

```
err_t rfc7668_input (struct pbuf * p, struct netif * netif)
```

Process a received raw payload from an L2CAP channel

Parameters

Returns ERR_OK if everything was fine

p	the received packet, p->payload pointing to the IPv6 header (maybe compressed)
netif	the network interface on which the packet was received

2.4.5.3.5 rfc7668_output()

`err_t rfc7668_output (struct netif * netif, struct pbuf * q, const ip6_addr_t * ip6addr)`

Compress outgoing IPv6 packet and pass it on to netif->linkoutput

Parameters

netif	The lwIP network interface which the IP packet will be sent on.
q	The pbuf(s) containing the IP packet to be sent.
ip6addr	The IP address of the packet destination.

Returns See rfc7668_compress

2.4.5.3.6 rfc7668_set_context()

`err_t rfc7668_set_context (u8_t idx, const ip6_addr_t * context)`

Set context id IPv6 address

Store one IPv6 address to a given context id.

Parameters

idx	Context id
context	IPv6 addr for this context

Returns ERR_OK (if everything is fine), ERR_ARG (if the context id is out of range), ERR_VAL (if contexts disabled)

2.4.6 PPP

```
PPP interface for lwIP

Author: Sylvain Rochet

Table of Contents:

1 - Supported PPP protocols and features
2 - Raw API PPP example for all protocols
3 - PPPoS input path (raw API, IRQ safe API, TCPIP API)
4 - Thread safe PPP API (PPPAPI)
5 - Notify phase callback (PPP_NOTIFY_PHASE)
6 - Upgrading from lwIP <= 1.4.x to lwIP >= 2.0.x



1 Supported PPP protocols and features
======================================

Supported Low level protocols:
* PPP over serial using HDLC-like framing, such as wired dialup modems
  or mobile telecommunications GPRS/EDGE/UMTS/HSPA+/LTE modems
* PPP over Ethernet, such as xDSL modems
* PPP over L2TP (Layer 2 Tunneling Protocol) LAC (L2TP Access Concentrator),
  IP tunnel over UDP, such as VPN access

Supported auth protocols:
* PAP, Password Authentication Protocol
* CHAP, Challenge-Handshake Authentication Protocol, also known as CHAP-MD5
* MSCHAPv1, Microsoft version of CHAP, version 1
* MSCHAPv2, Microsoft version of CHAP, version 2
* EAP, Extensible Authentication Protocol

Supported address protocols:
* IPCP, IP Control Protocol, IPv4 addresses negotiation
* IP6CP, IPv6 Control Protocol, IPv6 link-local addresses negotiation

Supported encryption protocols:
* MPPE, Microsoft Point-to-Point Encryption

Supported compression or miscellaneous protocols, for serial links only:
* PFC, Protocol Field Compression
* ACFC, Address-and-Control-Field-Compression
* ACCM, Asynchronous-Control-Character-Map
* VJ, Van Jacobson TCP/IP Header Compression



2 Raw API PPP example for all protocols
=======================================

As usual, raw API for lwIP means the lightweight API which *MUST* only be used
for NO_SYS=1 systems or called inside lwIP core thread for NO_SYS=0 systems.

/*
 * Globals
 * =======
 */

/* The PPP control block */
ppp_pcb *ppp;

/* The PPP IP interface */
struct netif ppp_netif;


/*
 * PPP status callback
 * ===================
 *
 * PPP status callback is called on PPP status change (up, down, ...) from lwIP
 * core thread
 */

/* PPP status callback example */
static void status_cb(ppp_pcb *pcb, int err_code, void *ctx) {
  struct netif *pppif = ppp_netif(pcb);
  LWIP_UNUSED_ARG(ctx);

  switch(err_code) {
    case PPPERR_NONE: {
#if LWIP_DNS
      const ip_addr_t *ns;
#endif /* LWIP_DNS */
      printf("status_cb: Connected\n");
```

```
#if PPP_IPV4_SUPPORT
      printf("   our_ipaddr  = %s\n", ipaddr_ntoa(&pppif->ip_addr));
      printf("   his_ipaddr  = %s\n", ipaddr_ntoa(&pppif->gw));
      printf("   netmask     = %s\n", ipaddr_ntoa(&pppif->netmask));
#if LWIP_DNS
      ns = dns_getserver(0);
      printf("   dns1        = %s\n", ipaddr_ntoa(ns));
      ns = dns_getserver(1);
      printf("   dns2        = %s\n", ipaddr_ntoa(ns));
#endif /* LWIP_DNS */
#endif /* PPP_IPV4_SUPPORT */
#if PPP_IPV6_SUPPORT
      printf("   our6_ipaddr = %s\n", ip6addr_ntoa(netif_ip6_addr(pppif, 0)));
#endif /* PPP_IPV6_SUPPORT */
      break;
    }
    case PPPERR_PARAM: {
      printf("status_cb: Invalid parameter\n");
      break;
    }
    case PPPERR_OPEN: {
      printf("status_cb: Unable to open PPP session\n");
      break;
    }
    case PPPERR_DEVICE: {
      printf("status_cb: Invalid I/O device for PPP\n");
      break;
    }
    case PPPERR_ALLOC: {
      printf("status_cb: Unable to allocate resources\n");
      break;
    }
    case PPPERR_USER: {
      printf("status_cb: User interrupt\n");
      break;
    }
    case PPPERR_CONNECT: {
      printf("status_cb: Connection lost\n");
      break;
    }
    case PPPERR_AUTHFAIL: {
      printf("status_cb: Failed authentication challenge\n");
      break;
    }
    case PPPERR_PROTOCOL: {
      printf("status_cb: Failed to meet protocol\n");
      break;
    }
    case PPPERR_PEERDEAD: {
      printf("status_cb: Connection timeout\n");
      break;
    }
    case PPPERR_IDLETIMEOUT: {
      printf("status_cb: Idle Timeout\n");
      break;
    }
    case PPPERR_CONNECTTIME: {
      printf("status_cb: Max connect time reached\n");
      break;
    }
    case PPPERR_LOOPBACK: {
      printf("status_cb: Loopback detected\n");
      break;
    }
    default: {
      printf("status_cb: Unknown error code %d\n", err_code);
      break;
    }
  }

/*
 * This should be in the switch case, this is put outside of the switch
 * case for example readability.
 */

  if (err_code == PPPERR_NONE) {
    return;
  }

  /* ppp_close() was previously called, don’t reconnect */
  if (err_code == PPPERR_USER) {
    /* ppp_free(); -- can be called here */
    return;
  }

  /*
   * Try to reconnect in 30 seconds, if you need a modem chatscript you have
   * to do a much better signaling here ;-)
   */
  ppp_connect(pcb, 30);
  /* OR ppp_listen(pcb); */
}


/*
 * Creating a new PPPoS session
 * ============================
 *
 * In lwIP, PPPoS is not PPPoSONET, in lwIP PPPoS is PPPoSerial.
 */

#include "netif/ppp/pppos.h"

/*
 * PPPoS serial output callback
 *
 * ppp_pcb, PPP control block
 * data, buffer to write to serial port
 * len, length of the data buffer
 * ctx, optional user-provided callback context pointer
 *
 * Return value: len if write succeed
 */
static u32_t output_cb(ppp_pcb *pcb, const void *data, u32_t len, void *ctx) {
  return uart_write(UART, data, len);
}

/*
 * Create a new PPPoS interface
 *
 * ppp_netif, netif to use for this PPP link, i.e. PPP IP interface
 * output_cb, PPPoS serial output callback
 * status_cb, PPP status callback, called on PPP status change (up, down, ...)
 * ctx_cb, optional user-provided callback context pointer
 */
ppp = pppos_create(&ppp_netif,
       output_cb, status_cb, ctx_cb);


/*
 * Creating a new PPPoE session
 * ============================
 */

#include "netif/ppp/pppoe.h"

/*
 * Create a new PPPoE interface
 *
 * ppp_netif, netif to use for this PPP link, i.e. PPP IP interface
 * ethif, already existing and setup Ethernet interface to use
 * service_name, PPPoE service name discriminator (not supported yet)
 * concentrator_name, PPPoE concentrator name discriminator (not supported yet)
 * status_cb, PPP status callback, called on PPP status change (up, down, ...)
 * ctx_cb, optional user-provided callback context pointer
 */
ppp = pppoe_create(&ppp_netif,
       &ethif,
       service_name, concentrator_name,
       status_cb, ctx_cb);


/*
 * Creating a new PPPoL2TP session
 * ===============================
 */

#include "netif/ppp/pppol2tp.h"

/*
 * Create a new PPPoL2TP interface
 *
 * ppp_netif, netif to use for this PPP link, i.e. PPP IP interface
 * netif, optional already existing and setup output netif, necessary if you
 *        want to set this interface as default route to settle the chicken
 *        and egg problem with VPN links
 * ipaddr, IP to connect to
 * port, UDP port to connect to (usually 1701)
 * secret, L2TP secret to use
 * secret_len, size in bytes of the L2TP secret
 * status_cb, PPP status callback, called on PPP status change (up, down, ...)
 * ctx_cb, optional user-provided callback context pointer
 */
ppp = pppol2tp_create(&ppp_netif,
       struct netif *netif, ip_addr_t *ipaddr, u16_t port,
       u8_t *secret, u8_t secret_len,
       ppp_link_status_cb_fn link_status_cb, void *ctx_cb);


/*
 * Initiate PPP client connection
 * ==============================
 */

/* Set this interface as default route */
ppp_set_default(ppp);
```

```
/*
 * Basic PPP client configuration. Can only be set if PPP session is in the
 * dead state (i.e. disconnected). We don't need to provide thread-safe
 * equivalents through PPPAPI because those helpers are only changing
 * structure members while session is inactive for lwIP core. Configuration
 * only need to be done once.
 */

/* Ask the peer for up to 2 DNS server addresses. */
ppp_set_usepeerdns(ppp, 1);

/* Auth configuration, this is pretty self-explanatory */
ppp_set_auth(ppp, PPPAUTHTYPE_ANY, "login", "password");

/*
 * Initiate PPP negotiation, without waiting (holdoff=0), can only be called
 * if PPP session is in the dead state (i.e. disconnected).
 */
u16_t holdoff = 0;
ppp_connect(ppp, holdoff);


/*
 * Initiate PPP server listener
 * ============================
 */

/*
 * Basic PPP server configuration. Can only be set if PPP session is in the
 * dead state (i.e. disconnected). We don't need to provide thread-safe
 * equivalents through PPPAPI because those helpers are only changing
 * structure members while session is inactive for lwIP core. Configuration
 * only need to be done once.
 */
ip4_addr_t addr;

/* Set our address */
IP4_ADDR(&addr, 192,168,0,1);
ppp_set_ipcp_ouraddr(ppp, &addr);

/* Set peer(his) address */
IP4_ADDR(&addr, 192,168,0,2);
ppp_set_ipcp_hisaddr(ppp, &addr);

/* Set primary DNS server */
IP4_ADDR(&addr, 192,168,10,20);
ppp_set_ipcp_dnsaddr(ppp, 0, &addr);

/* Set secondary DNS server */
IP4_ADDR(&addr, 192,168,10,21);
ppp_set_ipcp_dnsaddr(ppp, 1, &addr);

/* Auth configuration, this is pretty self-explanatory */
ppp_set_auth(ppp, PPPAUTHTYPE_ANY, "login", "password");

/* Require peer to authenticate */
ppp_set_auth_required(ppp, 1);

/*
 * Only for PPPoS, the PPP session should be up and waiting for input.
 *
 * Note: for PPPoS, ppp_connect() and ppp_listen() are actually the same thing.
 * The listen call is meant for future support of PPPoE and PPPoL2TP server
 * mode, where we will need to negotiate the incoming PPPoE session or L2TP
 * session before initiating PPP itself. We need this call because there is
 * two passive modes for PPPoS, ppp_set_passive and ppp_set_silent.
 */
ppp_set_silent(pppos, 1);

/*
 * Initiate PPP listener (i.e. wait for an incoming connection), can only
 * be called if PPP session is in the dead state (i.e. disconnected).
 */
ppp_listen(ppp);


/*
 * Closing PPP connection
 * ======================
 */

/*
 * Initiate the end of the PPP session, without carrier lost signal
 * (nocarrier=0), meaning a clean shutdown of PPP protocols.
 * You can call this function at anytime.
 */
u8_t nocarrier = 0;
ppp_close(ppp, nocarrier);
/*
 * Then you must wait your status_cb() to be called, it may takes from a few
 * seconds to several tens of seconds depending on the current PPP state.
 */

/*
 * Freeing a PPP connection
 * ========================
 */

/*
 * Free the PPP control block, can only be called if PPP session is in the
 * dead state (i.e. disconnected). You need to call ppp_close() before.
 */
ppp_free(ppp);



3 PPPoS input path (raw API, IRQ safe API, TCPIP API)
=====================================================

Received data on serial port should be sent to lwIP using the pppos_input()
function or the pppos_input_tcpip() function.

If NO_SYS is 1 and if PPP_INPROC_IRQ_SAFE is 0 (the default), pppos_input()
is not IRQ safe and then *MUST* only be called inside your main loop.

Whatever the NO_SYS value, if PPP_INPROC_IRQ_SAFE is 1, pppos_input() is IRQ
safe and can be safely called from an interrupt context, using that is going
to reduce your need of buffer if pppos_input() is called byte after byte in
your rx serial interrupt.

if NO_SYS is 0, the thread safe way outside an interrupt context is to use
the pppos_input_tcpip() function to pass input data to the lwIP core thread
using the TCPIP API. This is thread safe in all cases but you should avoid
```

```
passing data byte after byte because it uses heavy locking (mailbox) and it
allocates pbuf, better fill them !

if NO_SYS is 0 and if PPP_INPROC_IRQ_SAFE is 1, you may also use pppos_input()
from an RX thread, however pppos_input() is not thread safe by itself. You can
do that *BUT* you should NEVER call pppos_connect(), pppos_listen() and
ppp_free() if pppos_input() can still be running, doing this is NOT thread safe
at all. Using PPP_INPROC_IRQ_SAFE from an RX thread is discouraged unless you
really know what you are doing, your move ;-)


/*
 * Function to call for received data
 *
 * ppp, PPP control block
 * buffer, input buffer
 * buffer_len, buffer length in bytes
 */
void pppos_input(ppp, buffer, buffer_len);

or

void pppos_input_tcpip(ppp, buffer, buffer_len);



4 Thread safe PPP API (PPPAPI)
==============================

There is a thread safe API for all corresponding ppp_* functions, you have to
enable LWIP_PPP_API in your lwipopts.h file, then see
include/netif/ppp/pppapi.h, this is actually pretty obvious.



5 Notify phase callback (PPP_NOTIFY_PHASE)
==========================================

Notify phase callback, enabled using the PPP_NOTIFY_PHASE config option, let
you configure a callback that is called on each PPP internal state change.
This is different from the status callback which only warns you about
up(running) and down(dead) events.

Notify phase callback can be used, for example, to set a LED pattern depending
on the current phase of the PPP session. Here is a callback example which
tries to mimic what we usually see on xDSL modems while they are negotiating
the link, which should be self-explanatory:

static void ppp_notify_phase_cb(ppp_pcb *pcb, u8_t phase, void *ctx) {
  switch (phase) {

  /* Session is down (either permanently or briefly) */
  case PPP_PHASE_DEAD:
    led_set(PPP_LED, LED_OFF);
    break;

  /* We are between two sessions */
  case PPP_PHASE_HOLDOFF:
    led_set(PPP_LED, LED_SLOW_BLINK);
    break;

  /* Session just started */
  case PPP_PHASE_INITIALIZE:
    led_set(PPP_LED, LED_FAST_BLINK);
    break;

  /* Session is running */
  case PPP_PHASE_RUNNING:
    led_set(PPP_LED, LED_ON);
    break;

  default:
    break;
  }
}



6 Upgrading from lwIP <= 1.4.x to lwIP >= 2.0.x
===============================================

PPP API was fully reworked between 1.4.x and 2.0.x releases. However porting
from previous lwIP version is pretty easy:

* Previous PPP API used an integer to identify PPP sessions, we are now
  using ppp_pcb* control block, therefore all functions changed from "int ppp"
  to "ppp_pcb *ppp"

* struct netif was moved outside the PPP structure, you have to provide a netif
  for PPP interface in pppoX_create() functions

* PPP session are not started automatically after you created them anymore,
  you have to call ppp_connect(), this way you can configure the session before
  starting it.

* Previous PPP API used CamelCase, we are now using snake_case.

* Previous PPP API mixed PPPoS and PPPoE calls, this isn't the case anymore,
  PPPoS functions are now prefixed pppos_ and PPPoE functions are now prefixed
  pppoe_, common functions are now prefixed ppp_.

* New PPPERR_ error codes added, check you have all of them in your status
  callback function

* Only the following include files should now be used in user application:
  #include "netif/ppp/pppapi.h"
  #include "netif/ppp/pppos.h"
  #include "netif/ppp/pppoe.h"
  #include "netif/ppp/pppol2tp.h"

  Functions from ppp.h can be used, but you don't need to include this header
  file as it is already included by above header files.

* PPP_INPROC_OWNTHREAD was broken by design and was removed, you have to create
  your own serial rx thread

* PPP_INPROC_MULTITHREADED option was misnamed and confusing and was renamed
  PPP_INPROC_IRQ_SAFE, please read the "PPPoS input path" documentation above
  because you might have been fooled by that

* If you used tcpip_callback_with_block() on ppp_ functions you may wish to use
  the PPPAPI API instead.

* ppp_sighup and ppp_close functions were merged using an optional argument
```

```
  "nocarrier" on ppp_close.

* DNS servers are now only remotely asked if LWIP_DNS is set and if
  ppp_set_usepeerdns() is set to true, they are now automatically registered
  using the dns_setserver() function so you don't need to do that in the PPP
  callback anymore.

* PPPoS does not use the SIO API anymore, as such it now requires a serial
  output callback in place of sio_write

* PPP_MAXIDLEFLAG is now in ms instead of jiffies
```

2.4.7 SLIP

2.4.7.1 Functions

- err_t slipif_init (struct netif *netif)
- void slipif_poll (struct netif *netif)
- void slipif_process_rxqueue (struct netif *netif)
- void slipif_received_byte (struct netif *netif, u8_t data)
- void slipif_received_bytes (struct netif *netif, u8_t *data, u8_t len)

2.4.7.2 Detailed Description

This is an arch independent SLIP netif. The specific serial hooks must be provided by another file. They are sio_open, sio_read/sio_tryread and sio_send

Usage: This netif can be used in three ways:

1. For NO_SYS==0, an RX thread can be used which blocks on sio_read() until data is received.
2. In your main loop, call slipif_poll() to check for new RX bytes, completed packets are fed into netif->input().
3. Call slipif_received_byte[s]() from your serial RX ISR and slipif_process_rxqueue() from your main loop. ISR level decodes packets and puts completed packets on a queue which is fed into the stack from the main loop (needs SYS_LIGHTWEIGHT_ for pbuf_alloc to work on ISR level!).

2.4.7.3 Function Documentation

2.4.7.3.1 slipif_init()

`err_t slipif_init (struct netif * netif)`

SLIP netif initialization

Call the arch specific sio_open and remember the opened device in the state field of the netif.

Parameters

netif	the lwip network interface structure for this slipif

Returns ERR_OK if serial line could be opened, ERR_MEM if no memory could be allocated, ERR_IF is serial line couldn't be opened

Note

If netif->state is interpreted as an u8_t serial port number.

2.4.7.3.2 slipif_poll()

`void slipif_poll (struct netif * netif)`

Polls the serial device and feeds the IP layer with incoming packets.

Parameters

netif	The lwip network interface structure for this slipif

2.4.7.3.3 slipif_process_rxqueue()

`void slipif_process_rxqueue (struct netif * netif)`

Feeds the IP layer with incoming packets that were receive

Parameters

netif	The lwip network interface structure for this slipif

2.4.7.3.4 slipif_received_byte()

`void slipif_received_byte (struct netif * netif, u8_t data)`

Process a received byte, completed packets are put on a queue that is fed into IP through slipif_process_rxqueue().

This function can be called from ISR if SYS_LIGHTWEIGHT_PROT is enabled.

Parameters

netif	The lwip network interface structure for this slipif
data	received character

2.4.7.3.5 slipif_received_bytes()

`void slipif_received_bytes (struct netif * netif, u8_t * data, u8_t len)`

Process multiple received byte, completed packets are put on a queue that is fed into IP through slipif_process_rxqueue().

This function can be called from ISR if SYS_LIGHTWEIGHT_PROT is enabled.

Parameters

2.4.8 ZEP - ZigBee Encapsulation Protocol

2.4.8.1 Functions

- err_t zepif_init (struct netif *netif)

netif	The lwip network interface structure for this slipif
data	received character
len	Number of received characters

2.4.8.2 Detailed Description

A netif implementing the ZigBee Encapsulation Protocol (ZEP). This is used to tunnel 6LowPAN over UDP.

Usage (there must be a default netif before!):

```
netif_add(&zep_netif, NULL, NULL, NULL, NULL, zepif_init, tcpip_6lowpan_input);
netif_create_ip6_linklocal_address(&zep_netif, 1);
netif_set_up(&zep_netif);
netif_set_link_up(&zep_netif);
```

2.4.8.3 Function Documentation

2.4.8.3.1 zepif_init()

`err_t zepif_init (struct netif * netif)`

Set up a raw 6LowPAN netif and surround it with input- and output functions for ZEP

2.5 Applications

2.5.1 Modules

- HTTP client
- HTTP server
- Iperf server
- MDNS
- MQTT client
- NETBIOS responder
- SMTP client
- SNMPv2c/v3 agent
- SNTP
- TFTP client/server

2.5.2 Detailed Description

2.5.3 HTTP client

2.5.3.1 Macros

- #define LWIP_HTTPC_HAVE_FILE_IO 0
- #define HTTP_DEFAULT_PORT LWIP_IANA_PORT_HTTP

2.5.3.2 Typedefs

- typedef enum ehttpc_result httpc_result_t
- typedef void(* httpc_result_fn) (void *arg, httpc_result_t httpc_result, u32_t rx_content_len, u32_t srv_res, err_t err)
- typedef err_t(* httpc_headers_done_fn) (httpc_state_t *connection, void *arg, struct pbuf *hdr, u16_t hdr_len, u32_t content_len)

2.5.3.3 Enumerations

- enum ehttpc_result { HTTPC_RESULT_OK = 0 , HTTPC_RESULT_ERR_UNKNOWN = 1 , HTTPC_RESULT_ERR_CONNECT = 2 , HTTPC_RESULT_ERR_HOSTNAME = 3 , HTTPC_RESULT_ERR_CLOSED = 4 , HTTPC_RESULT_ERR_TIMEOUT = 5 , HTTPC_RESULT_ERR_SVR_RESP = 6 , HTTPC_RESULT_ERR_MEM = 7 , HTTPC_RESULT_LOCAL_ABORT = 8 , HTTPC_RESULT_ERR_CONTENT_LEN = 9 }

2.5.3.4 Functions

- err_t httpc_get_file (const ip_addr_t *server_addr, u16_t port, const char *uri, const httpc_connection_t *settings, altcp_recv_fn recv_fn, void *callback_arg, httpc_state_t **connection)
- err_t httpc_get_file_dns (const char *server_name, u16_t port, const char *uri, const httpc_connection_t *settings, altcp_recv_fn recv_fn, void *callback_arg, httpc_state_t **connection)

2.5.3.5 Detailed Description

2.5.3.6 Macro Definition Documentation

2.5.3.6.1 HTTP_DEFAULT_PORT

```
#define HTTP_DEFAULT_PORT   LWIP_IANA_PORT_HTTP
```

The default TCP port used for HTTP

2.5.3.6.2 LWIP_HTTPC_HAVE_FILE_IO

```
#define LWIP_HTTPC_HAVE_FILE_IO   0
```

HTTPC_HAVE_FILE_IO: define this to 1 to have functions downloading directly to disk via fopen/fwrite. These functions are example implementations of the interface only.

2.5.3.7 Typedef Documentation

2.5.3.7.1 httpc_headers_done_fn

```
typedef err_t(* httpc_headers_done_fn) (httpc_state_t *connection, void *arg, struct pbuf
*hdr, u16_t hdr_len, u32_t content_len)
```

Prototype of http client callback: called when the headers are received

Parameters

Returns if != ERR_OK is returned, the connection is aborted

connection	http client connection
arg	argument specified when initiating the request
hdr	header pbuf(s) (may contain data also)
hdr_len	length of the headers in 'hdr'
content_len	content length as received in the headers (-1 if not received)

arg	argument specified when initiating the request
httpc_result	result of the http transfer (see enum httpc_result_t)
rx_content_len	number of bytes received (without headers)
srv_res	this contains the http status code received (if any)
err	an error returned by internal lwip functions, can help to specify the source of the error but must not necessarily be != ERR_OK

2.5.3.7.2 httpc_result_fn

```
typedef void(* httpc_result_fn) (void *arg, httpc_result_t httpc_result, u32_t rx_content_
u32_t srv_res, err_t err)
```

Prototype of a http client callback function

Parameters

2.5.3.7.3 httpc_result_t

```
typedef enum ehttpc_result httpc_result_t
```

HTTP client result codes

2.5.3.8 Enumeration Type Documentation

2.5.3.8.1 ehttpc_result

```
enum ehttpc_result
```

HTTP client result codes

HTTPC_RESULT_OK	File successfully received
HTTPC_RESULT_ERR_UNKNOWN	Unknown error
HTTPC_RESULT_ERR_CONNECT	Connection to server failed
HTTPC_RESULT_ERR_HOSTNAME	Failed to resolve server hostname
HTTPC_RESULT_ERR_CLOSED	Connection unexpectedly closed by remote server
HTTPC_RESULT_ERR_TIMEOUT	Connection timed out (server didn't respond in time)
HTTPC_RESULT_ERR_SVR_RESP	Server responded with an error code
HTTPC_RESULT_ERR_MEM	Local memory error
HTTPC_RESULT_LOCAL_ABORT	Local abort
HTTPC_RESULT_ERR_CONTENT_LEN	Content length mismatch

2.5.3.9 Function Documentation

2.5.3.9.1 httpc_get_file()

```
err_t httpc_get_file (const ip_addr_t * server_addr, u16_t port, const char * uri, const
httpc_connection_t * settings, altcp_recv_fn recv_fn, void * callback_arg, httpc_state_t
** connection)
```

HTTP client API: get a file by passing server IP address

Parameters

server_addr	IP address of the server to connect
port	tcp port of the server
uri	uri to get from the server, remember leading "/"!
settings	connection settings (callbacks, proxy, etc.)
recv_fn	the http body (not the headers) are passed to this callback
callback_arg	argument passed to all the callbacks
connection	retrieves the connection handle (to match in callbacks)

2.5.3.9.2 httpc_get_file_dns()

```
err_t httpc_get_file_dns (const char * server_name, u16_t port, const char * uri, const
httpc_connection_t * settings, altcp_recv_fn recv_fn, void * callback_arg, httpc_state_t
** connection)
```

HTTP client API: get a file by passing server name as string (DNS name or IP address string)

Parameters

server_name	server name as string (DNS name or IP address string)
port	tcp port of the server
uri	uri to get from the server, remember leading "/"!
settings	connection settings (callbacks, proxy, etc.)
recv_fn	the http body (not the headers) are passed to this callback
callback_arg	argument passed to all the callbacks
connection	retrieves the connection handle (to match in callbacks)

Returns ERR_OK if starting the request succeeds (callback_fn will be called later) or an error code

2.5.4 HTTP server

2.5.4.1 Modules

- Options

2.5.4.2 Data Structures

- struct tCGI

2.5.4.3 Typedefs

- typedef const char *(* tCGIHandler) (int iIndex, int iNumParams, char *pcParam[], char *pcValue[])
- typedef u16_t(* tSSIHandler) (const char *ssi_tag_name, char *pcInsert, int iInsertLen)

2.5.4.4 Functions

- void httpd_post_data_recved (void *connection, u16_t recved_len)
- void httpd_init (void)
- void httpd_inits (struct altcp_tls_config *conf)
- void http_set_ssi_handler (tSSIHandler ssi_handler, const char **tags, int num_tags)
- void http_set_cgi_handlers (const tCGI *cgis, int num_handlers)
- err_t httpd_post_begin (void *connection, const char *uri, const char *http_request, u16_t http_request_len, int content_len, char *response_uri, u16_t response_uri_len, u8_t *post_auto_wnd)
- err_t httpd_post_receive_data (void *connection, struct pbuf *p)
- void httpd_post_finished (void *connection, char *response_uri, u16_t response_uri_len)

2.5.4.5 Detailed Description

This httpd supports for a rudimentary server-side-include facility which will replace tags of the form in any file whose extension is .shtml, .shtm or .ssi with strings provided by an include handler whose pointer is provided to the module via function http_set_ssi_handler(). Additionally, a simple common gateway interface (CGI) handling mechanism has been added to allow clients to hook functions to particular request URIs.

To enable SSI support, define label LWIP_HTTPD_SSI in lwipopts.h. To enable CGI support, define label LWIP_HTTPD_CGI in lwipopts.h.

By default, the server assumes that HTTP headers are already present in each file stored in the file system. By defining LWIP_HTTPD_DYNAMIC_HEADERS in lwipopts.h, this behavior can be changed such that the server inserts the headers automatically based on the extension of the file being served. If this mode is used, be careful to ensure that the file system image used does not already contain the header information.

File system images without headers can be created using the makefsfile tool with the -h command line option.

Notes about valid SSI tags

The following assumptions are made about tags used in SSI markers:

1. No tag may contain '-' or whitespace characters within the tag name.
2. Whitespace is allowed between the tag leadin "<!--#" and the start of the tag name and between the tag name and the leadout string "-->".
3. The maximum tag name length is LWIP_HTTPD_MAX_TAG_NAME_LEN, currently 8 characters.

Notes on CGI usage

The simple CGI support offered here works with GET method requests only and can handle up to 16 parameters encoded into the URI. The handler function may not write directly to the HTTP output but must return a filename that the HTTP server will send to the browser as a response to the incoming CGI request.

The list of supported file types is quite short, so if makefsdata complains about an unknown extension, make sure to add it (and its doctype) to the 'g_psHTTPHeaders' list.

2.5.4.6 Typedef Documentation

2.5.4.6.1 tCGIHandler

```
typedef const char *(* tCGIHandler) (int iIndex, int iNumParams, char *pcParam[], char
*pcValue[])
```

Function pointer for a CGI script handler.

This function is called each time the HTTPD server is asked for a file whose name was previously registered as a CGI function using a call to http_set_cgi_handlers. The iIndex parameter provides the index of the CGI within the cgis array passed to http_set_cgi_handlers. Parameters pcParam and pcValue provide access to the parameters provided along with the URI. iNumParams provides a count of the entries in the pcParam and pcValue arrays. Each entry in the pcParam array contains the name of a parameter with the corresponding entry in the pcValue array containing the value for that parameter. Note that pcParam may contain multiple elements with the same name if, for example, a multi-selection list control is used in the form generating the data.

The function should return a pointer to a character string which is the path and filename of the response that is to be sent to the connected browser, for example "/thanks.htm" or "/response/error.ssi".

The maximum number of parameters that will be passed to this function via iNumParams is defined by LWIP_HTTPD_MAX_CGI_PAR
Any parameters in the incoming HTTP request above this number will be discarded.

Requests intended for use by this CGI mechanism must be sent using the GET method (which encodes all parameters within the URI rather than in a block later in the request). Attempts to use the POST method will result in the request being ignored.

2.5.4.6.2 tSSIHandler

```
typedef u16_t(* tSSIHandler) (const char *ssi_tag_name, char *pcInsert, int iInsertLen)
```

Function pointer for the SSI tag handler callback.

This function will be called each time the HTTPD server detects a tag of the form in files with extensions mentioned in the g_pcSSIExtensions array (currently .shtml, .shtm, .ssi, .xml, .json) where "name" appears as one of the tags supplied to http_set_ssi_handler in the tags array. The returned insert string, which will be appended after the the string "<!--#name-->" in file sent back to the client, should be written to pointer pcInsert. iInsertLen contains the size of the buffer pointed to by pcInsert. The iIndex parameter provides the zero-based index of the tag as found in the tags array and identifies the tag that is to be processed.

The handler returns the number of characters written to pcInsert excluding any terminating NULL or HTTPD_SSI_TAG_UNKNOWN when tag is not recognized.

Note that the behavior of this SSI mechanism is somewhat different from the "normal" SSI processing as found in, for example, the Apache web server. In this case, the inserted text is appended following the SSI tag rather than replacing the tag entirely. This allows for an implementation that does not require significant additional buffering of output data yet which will still offer usable SSI functionality. One downside to this approach is when attempting to use SSI within JavaScript. The SSI tag is structured to resemble an HTML comment but this syntax does not constitute a comment within JavaScript and, hence, leaving the tag in place will result in problems in these cases. In order to avoid these problems, define LWIP_HTTPD_SSI_INCLUDE_TAG as zero in your lwip options file, or use JavaScript style block comments in the form / * # name * / (without the spaces).

2.5.4.7 Function Documentation

2.5.4.7.1 http_set_cgi_handlers()

```
void http_set_cgi_handlers (const tCGI * cgis, int num_handlers)
```

Set an array of CGI filenames/handler functions

Parameters

cgis	an array of CGI filenames/handler functions
num_handlers	number of elements in the ’cgis’ array

2.5.4.7.2 http_set_ssi_handler()

```
void http_set_ssi_handler (tSSIHandler ssi_handler, const char ** tags, int num_tags)
```

Set the SSI handler function.

Parameters

ssi_handler	the SSI handler function
tags	an array of SSI tag strings to search for in SSI-enabled files
num_tags	number of tags in the ’tags’ array

2.5.4.7.3 httpd_init()

```
void httpd_init (void )
```

Initialize the httpd: set up a listening PCB and bind it to the defined port

2.5.4.7.4 httpd_inits()

```
void httpd_inits (struct altcp_tls_config * conf)
```

Initialize the httpd: set up a listening PCB and bind it to the defined port. Also set up TLS connection handling (HTTPS).

2.5.4.7.5 httpd_post_begin()

```
err_t httpd_post_begin (void * connection, const char * uri, const char * http_request,
u16_t http_request_len, int content_len, char * response_uri, u16_t response_uri_len, u8_t
* post_auto_wnd)
```

Called when a POST request has been received. The application can decide whether to accept it or not.

Parameters

connection	Unique connection identifier, valid until httpd_post_end is called.
uri	The HTTP header URI receiving the POST request.
http_request	The raw HTTP request (the first packet, normally).
http_request_len	Size of 'http_request'.
content_len	Content-Length from HTTP header.
response_uri	Filename of response file, to be filled when denying the request
response_uri_len	Size of the 'response_uri' buffer.
post_auto_wnd	Set this to 0 to let the callback code handle window updates by calling 'httpd_post_data_recved' (to throttle rx speed) default is 1 (httpd handles window updates automatically)

Returns ERR_OK: Accept the POST request, data may be passed in another err_t: Deny the POST request, send back 'bad request'.

2.5.4.7.6 httpd_post_data_recved()

```
void httpd_post_data_recved (void * connection, u16_t recved_len)
```

A POST implementation can call this function to update the TCP window. This can be used to throttle data reception (e.g. when received data is programmed to flash and data is received faster than programmed).

Parameters

connection	A connection handle passed to httpd_post_begin for which httpd_post_finished has *NOT* been called yet!
recved_len	Length of data received (for window update)

2.5.4.7.7 httpd_post_finished()

```
void httpd_post_finished (void * connection, char * response_uri, u16_t response_uri_len)
```

Called when all data is received or when the connection is closed. The application must return the filename/URI of a file to send in response to this POST request. If the response_uri buffer is untouched, a 404 response is returned.

Parameters

connection	Unique connection identifier.
response_uri	Filename of response file, to be filled when denying the request
response_uri_len	Size of the 'response_uri' buffer.

connection	Unique connection identifier.
p	Received data.

2.5.4.7.8 httpd_post_receive_data()

```
err_t httpd_post_receive_data (void * connection, struct pbuf * p)
```

Called for each pbuf of data that has been received for a POST. ATTENTION: The application is responsible for freeing the pbufs passed in!

Parameters

Returns ERR_OK: Data accepted. another err_t: Data denied, http_post_get_response_uri will be called.

2.5.4.8 Options

2.5.4.8.1 Macros

- #define LWIP_HTTPD_CGI 0
- #define LWIP_HTTPD_CGI_SSI 0
- #define LWIP_HTTPD_SSI 0
- #define LWIP_HTTPD_SSI_RAW 0
- #define LWIP_HTTPD_SSI_BY_FILE_EXTENSION 1
- #define LWIP_HTTPD_SSI_EXTENSIONS ".shtml", ".shtm", ".ssi", ".xml", ".json"
- #define LWIP_HTTPD_SUPPORT_POST 0
- #define LWIP_HTTPD_SSI_MULTIPART 0
- #define HTTPD_SERVER_AGENT "lwIP/" LWIP_VERSION_STRING " (http://savannah.nongnu.org/projects/lwip)"
- #define LWIP_HTTPD_DYNAMIC_HEADERS 0
- #define HTTPD_USE_MEM_POOL 0
- #define HTTPD_SERVER_PORT LWIP_IANA_PORT_HTTP
- #define HTTPD_SERVER_PORT_HTTPS LWIP_IANA_PORT_HTTPS
- #define HTTPD_ENABLE_HTTPS 0
- #define HTTPD_MAX_RETRIES 4
- #define HTTPD_POLL_INTERVAL 4
- #define HTTPD_TCP_PRIO TCP_PRIO_MIN
- #define LWIP_HTTPD_TIMING 0
- #define HTTPD_DEBUG_TIMING LWIP_DBG_OFF
- #define LWIP_HTTPD_SUPPORT_EXTSTATUS 0
- #define LWIP_HTTPD_SUPPORT_V09 1

- #define LWIP_HTTPD_SUPPORT_11_KEEPALIVE 0
- #define LWIP_HTTPD_SUPPORT_REQUESTLIST 1
- #define LWIP_HTTPD_REQ_QUEUELEN 5
- #define LWIP_HTTPD_REQ_BUFSIZE LWIP_HTTPD_MAX_REQ_LENGTH
- #define LWIP_HTTPD_MAX_REQ_LENGTH LWIP_MIN(1023, (LWIP_HTTPD_REQ_QUEUELEN * PBUF_POOL_BUFSIZE)
- #define LWIP_HTTPD_MAX_REQUEST_URI_LEN 63
- #define LWIP_HTTPD_POST_MAX_RESPONSE_URI_LEN 63
- #define LWIP_HTTPD_SSI_INCLUDE_TAG 1
- #define LWIP_HTTPD_ABORT_ON_CLOSE_MEM_ERROR 0
- #define LWIP_HTTPD_KILL_OLD_ON_CONNECTIONS_EXCEEDED 0
- #define LWIP_HTTPD_OMIT_HEADER_FOR_EXTENSIONLESS_URI 0
- #define HTTP_IS_TAG_VOLATILE(ptr) TCP_WRITE_FLAG_COPY
- #define LWIP_HTTPD_CUSTOM_FILES 0
- #define LWIP_HTTPD_DYNAMIC_FILE_READ 0
- #define LWIP_HTTPD_FILE_STATE 0
- #define LWIP_HTTPD_FILE_EXTENSION 0
- #define HTTPD_PRECALCULATED_CHECKSUM 0
- #define LWIP_HTTPD_FS_ASYNC_READ 0
- #define HTTPD_FSDATA_FILE "fsdata.c"

2.5.4.8.2 Detailed Description

2.5.4.8.3 Macro Definition Documentation

2.5.4.8.3.1 HTTP_IS_TAG_VOLATILE

```
#define HTTP_IS_TAG_VOLATILE( ptr)   TCP_WRITE_FLAG_COPY
```

Default: Tags are sent from struct http_state and are therefore volatile

2.5.4.8.3.2 HTTPD_DEBUG_TIMING

```
#define HTTPD_DEBUG_TIMING   LWIP_DBG_OFF
```

Set this to 1 to enable timing each file sent

2.5.4.8.3.3 HTTPD_ENABLE_HTTPS

```
#define HTTPD_ENABLE_HTTPS   0
```

Enable https support?

2.5.4.8.3.4 HTTPD_FSDATA_FILE

```
#define HTTPD_FSDATA_FILE   "fsdata.c"
```

Filename (including path) to use as FS data file

2.5.4.8.3.5 HTTPD_MAX_RETRIES

```
#define HTTPD_MAX_RETRIES   4
```

Maximum retries before the connection is aborted/closed.

- number of times pcb->poll is called -> default is 4*500ms = 2s;
- reset when pcb->sent is called

2.5.4.8.3.6 HTTPD_POLL_INTERVAL

```
#define HTTPD_POLL_INTERVAL   4
```

The poll delay is X*500ms

2.5.4.8.3.7 HTTPD_PRECALCULATED_CHECKSUM

```
#define HTTPD_PRECALCULATED_CHECKSUM   0
```

HTTPD_PRECALCULATED_CHECKSUM==1: include precompiled checksums for predefined (MSS-sized) chunks of the files to prevent having to calculate the checksums at runtime.

2.5.4.8.3.8 HTTPD_SERVER_AGENT

```
#define HTTPD_SERVER_AGENT   "lwIP/" LWIP_VERSION_STRING " (http://savannah.nongnu.org/pro
```

This string is passed in the HTTP header as "Server: "

2.5.4.8.3.9 HTTPD_SERVER_PORT

```
#define HTTPD_SERVER_PORT   LWIP_IANA_PORT_HTTP
```

The server port for HTTPD to use

2.5.4.8.3.10 HTTPD_SERVER_PORT_HTTPS

```
#define HTTPD_SERVER_PORT_HTTPS   LWIP_IANA_PORT_HTTPS
```

The https server port for HTTPD to use

2.5.4.8.3.11 HTTPD_TCP_PRIO

```
#define HTTPD_TCP_PRIO   TCP_PRIO_MIN
```

Priority for tcp pcbs created by HTTPD (very low by default). Lower priorities get killed first when running out of memory.

2.5.4.8.3.12 HTTPD_USE_MEM_POOL

```
#define HTTPD_USE_MEM_POOL   0
```

Set this to 1 to use a memp pool for allocating struct http_state instead of the heap. If enabled, you'll need to define MEMP_NUM_PARAI
(and MEMP_NUM_PARALLEL_HTTPD_SSI_CONNS for SSI) to set the size of the pool(s).

2.5.4.8.3.13 LWIP_HTTPD_ABORT_ON_CLOSE_MEM_ERROR

```
#define LWIP_HTTPD_ABORT_ON_CLOSE_MEM_ERROR   0
```

Set this to 1 to call tcp_abort when tcp_close fails with memory error. This can be used to prevent consuming all memory in situations where the HTTP server has low priority compared to other communication.

2.5.4.8.3.14 LWIP_HTTPD_CGI

```
#define LWIP_HTTPD_CGI   0
```

Set this to 1 to support CGI (old style).

This old style CGI support works by registering an array of URLs and associated CGI handler functions (http_set_cgi_handlers). This list is scanned just before fs_open is called from request handling. The handler can return a new URL that is used internally by the httpd to load the returned page (passed to fs_open).

Use this CGI type e.g. to execute specific actions and return a page that does not depend on the CGI parameters.

2.5.4.8.3.15 LWIP_HTTPD_CGI_SSI

```
#define LWIP_HTTPD_CGI_SSI   0
```

Set this to 1 to support CGI (new style).

This new style CGI support works by calling a global function (tCGIHandler) for all URLs that are found. fs_open is called first and the URL can not be written by the CGI handler. Instead, this handler gets passed the http file state, an object where it can store information derived from the CGI URL or parameters. This file state is later passed to SSI, so the SSI code can return data depending on CGI input.

Use this CGI handler if you want CGI information passed on to SSI.

2.5.4.8.3.16 LWIP_HTTPD_CUSTOM_FILES

```
#define LWIP_HTTPD_CUSTOM_FILES   0
```

Set this to 1 and provide the functions:

- "int fs_open_custom(struct fs_file *file, const char *name)" Called first for every opened file to allow opening files that are not included in fsdata(_custom).c
- "void fs_close_custom(struct fs_file *file)" Called to free resources allocated by fs_open_custom().

2.5.4.8.3.17 LWIP_HTTPD_DYNAMIC_FILE_READ

```
#define LWIP_HTTPD_DYNAMIC_FILE_READ   0
```

Set this to 1 to support fs_read() to dynamically read file data. Without this (default=off), only one-block files are supported, and the contents must be ready after fs_open().

2.5.4.8.3.18 LWIP_HTTPD_DYNAMIC_HEADERS

```
#define LWIP_HTTPD_DYNAMIC_HEADERS   0
```

Set this to 1 if you want to include code that creates HTTP headers at runtime. Default is off: HTTP headers are then created statically by the makefsdata tool. Static headers mean smaller code size, but the (readonly) fsdata will grow a bit as every file includes the HTTP header.

2.5.4.8.3.19 LWIP_HTTPD_FILE_EXTENSION

```
#define LWIP_HTTPD_FILE_EXTENSION   0
```

Set this to 1 to add the pextension field to the fs_file structure. This is included here to retain compatibility with legacy code that relies on the presence of the pextension field. New code should use LWIP_HTTPD_FILE_STATE instead. This option may be removed in a future version of lwip.

2.5.4.8.3.20 LWIP_HTTPD_FILE_STATE

```
#define LWIP_HTTPD_FILE_STATE   0
```

Set this to 1 to include an application state argument per file that is opened. This allows to keep a state per connection/file.

2.5.4.8.3.21 LWIP_HTTPD_FS_ASYNC_READ

```
#define LWIP_HTTPD_FS_ASYNC_READ   0
```

LWIP_HTTPD_FS_ASYNC_READ==1: support asynchronous read operations (fs_read_async returns FS_READ_DELAYED and calls a callback when finished).

2.5.4.8.3.22 LWIP_HTTPD_KILL_OLD_ON_CONNECTIONS_EXCEEDED

```
#define LWIP_HTTPD_KILL_OLD_ON_CONNECTIONS_EXCEEDED   0
```

Set this to 1 to kill the oldest connection when running out of memory for 'struct http_state' or 'struct http_ssi_state'. ATTENTION: This puts all connections on a linked list, so may be kind of slow.

2.5.4.8.3.23 LWIP_HTTPD_MAX_REQ_LENGTH

```
#define LWIP_HTTPD_MAX_REQ_LENGTH   LWIP_MIN(1023, (LWIP_HTTPD_REQ_QUEUELEN * PBUF_POOL_BU
```

Defines the maximum length of a HTTP request line (up to the first CRLF, copied from pbuf into this a global buffer when pbuf- or packet-queues are received - otherwise the input pbuf is used directly)

2.5.4.8.3.24 LWIP_HTTPD_MAX_REQUEST_URI_LEN

```
#define LWIP_HTTPD_MAX_REQUEST_URI_LEN   63
```

This is the size of a static buffer used when URIs end with '/'. In this buffer, the directory requested is concatenated with all the configured default file names. Set to 0 to disable checking default filenames on non-root directories.

2.5.4.8.3.25 LWIP_HTTPD_OMIT_HEADER_FOR_EXTENSIONLESS_URI

```
#define LWIP_HTTPD_OMIT_HEADER_FOR_EXTENSIONLESS_URI   0
```

Set this to 1 to send URIs without extension without headers (who uses this at all??)

2.5.4.8.3.26 LWIP_HTTPD_POST_MAX_RESPONSE_URI_LEN

```
#define LWIP_HTTPD_POST_MAX_RESPONSE_URI_LEN   63
```

Maximum length of the filename to send as response to a POST request, filled in by the application when a POST is finished.

2.5.4.8.3.27 LWIP_HTTPD_REQ_BUFSIZE

```
#define LWIP_HTTPD_REQ_BUFSIZE   LWIP_HTTPD_MAX_REQ_LENGTH
```

Number of (TCP payload-) bytes (in pbufs) to enqueue to parse and incoming request (up to the first double-newline)

2.5.4.8.3.28 LWIP_HTTPD_REQ_QUEUELEN

```
#define LWIP_HTTPD_REQ_QUEUELEN   5
```

Number of rx pbufs to enqueue to parse an incoming request (up to the first newline)

2.5.4.8.3.29 LWIP_HTTPD_SSI

```
#define LWIP_HTTPD_SSI   0
```

Set this to 1 to support SSI (Server-Side-Includes)

In contrast to other http servers, this only calls a preregistered callback function (**See also** http_set_ssi_handler) for each tag (in the format of) encountered in SSI-enabled pages. SSI-enabled pages must have one of the predefined SSI-enabled file extensions. All files with one of these extensions are parsed when sent.

A downside of the current SSI implementation is that persistent connections don't work, as the file length is not known in advance (and httpd currently relies on the Content-Length header for persistent connections).

To save memory, the maximum tag length is limited (**See also** LWIP_HTTPD_MAX_TAG_NAME_LEN). To save memory, the maximum insertion string length is limited (

LWIP_HTTPD_MAX_TAG_INSERT_LEN). If this is not enough, LWIP_HTTPD_SSI_MULTIPART can be used.

2.5.4.8.3.30 LWIP_HTTPD_SSI_BY_FILE_EXTENSION

```
#define LWIP_HTTPD_SSI_BY_FILE_EXTENSION   1
```

Set this to 0 to prevent parsing the file extension at runtime to decide if a file should be scanned for SSI tags or not. Default is 1 (file extensions are checked using the g_pcSSIExtensions array) Set to 2 to override this runtime test function. In this case, you have to provide an external function that does the check: u8_t http_uri_is_ssi(struct fs_file *file, const char *uri)

This is enabled by default, but if you only use a newer version of makefsdata supporting the "-ssi" option, this info is already present in

2.5.4.8.3.31 LWIP_HTTPD_SSI_EXTENSIONS

```
#define LWIP_HTTPD_SSI_EXTENSIONS   ".shtml", ".shtm", ".ssi", ".xml", ".json"
```

This is a list of file extensions handled as SSI files. This define is used to initialize a 'const char *const[]'. It is only used if LWIP_HTTPD_SSI_BY_FILE_EXTENSION != 0.

2.5.4.8.3.32 LWIP_HTTPD_SSI_INCLUDE_TAG

```
#define LWIP_HTTPD_SSI_INCLUDE_TAG   1
```

Set this to 0 to not send the SSI tag (default is on, so the tag will be sent in the HTML page

2.5.4.8.3.33 LWIP_HTTPD_SSI_MULTIPART

```
#define LWIP_HTTPD_SSI_MULTIPART   0
```

LWIP_HTTPD_SSI_MULTIPART==1: SSI handler function is called with 2 more arguments indicating a counter for insert string that are too long to be inserted at once: the SSI handler function must then set 'next_tag_part' which will be passed back to it in the next call.

2.5.4.8.3.34 LWIP_HTTPD_SSI_RAW

```
#define LWIP_HTTPD_SSI_RAW   0
```

Set this to 1 to implement an SSI tag handler callback that gets a const char* to the tag (instead of an index into a pre-registered array of known tags) If this is 0, the SSI handler callback function is only called pre-registered tags.

2.5.4.8.3.35 LWIP_HTTPD_SUPPORT_11_KEEPALIVE

```
#define LWIP_HTTPD_SUPPORT_11_KEEPALIVE   0
```

Set this to 1 to enable HTTP/1.1 persistent connections. ATTENTION: If the generated file system includes HTTP headers, these must include the "Connection: keep-alive" header (pass argument "-11" to makefsdata).

2.5.4.8.3.36 LWIP_HTTPD_SUPPORT_EXTSTATUS

```
#define LWIP_HTTPD_SUPPORT_EXTSTATUS   0
```

Set this to one to show error pages when parsing a request fails instead of simply closing the connection.

2.5.4.8.3.37 LWIP_HTTPD_SUPPORT_POST

```
#define LWIP_HTTPD_SUPPORT_POST   0
```

Set this to 1 to support HTTP POST

2.5.4.8.3.38 LWIP_HTTPD_SUPPORT_REQUESTLIST

```
#define LWIP_HTTPD_SUPPORT_REQUESTLIST   1
```

Set this to 1 to support HTTP request coming in in multiple packets/pbufs

2.5.4.8.3.39 LWIP_HTTPD_SUPPORT_V09

```
#define LWIP_HTTPD_SUPPORT_V09   1
```

Set this to 0 to drop support for HTTP/0.9 clients (to save some bytes)

2.5.4.8.3.40 LWIP_HTTPD_TIMING

```
#define LWIP_HTTPD_TIMING   0
```

Set this to 1 to enable timing each file sent

2.5.5 Iperf server

2.5.5.1 Functions

- void * lwiperf_start_tcp_server_default (lwiperf_report_fn report_fn, void *report_arg)
- void * lwiperf_start_tcp_server (const ip_addr_t *local_addr, u16_t local_port, lwiperf_report_fn report_fn, void *report_arg)
- void * lwiperf_start_tcp_client_default (const ip_addr_t *remote_addr, lwiperf_report_fn report_fn, void *report_arg)
- void * lwiperf_start_tcp_client (const ip_addr_t *remote_addr, u16_t remote_port, enum lwiperf_client_type type, lwiperf_report_fn report_fn, void *report_arg)
- void lwiperf_abort (void *lwiperf_session)

2.5.5.2 Detailed Description

This is a simple performance measuring client/server to check your bandwidth using iPerf2 on a PC as server/client. It is currently a minimal implementation providing a TCP client/server only.

2.5.5.3 Function Documentation

2.5.5.3.1 lwiperf_abort()

```
void lwiperf_abort (void * lwiperf_session)
```

Abort an iperf session (handle returned by lwiperf_start_tcp_server*())

2.5.5.3.2 lwiperf_start_tcp_client()

```
void * lwiperf_start_tcp_client (const ip_addr_t * remote_addr, u16_t remote_port, enum
lwiperf_client_type type, lwiperf_report_fn report_fn, void * report_arg)
```

Start a TCP iperf client to a specific IP address and port.

Returns a connection handle that can be used to abort the client by calling lwiperf_abort()

2.5.5.3.3 lwiperf_start_tcp_client_default()

```
void * lwiperf_start_tcp_client_default (const ip_addr_t * remote_addr, lwiperf_report_fn
report_fn, void * report_arg)
```

Start a TCP iperf client to the default TCP port (5001).

Returns a connection handle that can be used to abort the client by calling lwiperf_abort()

2.5.5.3.4 lwiperf_start_tcp_server()

```
void * lwiperf_start_tcp_server (const ip_addr_t * local_addr, u16_t local_port, lwiperf_r
report_fn, void * report_arg)
```

Start a TCP iperf server on a specific IP address and port and listen for incoming connections from iperf clients.

Returns a connection handle that can be used to abort the server by calling lwiperf_abort()

2.5.5.3.5 lwiperf_start_tcp_server_default()

```
void * lwiperf_start_tcp_server_default (lwiperf_report_fn report_fn, void * report_arg)
```

Start a TCP iperf server on the default TCP port (5001) and listen for incoming connections from iperf clients.

Returns a connection handle that can be used to abort the server by calling lwiperf_abort()

2.5.6 MDNS

2.5.6.1 Modules

- Options

2.5.6.2 Macros

- #define mdns_resp_netif_settings_changed(netif)   mdns_resp_announce(netif)

2.5.6.3 Functions

- err_t mdns_resp_add_netif (struct netif *netif, const char *hostname)
- err_t mdns_resp_remove_netif (struct netif *netif)
- err_t mdns_resp_rename_netif (struct netif *netif, const char *hostname)
- int mdns_resp_netif_active (struct netif *netif)
- s8_t mdns_resp_add_service (struct netif *netif, const char *name, const char *service, enum mdns_sd_proto proto, u16_t port, service_get_txt_fn_t txt_fn, void *txt_data)
- err_t mdns_resp_del_service (struct netif *netif, u8_t slot)

- err_t mdns_resp_rename_service (struct netif *netif, u8_t slot, const char *name)
- err_t mdns_resp_add_service_txtitem (struct mdns_service *service, const char *txt, u8_t txt_len)
- void mdns_search_stop (u8_t request_id)
- err_t mdns_search_service (const char *name, const char *service, enum mdns_sd_proto proto, struct netif *netif, search_result_fn_t result_fn, void *arg, u8_t *request_id)
- void mdns_resp_announce (struct netif *netif)
- void mdns_resp_restart_delay (struct netif *netif, uint32_t delay)
- void mdns_resp_restart (struct netif *netif)
- void mdns_resp_init (void)
- void * mdns_get_service_txt_userdata (struct netif *netif, s8_t slot)

2.5.6.4 Detailed Description

RFC 6762 - Multicast DNS RFC 6763 - DNS-Based Service Discovery

You need to increase MEMP_NUM_SYS_TIMEOUT by one if you use MDNS!

```
Multicast DNS for lwIP

Author: Erik Ekman


Note! The MDNS responder does not have all features required by the standards.
See notes in src/apps/mdns/mdns.c for what is left. It is however usable in normal
cases - but watch out if many devices on the same network try to use the same
host/service instance names.


How to enable:
==============

MDNS support does not depend on DNS.
MDNS supports using IPv4 only, v6 only, or v4+v6.

To enable MDNS responder, set
  LWIP_MDNS_RESPONDER = 1
in lwipopts.h and add src/apps/mdns/mdns.c to your list of files to build.

The max number of services supported per netif is defined by MDNS_MAX_SERVICES,
default is 1.

Increase MEMP_NUM_UDP_PCB by 1. MDNS needs one PCB.
Increase LWIP_NUM_NETIF_CLIENT_DATA by 1 (MDNS needs one entry on netif).

MDNS with IPv4 requires LWIP_IGMP = 1, and preferably LWIP_AUTOIP = 1.
MDNS with IPv6 requires LWIP_IPV6_MLD = 1, and that a link-local address is
generated.

The MDNS code puts its structs on the stack where suitable to reduce dynamic
memory allocation. It may use up to 1kB of stack.
```

```
MDNS (like other apps) needs a strncasecmp() implementation. If you have one,  ↩
    define
'lwip_strnicmp' to it. Otherwise the code will provide an implementation
for you.


How to use:
===========

Call mdns_resp_init() during system initialization.
This opens UDP sockets on port 5353 for IPv4 and IPv6.


To start responding on a netif, run
  mdns_resp_add_netif(struct netif *netif, const char *hostname)

The hostname will be copied. If this returns successfully, the netif will join
the multicast groups and any MDNS/legacy DNS requests sent unicast or multicast
to port 5353 will be handled:
- <hostname>.local type A, AAAA or ANY returns relevant IP addresses
- Reverse lookups (PTR in-addr.arpa, ip6.arpa) of netif addresses
  returns <hostname>.local
MDNS allows UTF-8 names, but it is recommended to stay within ASCII,
since the default case-insensitive comparison assumes this.

Call mdns_resp_announce() every time the IP address on the netif has changed.

Call mdns_resp_restart() every time the network interface comes up after being
down, for example cable connected after being disconnected, administrative
interface comes up after being down, or the device wakes up from sleep.

To stop responding on a netif, run
  mdns_resp_remove_netif(struct netif *netif)


Adding services:
================

The netif first needs to be registered. Then run
  mdns_resp_add_service(struct netif *netif, const char *name, const char *service ↩
      ,
      enum mdns_sd_proto proto, u16_t port,
      service_get_txt_fn_t txt_fn, void *txt_userdata);

The name and service pointers will be copied. Name refers to the name of the
service instance, and service is the type of service, like _http
proto can be DNSSD_PROTO_UDP or DNSSD_PROTO_TCP which represent _udp and _tcp.
If this call returns successfully, the following queries will be answered:
- _services._dns-sd._udp.local type PTR returns <service>.<proto>.local
- <service>.<proto>.local type PTR returns <name>.<service>.<proto>.local
- <name>.<service>.<proto>.local type SRV returns hostname and port of service
- <name>.<service>.<proto>.local type TXT builds text strings by calling txt_fn
  with the supplied userdata. The callback adds strings to the reply by calling
  mdns_resp_add_service_txtitem(struct mdns_service *service, char *txt,
   int txt_len). Example callback method:

   static void srv_txt(struct mdns_service *service, void *txt_userdata)
   {
     res = mdns_resp_add_service_txtitem(service, "path=/", 6);
     LWIP_ERROR("mdns add service txt failed\n", (res == ERR_OK), return);
   }

  Since a hostname struct is used for TXT storage each single item can be max
  63 bytes long, and  the total max length (including length bytes for each
  item) is 255 bytes.

If your device runs a webserver on port 80, an example call might be:

  mdns_resp_add_service(netif, "myweb", "_http"
      DNSSD_PROTO_TCP, 80, srv_txt, NULL);

which will publish myweb._http._tcp.local for any hosts looking for web servers,
and point them to <hostname>.local:80

Relevant information will be sent as additional records to reduce number of
requests required from a client.

To remove a service from a netif, run
  mdns_resp_del_service(struct netif *netif, u8_t slot)
```

Things left to implement:

- Sending goodbye messages (zero ttl) - shutdown, DHCP lease about to expire, DHCP turned off...
- Sending negative responses NSEC
- Fragmenting replies if required
- Individual known answer detection for all local IPv6 addresses
- Dynamic size of outgoing packet

2.5.6.5 Macro Definition Documentation

2.5.6.5.1 mdns_resp_netif_settings_changed

`#define mdns_resp_netif_settings_changed( netif)   mdns_resp_announce(netif)`

Announce IP settings have changed on netif. Call this in your callback registered by netif_set_status_callback(). No need to call this function when LWIP_NETIF_EXT_STATUS_CALLBACK==1, this handled automatically for you.

Parameters

netif	The network interface where settings have changed.

2.5.6.6 Function Documentation

2.5.6.6.1 mdns_get_service_txt_userdata()

`void * mdns_get_service_txt_userdata (struct netif * netif, s8_t slot)`

Return TXT userdata of a specific service on a network interface.

Parameters

netif	Network interface.
slot	Service index.

2.5.6.6.2 mdns_resp_add_netif()

```
err_t mdns_resp_add_netif (struct netif * netif, const char * hostname)
```

Activate MDNS responder for a network interface.

Parameters

netif	The network interface to activate.
hostname	Name to use. Queries for <hostname>.local will be answered with the IP addresses of the netif. The hostname will be copied, the given pointer can be on the stack.

Returns ERR_OK if netif was added, an err_t otherwise

2.5.6.6.3 mdns_resp_add_service()

```
s8_t mdns_resp_add_service (struct netif * netif, const char * name, const char * service,
enum mdns_sd_proto proto, u16_t port, service_get_txt_fn_t txt_fn, void * txt_data)
```

Add a service to the selected network interface.

Parameters

netif	The network interface to publish this service on
name	The name of the service
service	The service type, like "_http"
proto	The service protocol, DNSSD_PROTO_TCP for TCP ("_tcp") and DNSSD_PROTO_UDP for others ("_udp")
port	The port the service listens to
txt_fn	Callback to get TXT data. Will be called each time a TXT reply is created to allow dynamic replies.
txt_data	Userdata pointer for txt_fn

Returns service_id if the service was added to the netif, an err_t otherwise

2.5.6.6.4 mdns_resp_add_service_txtitem()

```
err_t mdns_resp_add_service_txtitem (struct mdns_service * service, const char * txt, u8_t
txt_len)
```

Call this function from inside the service_get_txt_fn_t callback to add text data. Buffer for TXT data is 256 bytes, and each field is prefixed with a length byte.

Parameters

Returns ERR_OK if the string was added to the reply, an err_t otherwise

2.5.6.6.5 mdns_resp_announce()

```
void mdns_resp_announce (struct netif * netif)
```

Send unsolicited answer containing all our known data

Parameters

service	The service provided to the get_txt callback
txt	String to add to the TXT field.
txt_len	Length of string

netif	The network interface to send on

2.5.6.6.6 mdns_resp_del_service()

`err_t mdns_resp_del_service (struct netif * netif, u8_t slot)`

Delete a service on the selected network interface.

Parameters

netif	The network interface on which service should be removed
slot	The service slot number returned by mdns_resp_add_service

Returns ERR_OK if the service was removed from the netif, an err_t otherwise

2.5.6.6.7 mdns_resp_init()

`void mdns_resp_init (void )`

Initiate MDNS responder. Will open UDP sockets on port 5353

2.5.6.6.8 mdns_resp_netif_active()

`int mdns_resp_netif_active (struct netif * netif)`

Checks if an MDNS responder is active for a given network interface.

Parameters

Returns nonzero if responder active, zero otherwise.

2.5.6.6.9 mdns_resp_remove_netif()

`err_t mdns_resp_remove_netif (struct netif * netif)`

Stop responding to MDNS queries on this interface, leave multicast groups, and free the helper structure and any of its services.

Parameters

Returns ERR_OK if netif was removed, an err_t otherwise

2.5.6.6.10 mdns_resp_rename_netif()

`err_t mdns_resp_rename_netif (struct netif * netif, const char * hostname)`

Update MDNS hostname for a network interface.

Parameters

Returns ERR_OK if name could be set on netif, an err_t otherwise

netif	The network interface to test.

netif	The network interface to remove.

2.5.6.6.11 mdns_resp_rename_service()

```
err_t mdns_resp_rename_service (struct netif * netif, u8_t slot, const char * name)
```

Update name for an MDNS service.

Parameters

Returns ERR_OK if name could be set on service, an err_t otherwise

2.5.6.6.12 mdns_resp_restart()

```
void mdns_resp_restart (struct netif * netif)
```

Restart mdns responder. Call this when cable is connected after being disconnected or administrative interface is set up after being down

Parameters

2.5.6.6.13 mdns_resp_restart_delay()

```
void mdns_resp_restart_delay (struct netif * netif, uint32_t delay)
```

Restart mdns responder after a specified delay. Call this when cable is connected after being disconnected or administrative interface is set up after being down

Parameters

2.5.6.6.14 mdns_search_service()

```
err_t mdns_search_service (const char * name, const char * service, enum mdns_sd_proto
proto, struct netif * netif, search_result_fn_t result_fn, void * arg, u8_t * request_id)
```

Search a specific service on the network.

Parameters

Returns ERR_OK if the search request was created and sent, an err_t otherwise

2.5.6.6.15 mdns_search_stop()

```
void mdns_search_stop (u8_t request_id)
```

Stop a search request.

Parameters

netif	The network interface to activate.
hostname	Name to use. Queries for <hostname>.local will be answered with the IP addresses of the netif. The hostname will be copied, the given pointer can be on the stack.

netif	The network interface to activate.
slot	The service slot number returned by mdns_resp_add_service
name	The new name for the service

netif	The network interface to send on

2.5.6.7 Options

2.5.6.7.1 Macros

- #define MDNS_MAX_SERVICES   1
- #define MDNS_PROBE_DELAY_MS   250
- #define MDNS_MAX_STORED_PKTS   4
- #define MDNS_OUTPUT_PACKET_SIZE   ((MDNS_MAX_SERVICES == 1) ? 512 : 1450)
- #define MDNS_RESP_USENETIF_EXTCALLBACK LWIP_NETIF_EXT_STATUS_CALLBACK
- #define LWIP_MDNS_SEARCH   1
- #define MDNS_MAX_REQUESTS   2
- #define MDNS_DEBUG LWIP_DBG_OFF

2.5.6.7.2 Detailed Description

2.5.6.7.3 Macro Definition Documentation

2.5.6.7.3.1 LWIP_MDNS_SEARCH

```
#define LWIP_MDNS_SEARCH   1
```

LWIP_MDNS_SEARCH==1: Turn on search over multicast DNS module.

2.5.6.7.3.2 MDNS_DEBUG

```
#define MDNS_DEBUG   LWIP_DBG_OFF
```

MDNS_DEBUG: Enable debugging for multicast DNS.

2.5.6.7.3.3 MDNS_MAX_REQUESTS

```
#define MDNS_MAX_REQUESTS   2
```

The maximum number of running requests

2.5.6.7.3.4 MDNS_MAX_SERVICES

```
#define MDNS_MAX_SERVICES   1
```

LWIP_MDNS_RESPONDER==1: Turn on multicast DNS module. UDP must be available for MDNS transport. IGMP is needed for IPv4 multicast. The maximum number of services per netif

netif	The network interface to send on
delay	The delay to use before sending probe

name	The name of the service
service	The service type, like "_http"
proto	The service protocol, DNSSD_PROTO_TCP for TCP ("_tcp") and DNSSD_PROTO_UDP for others ("_udp")
netif	The network interface where to send search request
result_fn	Callback to send answer received. Will be called for each answer of a response frame matching request sent.
arg	Userdata pointer for result_fn
request_id	Returned request identifier to allow stop it.

request_id	The search request to stop

2.5.6.7.3.5 MDNS_MAX_STORED_PKTS

```
#define MDNS_MAX_STORED_PKTS   4
```

The maximum number of received packets stored in chained list of known answers for pending truncated questions. This value define the size of the MDNS_PKTS mempool. Up to MDNS_MAX_STORED_PKTS pbuf can be stored in addition to TC questions that are pending.

2.5.6.7.3.6 MDNS_OUTPUT_PACKET_SIZE

```
#define MDNS_OUTPUT_PACKET_SIZE   ((MDNS_MAX_SERVICES == 1) ?  512 :  1450)
```

Payload size allocated for each outgoing UDP packet. Will be allocated with PBUF_RAM and freed after packet was sent. According to RFC 6762, there is no reason to retain the 512 bytes restriction for link-local multicast packet. 512 bytes isn't enough when 2 services need to be probed.

2.5.6.7.3.7 MDNS_PROBE_DELAY_MS

```
#define MDNS_PROBE_DELAY_MS   250
```

The minimum delay between probes in ms. RFC 6762 require 250ms. In noisy WiFi environment, adding 30-50ms to this value help a lot for a successful Apple BCT tests.

2.5.6.7.3.8 MDNS_RESP_USENETIF_EXTCALLBACK

```
#define MDNS_RESP_USENETIF_EXTCALLBACK   LWIP_NETIF_EXT_STATUS_CALLBACK
```

MDNS_RESP_USENETIF_EXTCALLBACK==1: register an ext_callback on the netif to automatically restart probing/announcing on status or address change.

2.5.7 MQTT client

2.5.7.1 Modules

- Options

2.5.7.2 Data Structures

- struct mqtt_connect_client_info_t

2.5.7.3 Macros

- #define MQTT_PORT LWIP_IANA_PORT_MQTT
- #define MQTT_TLS_PORT LWIP_IANA_PORT_SECURE_MQTT
- #define mqtt_subscribe(client, topic, qos, cb, arg)   mqtt_sub_unsub(client, topic, qos, cb, arg, 1)
- #define mqtt_unsubscribe(client, topic, cb, arg)   mqtt_sub_unsub(client, topic, 0, cb, arg, 0)

2.5.7.4 Typedefs

- typedef void(* mqtt_connection_cb_t) (mqtt_client_t *client, void *arg, mqtt_connection_status_t status)
- typedef void(* mqtt_incoming_data_cb_t) (void *arg, const u8_t *data, u16_t len, u8_t flags)
- typedef void(* mqtt_incoming_publish_cb_t) (void *arg, const char *topic, u32_t tot_len)
- typedef void(* mqtt_request_cb_t) (void *arg, err_t err)

2.5.7.5 Enumerations

- enum mqtt_connection_status_t { MQTT_CONNECT_ACCEPTED = 0 , MQTT_CONNECT_REFUSED_PROTOCOL_VERSION = 1 , MQTT_CONNECT_REFUSED_IDENTIFIER = 2 , MQTT_CONNECT_REFUSED_SERVER = 3 , MQTT_CONNECT_REFU
  = 4 , MQTT_CONNECT_REFUSED_NOT_AUTHORIZED_ = 5 , MQTT_CONNECT_DISCONNECTED = 256 , MQTT_CONNE
  = 257 }
- enum { MQTT_DATA_FLAG_LAST = 1 }

2.5.7.6 Functions

- err_t mqtt_publish (mqtt_client_t *client, const char *topic, const void *payload, u16_t payload_length, u8_t qos, u8_t retain, mqtt_request_cb_t cb, void *arg)
- err_t mqtt_sub_unsub (mqtt_client_t *client, const char *topic, u8_t qos, mqtt_request_cb_t cb, void *arg, u8_t sub)
- void mqtt_set_inpub_callback (mqtt_client_t *client, mqtt_incoming_publish_cb_t pub_cb, mqtt_incoming_data_cb_t data_cb, void *arg)
- mqtt_client_t * mqtt_client_new (void)
- void mqtt_client_free (mqtt_client_t *client)
- err_t mqtt_client_connect (mqtt_client_t *client, const ip_addr_t *ip_addr, u16_t port, mqtt_connection_cb_t cb, void *arg, const struct mqtt_connect_client_info_t *client_info)
- void mqtt_disconnect (mqtt_client_t *client)
- u8_t mqtt_client_is_connected (mqtt_client_t *client)

2.5.7.7 Detailed Description

```
MQTT client for lwIP

Author: Erik Andersson

Details of the MQTT protocol can be found at:
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
```

```
-----------------------------------------------------------------
1. Initial steps, reserve memory and make connection to server:

You need to increase MEMP_NUM_SYS_TIMEOUT by one if you use MQTT!

1.1: Provide storage

Static allocation:
  mqtt_client_t static_client;
  example_do_connect(&static_client);

Dynamic allocation:
  mqtt_client_t *client = mqtt_client_new();
  if(client != NULL) {
    example_do_connect(&client);
  }

1.2: Establish Connection with server

void example_do_connect(mqtt_client_t *client)
{
  struct mqtt_connect_client_info_t ci;
  err_t err;

  /* Setup an empty client info structure */
  memset(&ci, 0, sizeof(ci));

  /* Minimal amount of information required is client identifier, so set it here  ←
     */
  ci.client_id = "lwip_test";

  /* Initiate client and connect to server, if this fails immediately an error  ←
      code is returned
     otherwise mqtt_connection_cb will be called with connection result after  ←
         attempting
     to establish a connection with the server.
     For now MQTT version 3.1.1 is always used */

  err = mqtt_client_connect(client, ip_addr, MQTT_PORT, mqtt_connection_cb, 0, &ci ←
      );

  /* For now just print the result code if something goes wrong */
  if(err != ERR_OK) {
    printf("mqtt_connect return %d\n", err);
  }
}

Connection to server can also be probed by calling mqtt_client_is_connected(client ←
   )

-----------------------------------------------------------------
2. Implementing the connection status callback


static void mqtt_connection_cb(mqtt_client_t *client, void *arg,  ←
   mqtt_connection_status_t status)
{
  err_t err;
  if(status == MQTT_CONNECT_ACCEPTED) {
    printf("mqtt_connection_cb: Successfully connected\n");

    /* Setup callback for incoming publish requests */
    mqtt_set_inpub_callback(client, mqtt_incoming_publish_cb,  ↩
       mqtt_incoming_data_cb, arg);

    /* Subscribe to a topic named "subtopic" with QoS level 1, call  ↩
       mqtt_sub_request_cb with result */
    err = mqtt_subscribe(client, "subtopic", 1, mqtt_sub_request_cb, arg);

    if(err != ERR_OK) {
      printf("mqtt_subscribe return: %d\n", err);
    }
  } else {
    printf("mqtt_connection_cb: Disconnected, reason: %d\n", status);

    /* Its more nice to be connected, so try to reconnect */
    example_do_connect(client);
  }
}

static void mqtt_sub_request_cb(void *arg, err_t result)
{
  /* Just print the result code here for simplicity,
     normal behaviour would be to take some action if subscribe fails like
     notifying user, retry subscribe or disconnect from server */
  printf("Subscribe result: %d\n", result);
}

-----------------------------------------------------------------
3. Implementing callbacks for incoming publish and data

/* The idea is to demultiplex topic and create some reference to be used in data  ↩
   callbacks
   Example here uses a global variable, better would be to use a member in arg
   If RAM and CPU budget allows it, the easiest implementation might be to just  ↩
      take a copy of
   the topic string and use it in mqtt_incoming_data_cb
*/
static int inpub_id;
static void mqtt_incoming_publish_cb(void *arg, const char *topic, u32_t tot_len)
{
  printf("Incoming publish at topic %s with total length %u\n", topic, (unsigned  ↩
     int)tot_len);

  /* Decode topic string into a user defined reference */
  if(strcmp(topic, "print_payload") == 0) {
    inpub_id = 0;
  } else if(topic[0] == 'A') {
    /* All topics starting with 'A' might be handled at the same way */
    inpub_id = 1;
  } else {
    /* For all other topics */
    inpub_id = 2;
  }
```

```
}

static void mqtt_incoming_data_cb(void *arg, const u8_t *data, u16_t len, u8_t ↩
    flags)
{
  printf("Incoming publish payload with length %d, flags %u\n", len, (unsigned int ↩
      )flags);

  if(flags & MQTT_DATA_FLAG_LAST) {
    /* Last fragment of payload received (or whole part if payload fits receive ↩
       buffer
       See MQTT_VAR_HEADER_BUFFER_LEN)  */

    /* Call function or do action depending on reference, in this case inpub_id */
    if(inpub_id == 0) {
      /* Don't trust the publisher, check zero termination */
      if(data[len-1] == 0) {
        printf("mqtt_incoming_data_cb: %s\n", (const char *)data);
      }
    } else if(inpub_id == 1) {
      /* Call an 'A' function... */
    } else {
      printf("mqtt_incoming_data_cb: Ignoring payload...\n");
    }
  } else {
    /* Handle fragmented payload, store in buffer, write to file or whatever */
  }
}

-----------------------------------------------------------------
4. Using outgoing publish


void example_publish(mqtt_client_t *client, void *arg)
{
  const char *pub_payload= "PubSubHubLubJub";
  err_t err;
  u8_t qos = 2; /* 0 1 or 2, see MQTT specification */
  u8_t retain = 0; /* No don't retain such crappy payload... */
  err = mqtt_publish(client, "pub_topic", pub_payload, strlen(pub_payload), qos, ↩
      retain, mqtt_pub_request_cb, arg);
  if(err != ERR_OK) {
    printf("Publish err: %d\n", err);
  }
}

/* Called when publish is complete either with success or failure */
static void mqtt_pub_request_cb(void *arg, err_t result)
{
  if(result != ERR_OK) {
    printf("Publish result: %d\n", result);
  }
}

-----------------------------------------------------------------
5. Disconnecting
```

```
Simply call mqtt_disconnect(client)
```

2.5.7.8 Macro Definition Documentation

2.5.7.8.1 MQTT_PORT

```
#define MQTT_PORT   LWIP_IANA_PORT_MQTT
```

Default MQTT port (non-TLS)

2.5.7.8.2 mqtt_subscribe

```
#define mqtt_subscribe( client, topic, qos, cb, arg)   mqtt_sub_unsub(client, topic, qos,
cb, arg, 1)
```

Subscribe to topic

2.5.7.8.3 MQTT_TLS_PORT

```
#define MQTT_TLS_PORT   LWIP_IANA_PORT_SECURE_MQTT
```

Default MQTT TLS port

2.5.7.8.4 mqtt_unsubscribe

```
#define mqtt_unsubscribe( client, topic, cb, arg)   mqtt_sub_unsub(client, topic, 0, cb,
arg, 0)
```

Unsubscribe to topic

2.5.7.9 Typedef Documentation

2.5.7.9.1 mqtt_connection_cb_t

```
typedef void(* mqtt_connection_cb_t) (mqtt_client_t *client, void *arg, mqtt_connection_st
status)
```

Function prototype for mqtt connection status callback. Called when client has connected to the server after initiating a mqtt connection attempt by calling mqtt_client_connect() or when connection is closed by server or an error

Parameters

client	MQTT client itself
arg	Additional argument to pass to the callback function
status	Connect result code or disconnection notification

See also mqtt_connection_status_t

2.5.7.9.2 mqtt_incoming_data_cb_t

```
typedef void(* mqtt_incoming_data_cb_t) (void *arg, const u8_t *data, u16_t len, u8_t flag
```

Function prototype for MQTT incoming publish data callback function. Called when data arrives to a subscribed topic **See also** mqtt_subscribe

Parameters

arg	Additional argument to pass to the callback function
data	User data, pointed object, data may not be referenced after callback return, NULL is passed when all publish data are delivered
len	Length of publish data fragment
flags	MQTT_DATA_FLAG_LAST set when this call contains the last part of data from publish message

2.5.7.9.3 mqtt_incoming_publish_cb_t

```
typedef void(* mqtt_incoming_publish_cb_t) (void *arg, const char *topic, u32_t tot_len)
```

Function prototype for MQTT incoming publish function. Called when an incoming publish arrives to a subscribed topic **See also** mqtt_subscribe

Parameters

arg	Additional argument to pass to the callback function
topic	Zero terminated Topic text string, topic may not be referenced after callback return
tot_len	Total length of publish data, if set to 0 (no publish payload) data callback will not be invoked

2.5.7.9.4 mqtt_request_cb_t

```
typedef void(* mqtt_request_cb_t) (void *arg, err_t err)
```

Function prototype for mqtt request callback. Called when a subscribe, unsubscribe or publish request has completed

Parameters

arg	Pointer to user data supplied when invoking request
err	ERR_OK on success ERR_TIMEOUT if no response was received within timeout, ERR_ABRT if (un)subscribe was denied

2.5.7.10 Enumeration Type Documentation

2.5.7.10.1 anonymous enum

```
anonymous enum
```

Data callback flags

MQTT_DATA_FLAG_LAST	Flag set when last fragment of data arrives in data callback

2.5.7.10.2 mqtt_connection_status_t

```
enum mqtt_connection_status_t
```

Connection status codes

MQTT_CONNECT_ACCEPTED	Accepted
MQTT_CONNECT_REFUSED_PROTOCOL_VERSION	Refused protocol version
MQTT_CONNECT_REFUSED_IDENTIFIER	Refused identifier
MQTT_CONNECT_REFUSED_SERVER	Refused server
MQTT_CONNECT_REFUSED_USERNAME_PASS	Refused user credentials
MQTT_CONNECT_REFUSED_NOT_AUTHORIZED_	Refused not authorized
MQTT_CONNECT_DISCONNECTED	Disconnected
MQTT_CONNECT_TIMEOUT	Timeout

2.5.7.11 Function Documentation

2.5.7.11.1 mqtt_client_connect()

```
err_t mqtt_client_connect (mqtt_client_t * client, const ip_addr_t * ip_addr, u16_t port,
mqtt_connection_cb_t cb, void * arg, const struct mqtt_connect_client_info_t * client_info
```

Connect to MQTT server

Parameters

Returns ERR_OK if successful,

See also err_t enum for other results

2.5.7.11.2 mqtt_client_free()

```
void mqtt_client_free (mqtt_client_t * client)
```

Free MQTT client instance

Parameters

2.5.7.11.3 mqtt_client_is_connected()

```
u8_t mqtt_client_is_connected (mqtt_client_t * client)
```

Check connection with server

Parameters

Returns 1 if connected to server, 0 otherwise

2.5.7.11.4 mqtt_client_new()

```
mqtt_client_t * mqtt_client_new (void )
```

Create a new MQTT client instance **Returns** Pointer to instance on success, NULL otherwise

2.5.7.11.5 mqtt_disconnect()

```
void mqtt_disconnect (mqtt_client_t * client)
```

Disconnect from MQTT server

Parameters

client	MQTT client
ip_addr	Server IP
port	Server port
cb	Connection state change callback
arg	User supplied argument to connection callback
client_info	Client identification and connection options

client	Pointer to instance to be freed

2.5.7.11.6 mqtt_publish()

```
err_t mqtt_publish (mqtt_client_t * client, const char * topic, const void * payload, u16_
payload_length, u8_t qos, u8_t retain, mqtt_request_cb_t cb, void * arg)
```

MQTT publish function.

Parameters

Returns ERR_OK if successful ERR_CONN if client is disconnected ERR_MEM if short on memory

2.5.7.11.7 mqtt_set_inpub_callback()

```
void mqtt_set_inpub_callback (mqtt_client_t * client, mqtt_incoming_publish_cb_t pub_cb,
mqtt_incoming_data_cb_t data_cb, void * arg)
```

Set callback to handle incoming publish requests from server

Parameters

2.5.7.11.8 mqtt_sub_unsub()

```
err_t mqtt_sub_unsub (mqtt_client_t * client, const char * topic, u8_t qos, mqtt_request_c
cb, void * arg, u8_t sub)
```

MQTT subscribe/unsubscribe function.

Parameters

Returns ERR_OK if successful,

See also err_t enum for other results

2.5.7.12 Options

2.5.7.12.1 Macros

- #define MQTT_OUTPUT_RINGBUF_SIZE 256
- #define MQTT_VAR_HEADER_BUFFER_LEN 128
- #define MQTT_REQ_MAX_IN_FLIGHT 4
- #define MQTT_CYCLIC_TIMER_INTERVAL 5
- #define MQTT_REQ_TIMEOUT 30
- #define MQTT_CONNECT_TIMOUT 100

client	MQTT client

client	MQTT client

client	MQTT client
topic	Publish topic string
payload	Data to publish (NULL is allowed)
payload_length	Length of payload (0 is allowed)
qos	Quality of service, 0 1 or 2
retain	MQTT retain flag
cb	Callback to call when publish is complete or has timed out
arg	User supplied argument to publish callback

2.5.7.12.2 Detailed Description

2.5.7.12.3 Macro Definition Documentation

2.5.7.12.3.1 MQTT_CONNECT_TIMOUT

```
#define MQTT_CONNECT_TIMOUT   100
```

Seconds for MQTT connect response timeout after sending connect request

2.5.7.12.3.2 MQTT_CYCLIC_TIMER_INTERVAL

```
#define MQTT_CYCLIC_TIMER_INTERVAL   5
```

Seconds between each cyclic timer call.

2.5.7.12.3.3 MQTT_OUTPUT_RINGBUF_SIZE

```
#define MQTT_OUTPUT_RINGBUF_SIZE   256
```

Output ring-buffer size, must be able to fit largest outgoing publish message topic+payloads

2.5.7.12.3.4 MQTT_REQ_MAX_IN_FLIGHT

```
#define MQTT_REQ_MAX_IN_FLIGHT   4
```

Maximum number of pending subscribe, unsubscribe and publish requests to server .

2.5.7.12.3.5 MQTT_REQ_TIMEOUT

```
#define MQTT_REQ_TIMEOUT   30
```

Publish, subscribe and unsubscribe request timeout in seconds.

2.5.7.12.3.6 MQTT_VAR_HEADER_BUFFER_LEN

```
#define MQTT_VAR_HEADER_BUFFER_LEN   128
```

Number of bytes in receive buffer, must be at least the size of the longest incoming topic + 8 If one wants to avoid fragmented incoming publish, set length to max incoming topic length + max payload length + 8

client	MQTT client
pub_cb	Callback invoked when publish starts, contain topic and total length of payload
data_cb	Callback for each fragment of payload that arrives
arg	User supplied argument to both callbacks

client	MQTT client
topic	topic to subscribe to
qos	Quality of service, 0 1 or 2 (only used for subscribe)
cb	Callback to call when subscribe/unsubscribe response is received
arg	User supplied argument to publish callback
sub	1 for subscribe, 0 for unsubscribe

2.5.8 NETBIOS responder

2.5.8.1 Modules

- Options

2.5.8.2 Functions

- void netbiosns_init (void)
- void netbiosns_stop (void)

2.5.8.3 Detailed Description

This is an example implementation of a NetBIOS name server. It responds to name queries for a configurable name. Name resolving is not supported.

Note that the device doesn't broadcast it's own name so can't detect duplicate names!

2.5.8.4 Function Documentation

2.5.8.4.1 netbiosns_init()

```
void netbiosns_init (void )
```

Init netbios responder

2.5.8.4.2 netbiosns_stop()

```
void netbiosns_stop (void )
```

Stop netbios responder

2.5.8.5 Options

2.5.8.5.1 Macros

- #define NETBIOS_LWIP_NAME "NETBIOSLWIPDEV"
- #define LWIP_NETBIOS_RESPOND_NAME_QUERY 0

2.5.8.5.2 Detailed Description

2.5.8.5.3 Macro Definition Documentation

2.5.8.5.3.1 LWIP_NETBIOS_RESPOND_NAME_QUERY

```
#define LWIP_NETBIOS_RESPOND_NAME_QUERY   0
```

Respond to NetBIOS name queries Default is disabled

2.5.8.5.3.2 NETBIOS_LWIP_NAME

```
#define NETBIOS_LWIP_NAME   "NETBIOSLWIPDEV"
```

NetBIOS name of lwip device This must be uppercase until NETBIOS_STRCMP() is defined to a string comparison function that is case insensitive. If you want to use the netif's hostname, use this (with LWIP_NETIF_HOSTNAME): (ip_current_netif() != NULL ? ip_current_netif()->hostname != NULL ? ip_current_netif()->hostname : "" : "")

If this is not defined, netbiosns_set_name() can be called at runtime to change the name.

2.5.9 SMTP client

2.5.9.1 Modules

- Options

2.5.9.2 Functions

- err_t smtp_set_server_addr (const char *server)
- void smtp_set_server_port (u16_t port)
- void smtp_set_tls_config (struct altcp_tls_config *tls_config)
- err_t smtp_set_auth (const char *username, const char *pass)
- err_t smtp_send_mail (const char *from, const char *to, const char *subject, const char *body, smtp_result_fn callback_fn, void *callback_arg)
- err_t smtp_send_mail_static (const char *from, const char *to, const char *subject, const char *body, smtp_result_fn callback_fn, void *callback_arg)
- void smtp_send_mail_int (void *arg)

2.5.9.3 Detailed Description

This is simple SMTP client for raw API. It is a minimal implementation of SMTP as specified in RFC 5321.

Example usage:

```
void my_smtp_result_fn(void *arg, u8_t smtp_result, u16_t srv_err, err_t err)
{
  printf("mail (%p) sent with results: 0x%02x, 0x%04x, 0x%08x\n", arg,
         smtp_result, srv_err, err);
}
static void my_smtp_test(void)
{
  smtp_set_server_addr("mymailserver.org");
  -> set both username and password as NULL if no auth needed
  smtp_set_auth("username", "password");
  smtp_send_mail("sender", "recipient", "subject", "body", my_smtp_result_fn,
                 some_argument);
}
```

When using from any other thread than the tcpip_thread (for NO_SYS==0), use smtp_send_mail_int()!

SMTP_BODYDH usage:

```
int my_smtp_bodydh_fn(void *arg, struct smtp_bodydh *bdh)
{
   if(bdh->state >= 10) {
      return BDH_DONE;
   }
   sprintf(bdh->buffer,"Line #%2d\r\n",bdh->state);
   bdh->length = strlen(bdh->buffer);
   ++bdh->state;
   return BDH_WORKING;
}

smtp_send_mail_bodycback("sender", "recipient", "subject",
               my_smtp_bodydh_fn, my_smtp_result_fn, some_argument);
```

2.5.9.4 Function Documentation

2.5.9.4.1 smtp_send_mail()

`err_t smtp_send_mail (const char * from, const char * to, const char * subject, const char * body, smtp_result_fn callback_fn, void * callback_arg)`

Send an email via the currently selected server, username and password.

Parameters

from	source email address (must be NULL-terminated)
to	target email address (must be NULL-terminated)
subject	email subject (must be NULL-terminated)
body	email body (must be NULL-terminated)
callback_fn	callback function
callback_arg	user argument to callback_fn

Returns - ERR_OK if structures were allocated and no error occurred starting the connection (this does not mean the email has been successfully sent!)

- another err_t on error.

2.5.9.4.2 smtp_send_mail_int()

`void smtp_send_mail_int (void * arg)`

Same as smtp_send_mail but takes a struct smtp_send_request as single parameter which contains all the other parameters. To be used with tcpip_callback to send mail from interrupt context or from another thread.

WARNING: server and authentication must stay untouched until this function has run!

Usage example:

- allocate a struct smtp_send_request (in a way that is allowed in interrupt context)
- fill the members of the struct as if calling smtp_send_mail
- specify a callback_function
- set callback_arg to the structure itself

- call this function
- wait for the callback function to be called
- in the callback function, deallocate the structure (passed as arg)

2.5.9.4.3 smtp_send_mail_static()

```
err_t smtp_send_mail_static (const char * from, const char * to, const char * subject,
const char * body, smtp_result_fn callback_fn, void * callback_arg)
```

Same as smtp_send_mail, but doesn't copy from, to, subject and body into an internal buffer to save memory. WARNING: the above data must stay untouched until the callback function is called (unless the function returns != ERR_OK)

2.5.9.4.4 smtp_set_auth()

```
err_t smtp_set_auth (const char * username, const char * pass)
```

Set authentication parameters for next SMTP connection

Parameters

username	login name as passed to the server
pass	password passed to the server together with username

2.5.9.4.5 smtp_set_server_addr()

```
err_t smtp_set_server_addr (const char * server)
```

Set IP address or DNS name for next SMTP connection

Parameters

server	IP address (in ASCII representation) or DNS name of the server

2.5.9.4.6 smtp_set_server_port()

```
void smtp_set_server_port (u16_t port)
```

Set TCP port for next SMTP connection

Parameters

port	TCP port

2.5.9.4.7 smtp_set_tls_config()

```
void smtp_set_tls_config (struct altcp_tls_config * tls_config)
```

Set TLS configuration for next SMTP connection

Parameters

tls_config	TLS configuration

2.5.9.5 Options

2.5.9.5.1 Macros

- #define SMTP_BODYDH 0
- #define SMTP_DEBUG LWIP_DBG_OFF
- #define SMTP_MAX_SERVERNAME_LEN 256
- #define SMTP_MAX_USERNAME_LEN 32
- #define SMTP_MAX_PASS_LEN 32
- #define SMTP_COPY_AUTHDATA 1
- #define SMTP_CHECK_DATA 1
- #define SMTP_SUPPORT_AUTH_PLAIN 1
- #define SMTP_SUPPORT_AUTH_LOGIN 1

2.5.9.5.2 Detailed Description

2.5.9.5.3 Macro Definition Documentation

2.5.9.5.3.1 SMTP_BODYDH

```
#define SMTP_BODYDH   0
```

Set this to 1 to enable data handler callback on BODY

2.5.9.5.3.2 SMTP_CHECK_DATA

```
#define SMTP_CHECK_DATA   1
```

Set this to 0 to save some code space if you know for sure that all data passed to this module conforms to the requirements in the SMTP RFC. WARNING: use this with care!

2.5.9.5.3.3 SMTP_COPY_AUTHDATA

```
#define SMTP_COPY_AUTHDATA   1
```

Set this to 0 if you know the authentication data will not change during the smtp session, which saves some heap space.

2.5.9.5.3.4 SMTP_DEBUG

```
#define SMTP_DEBUG   LWIP_DBG_OFF
```

SMTP_DEBUG: Enable debugging for SNTP.

2.5.9.5.3.5 SMTP_MAX_PASS_LEN

```
#define SMTP_MAX_PASS_LEN   32
```

Maximum length reserved for password

2.5.9.5.3.6 SMTP_MAX_SERVERNAME_LEN

```
#define SMTP_MAX_SERVERNAME_LEN   256
```

Maximum length reserved for server name including terminating 0 byte

2.5.9.5.3.7 SMTP_MAX_USERNAME_LEN

```
#define SMTP_MAX_USERNAME_LEN   32
```

Maximum length reserved for username

2.5.9.5.3.8 SMTP_SUPPORT_AUTH_LOGIN

```
#define SMTP_SUPPORT_AUTH_LOGIN   1
```

Set this to 1 to enable AUTH LOGIN support

2.5.9.5.3.9 SMTP_SUPPORT_AUTH_PLAIN

```
#define SMTP_SUPPORT_AUTH_PLAIN   1
```

Set this to 1 to enable AUTH PLAIN support

2.5.10 SNMPv2c/v3 agent

2.5.10.1 Modules

- Core
- Traps
- MIB2
- Options

2.5.10.2 Detailed Description

SNMPv2c and SNMPv3 compatible agent There is also a MIB compiler and a MIB viewer in lwIP/contrib subdir (lwip/contrib/apps/LwipMibCompiler). The agent implements the most important MIB2 MIBs including IPv6 support (interfaces, UDP, TCP, SNMP, ICMP, SYSTEM). IP MIB is an older version without IPv6 statistics (TODO). Rewritten by Martin Hentschel info@cl-soft.de and Dirk Ziegelmeier dziegel@gmx.de

0 Agent Capabilities

Features:

- SNMPv2c support.
- SNMPv3 support (a port to ARM mbedtls is provided, LWIP_SNMP_V3_MBEDTLS option).
- Low RAM usage - no memory pools, stack only.
- MIB2 implementation is separated from SNMP stack.
- Support for multiple MIBs (snmp_set_mibs() call) - e.g. for private MIB.
- Simple and generic API for MIB implementation.
- Comfortable node types and helper functions for scalar arrays and tables.

- Counter64, bit and truthvalue datatype support.
- Callbacks for SNMP writes e.g. to implement persistency.
- Runs on two APIs: RAW and netconn.
- Async API is gone - the stack now supports netconn API instead, so blocking operations can be done in MIB calls. SNMP runs in a worker thread when netconn API is used.
- Simplified thread sync support for MIBs - useful when MIBs need to access variables shared with other threads where no locking is possible. Used in MIB2 to access lwIP stats from lwIP thread.

MIB compiler (code generator):

- Provided in contrib dir.
- Written in C#. MIB viewer using Windows Forms.
- Developed on Windows with Visual Studio 2010.
- Can be compiled and used on all platforms with http://www.monodevelop.com/.
- Based on a heavily modified version of of SharpSnmpLib (a4bd05c6afb4) (https://sharpsnmplib.codeplex.com/SourceControl/-network/forks/Nemo157/MIBParserUpdate). This has been the last known revision of that code before being converted to closed source. The new code on github has completely changed, so we can not just update :-(
- MIB parser, C file generation framework and LWIP code generation are cleanly separated, which means the code may be useful as a base for code generation of other SNMP agents.

Notes:

- Stack and MIB compiler were used to implement a Profinet device. Compiled/implemented MIBs: LLDP-MIB, LLDP-EXT-DOT3-MIB, LLDP-EXT-PNO-MIB.

SNMPv1 per RFC1157 and SNMPv2c per RFC 3416

Note the S in SNMP stands for "Simple". Note that "Simple" is relative. SNMP is simple compared to the complex ISO network management protocols CMIP (Common Management Information Protocol) and CMOT (CMip Over Tcp).

SNMPv3

When SNMPv3 is used, several functions from snmpv3.h must be implemented by the user. This is mainly user management and persistence handling. The sample provided in lwip-contrib is insecure, don't use it in production systems, especially the missing persistence for engine boots variable simplifies replay attacks.

MIB II

The standard lwIP stack management information base. This is a required MIB, so this is always enabled. The groups EGP, CMOT and transmission are disabled by default.

Most mib-2 objects are not writable except: sysName, sysLocation, sysContact, snmpEnableAuthenTraps. Writing to or changing the ARP and IP address and route tables is not possible.

Note lwIP has a very limited notion of IP routing. It currently doesn't have a route table and doesn't have a notion of the U,G,H flags. Instead lwIP uses the interface list with only one default interface acting as a single gateway interface (G) for the default route.

The agent returns a "virtual table" with the default route 0.0.0.0 for the default interface and network routes (no H) for each network interface in the netif_list. All routes are considered to be up (U).

Loading additional MIBs

MIBs can only be added in compile-time, not in run-time.

1 Building the Agent

First of all you'll need to add the following define to your local lwipopts.h: #define LWIP_SNMP 1

and add the source files your makefile.

Note you'll might need to adapt you network driver to update the mib2 variables for your interface.

2 Running the Agent

The following function calls must be made in your program to actually get the SNMP agent running.

Before starting the agent you should supply pointers for sysContact, sysLocation, and snmpEnableAuthenTraps. You can do this by calling

- snmp_mib2_set_syscontact()
- snmp_mib2_set_syslocation()
- snmp_set_auth_traps_enabled()

You can register a callback which is called on successful write access: snmp_set_write_callback().

Additionally you may want to set

- snmp_mib2_set_sysdescr()
- snmp_set_device_enterprise_oid()
- snmp_mib2_set_sysname()

Also before starting the agent you need to setup one or more trap destinations using these calls:

- snmp_trap_dst_enable()
- snmp_trap_dst_ip_set()

If you need more than MIB2, set the MIBs you want to use by snmp_set_mibs().

Finally, enable the agent by calling snmp_init()

2.5.10.3 Core

2.5.10.3.1 Functions

- void snmp_set_mibs (const struct snmp_mib **mibs, u8_t num_mibs)
- void snmp_set_device_enterprise_oid (const struct snmp_obj_id *device_enterprise_oid)
- const struct snmp_obj_id * snmp_get_device_enterprise_oid (void)
- const char * snmp_get_community (void)
- void snmp_set_community (const char *const community)
- const char * snmp_get_community_write (void)
- void snmp_set_community_write (const char *const community)
- void snmp_set_write_callback (snmp_write_callback_fct write_callback, void *callback_arg)
- void snmp_set_inform_callback (snmp_inform_callback_fct inform_callback, void *callback_arg)
- void snmp_init (void)

2.5.10.3.2 Detailed Description

2.5.10.3.3 Function Documentation

2.5.10.3.3.1 snmp_get_community()

```
const char * snmp_get_community (void )
```

Returns current SNMP community string. **Returns** current SNMP community string

2.5.10.3.3.2 snmp_get_community_write()

```
const char * snmp_get_community_write (void )
```

Returns current SNMP write-access community string. **Returns** current SNMP write-access community string

2.5.10.3.3.3 snmp_get_device_enterprise_oid()

```
const struct snmp_obj_id * snmp_get_device_enterprise_oid (void )
```

Get 'device enterprise oid'

2.5.10.3.3.4 snmp_init()

```
void snmp_init (void )
```

Starts SNMP Agent. Allocates UDP pcb and binds it to IP_ANY_TYPE port 161.

Agent setup, start listening to port 161.

2.5.10.3.3.5 snmp_set_community()

```
void snmp_set_community (const char *const community)
```

Sets SNMP community string. The string itself (its storage) must be valid throughout the whole life of program (or until it is changed to sth else).

Parameters

community	is a pointer to new community string

2.5.10.3.3.6 snmp_set_community_write()

```
void snmp_set_community_write (const char *const community)
```

Sets SNMP community string for write-access. The string itself (its storage) must be valid throughout the whole life of program (or until it is changed to sth else).

Parameters

community	is a pointer to new write-access community string

2.5.10.3.3.7 snmp_set_device_enterprise_oid()

```
void snmp_set_device_enterprise_oid (const struct snmp_obj_id * device_enterprise_oid)
```

'device enterprise oid' is used for 'device OID' field in trap PDU's (for identification of generating device) as well as for value returned by MIB-2 'sysObjectID' field (if internal MIB2 implementation is used). The 'device enterprise oid' shall point to an OID located under 'private-enterprises' branch (1.3.6.1.4.1.XXX). If a vendor wants to provide a custom object there, he has to get its own enterprise oid from IANA (http://www.iana.org). It is not allowed to use LWIP enterprise ID! In order to identify a specific device it is recommended to create a dedicated OID for each device type under its own enterprise oid. e.g. device a > 1.3.6.1.4.1.XXX(ent-oid).1(devices).1(device a) device b > 1.3.6.1.4.1.XXX(ent-oid).1(devices).2(device b) for more details see description of 'sysObjectID' field in RFC1213-MIB

2.5.10.3.3.8 snmp_set_inform_callback()

```
void snmp_set_inform_callback (snmp_inform_callback_fct inform_callback, void * callback_a
```

Callback fired on every received INFORM confirmation (get-response)

2.5.10.3.3.9 snmp_set_mibs()

```
void snmp_set_mibs (const struct snmp_mib ** mibs, u8_t num_mibs)
```

Sets the MIBs to use. Example: call snmp_set_mibs() as follows: static const struct snmp_mib *my_snmp_mibs[] = { &mib2, &private_mib }; snmp_set_mibs(my_snmp_mibs, LWIP_ARRAYSIZE(my_snmp_mibs));

2.5.10.3.3.10 snmp_set_write_callback()

```
void snmp_set_write_callback (snmp_write_callback_fct write_callback, void * callback_arg)
```

Callback fired on every successful write access

2.5.10.4 Traps

2.5.10.4.1 Functions

- const char * snmp_get_community_trap (void)
- void snmp_set_community_trap (const char *const community)
- void snmp_trap_dst_enable (u8_t dst_idx, u8_t enable)
- void snmp_trap_dst_ip_set (u8_t dst_idx, const ip_addr_t *dst)
- void snmp_set_auth_traps_enabled (u8_t enable)
- u8_t snmp_get_auth_traps_enabled (void)
- void snmp_set_default_trap_version (u8_t snmp_version)
- u8_t snmp_get_default_trap_version (void)
- err_t snmp_send_trap (const struct snmp_obj_id *oid, s32_t generic_trap, s32_t specific_trap, struct snmp_varbind *varbinds)
- err_t snmp_send_trap_generic (s32_t generic_trap)
- err_t snmp_send_trap_specific (s32_t specific_trap, struct snmp_varbind *varbinds)
- void snmp_coldstart_trap (void)
- void snmp_authfail_trap (void)
- err_t snmp_send_inform_specific (s32_t specific_trap, struct snmp_varbind *varbinds, s32_t *ptr_request_id)
- err_t snmp_send_inform_generic (s32_t generic_trap, struct snmp_varbind *varbinds, s32_t *ptr_request_id)
- err_t snmp_send_inform (const struct snmp_obj_id *oid, s32_t generic_trap, s32_t specific_trap, struct snmp_varbind *varbinds, s32_t *ptr_request_id)

2.5.10.4.2 Detailed Description

2.5.10.4.3 Function Documentation

2.5.10.4.3.1 snmp_authfail_trap()

```
void snmp_authfail_trap (void )
```

Send authentication failure trap (used internally by agent)

2.5.10.4.3.2 snmp_coldstart_trap()

```
void snmp_coldstart_trap (void )
```

Send coldstart trap

2.5.10.4.3.3 snmp_get_auth_traps_enabled()

```
u8_t snmp_get_auth_traps_enabled (void )
```

Get authentication traps enabled state

Returns TRUE if traps are enabled, FALSE if they aren't

2.5.10.4.3.4 snmp_get_community_trap()

```
const char * snmp_get_community_trap (void )
```

Returns current SNMP community string used for sending traps. **Returns** current SNMP community string used for sending traps

2.5.10.4.3.5 snmp_get_default_trap_version()

```
u8_t snmp_get_default_trap_version (void )
```

Get default SNMP version for sending traps

Returns selected default version: 0 - SNMP_VERSION_1 1 - SNMP_VERSION_2c 3 - SNMP_VERSION_3

2.5.10.4.3.6 snmp_send_inform()

```
err_t snmp_send_inform (const struct snmp_obj_id * oid, s32_t generic_trap, s32_t specific
struct snmp_varbind * varbinds, s32_t * ptr_request_id)
```

Generic function for sending informs

Parameters

oid	points to object identifier
generic_trap	is the trap code
specific_trap	used for enterprise traps when generic_trap == 6
varbinds	linked list of varbinds (at the beginning of this list function will insert 2 special purpose varbinds [see RFC 3584])
ptr_request_id	[out] variable in which to store request_id needed to verify acknowledgement

Returns ERR_OK if successful

2.5.10.4.3.7 snmp_send_inform_generic()

err_t snmp_send_inform_generic (s32_t generic_trap, struct snmp_varbind * varbinds, s32_t * ptr_request_id)

Wrapper function for sending informs

Parameters

generic_trap	is the trap code
varbinds	linked list of varbinds (at the beginning of this list function will insert 2 special purpose varbinds [see RFC 3584])
ptr_request_id	[out] variable in which to store request_id needed to verify acknowledgement

Returns ERR_OK if successful

2.5.10.4.3.8 snmp_send_inform_specific()

err_t snmp_send_inform_specific (s32_t specific_trap, struct snmp_varbind * varbinds, s32_
* ptr_request_id)

Wrapper function for sending informs

Parameters

specific_trap	will be appended to enterprise oid [see RFC 3584]
varbinds	linked list of varbinds (at the beginning of this list function will insert 2 special purpose varbinds [see RFC 3584])
ptr_request_id	[out] variable in which to store request_id needed to verify acknowledgement

Returns ERR_OK if successful

2.5.10.4.3.9 snmp_send_trap()

err_t snmp_send_trap (const struct snmp_obj_id * oid, s32_t generic_trap, s32_t specific_t
struct snmp_varbind * varbinds)

This function is a wrapper function for preparing and sending generic or specific traps.

Parameters

oid	points to enterprise object identifier
generic_trap	is the trap code
specific_trap	used for enterprise traps when generic_trap == 6
varbinds	linked list of varbinds to be sent

Returns ERR_OK when success, ERR_MEM if we're out of memory

Note
the use of the enterprise identifier field is per RFC1215. Use .iso.org.dod.internet.mgmt.mib-2.snmp for generic traps and .iso.org.dod.internet.private.enterprises.yourenterprise (sysObjectID) for specific traps.

2.5.10.4.3.10 snmp_send_trap_generic()

```
err_t snmp_send_trap_generic (s32_t generic_trap)
```

Send generic SNMP trap

Parameters

generic_trap	is the trap code return ERR_OK when success

2.5.10.4.3.11 snmp_send_trap_specific()

```
err_t snmp_send_trap_specific (s32_t specific_trap, struct snmp_varbind * varbinds)
```

Send specific SNMP trap with variable bindings

Parameters

specific_trap	used for enterprise traps (generic_trap = 6)
varbinds	linked list of varbinds to be sent

Returns ERR_OK when success

2.5.10.4.3.12 snmp_set_auth_traps_enabled()

```
void snmp_set_auth_traps_enabled (u8_t enable)
```

Enable/disable authentication traps

Parameters

enable	enable SNMP traps

2.5.10.4.3.13 snmp_set_community_trap()

```
void snmp_set_community_trap (const char *const community)
```

Sets SNMP community string used for sending traps. The string itself (its storage) must be valid throughout the whole life of program (or until it is changed to sth else).

Parameters

2.5.10.4.3.14 snmp_set_default_trap_version()

```
void snmp_set_default_trap_version (u8_t snmp_version)
```

Choose default SNMP version for sending traps (if not specified, default version is SNMP_VERSION_1) SNMP_VERSION_1 0 SNMP_VERSION_2c 1 SNMP_VERSION_3 3

Parameters

2.5.10.4.3.15 snmp_trap_dst_enable()

```
void snmp_trap_dst_enable (u8_t dst_idx, u8_t enable)
```

Sets enable switch for this trap destination.

Parameters

community	is a pointer to new trap community string

snmp_version	version that will be used for sending traps

2.5.10.4.3.16 snmp_trap_dst_ip_set()

```
void snmp_trap_dst_ip_set (u8_t dst_idx, const ip_addr_t * dst)
```

Sets IPv4 address for this trap destination.

Parameters

2.5.10.5 MIB2

2.5.10.5.1 Functions

- void snmp_mib2_set_sysdescr (const u8_t *str, const u16_t *len)
- void snmp_mib2_set_syscontact (u8_t *ocstr, u16_t *ocstrlen, u16_t bufsize)
- void snmp_mib2_set_syscontact_readonly (const u8_t *ocstr, const u16_t *ocstrlen)
- void snmp_mib2_set_sysname (u8_t *ocstr, u16_t *ocstrlen, u16_t bufsize)
- void snmp_mib2_set_sysname_readonly (const u8_t *ocstr, const u16_t *ocstrlen)
- void snmp_mib2_set_syslocation (u8_t *ocstr, u16_t *ocstrlen, u16_t bufsize)
- void snmp_mib2_set_syslocation_readonly (const u8_t *ocstr, const u16_t *ocstrlen)

2.5.10.5.2 Detailed Description

2.5.10.5.3 Function Documentation

2.5.10.5.3.1 snmp_mib2_set_syscontact()

```
void snmp_mib2_set_syscontact (u8_t * ocstr, u16_t * ocstrlen, u16_t bufsize)
```

Initializes sysContact pointers

Parameters

2.5.10.5.3.2 snmp_mib2_set_syscontact_readonly()

```
void snmp_mib2_set_syscontact_readonly (const u8_t * ocstr, const u16_t * ocstrlen)
```

see snmp_mib2_set_syscontact but set pointer to readonly memory

2.5.10.5.3.3 snmp_mib2_set_sysdescr()

```
void snmp_mib2_set_sysdescr (const u8_t * str, const u16_t * len)
```

Initializes sysDescr pointers.

Parameters

dst_idx	index in 0 .. SNMP_TRAP_DESTINATIONS-1
enable	switch if 0 destination is disabled >0 enabled.

dst_idx	index in 0 .. SNMP_TRAP_DESTINATIONS-1
dst	IPv4 address in host order.

ocstr	if non-NULL then copy str pointer
ocstrlen	points to string length, excluding zero terminator. if set to NULL it is assumed that ocstr is NULL-terminated.
bufsize	size of the buffer in bytes. (this is required because the buffer can be overwritten by snmp-set) if ocstrlen is NULL buffer needs space for terminating 0 byte. otherwise complete buffer is used for string. if bufsize is set to 0, the value is regarded as read-only.

2.5.10.5.3.4 snmp_mib2_set_syslocation()

```
void snmp_mib2_set_syslocation (u8_t * ocstr, u16_t * ocstrlen, u16_t bufsize)
```

Initializes sysLocation pointers

Parameters

2.5.10.5.3.5 snmp_mib2_set_syslocation_readonly()

```
void snmp_mib2_set_syslocation_readonly (const u8_t * ocstr, const u16_t * ocstrlen)
```

see snmp_mib2_set_syslocation but set pointer to readonly memory

2.5.10.5.3.6 snmp_mib2_set_sysname()

```
void snmp_mib2_set_sysname (u8_t * ocstr, u16_t * ocstrlen, u16_t bufsize)
```

Initializes sysName pointers

Parameters

2.5.10.5.3.7 snmp_mib2_set_sysname_readonly()

```
void snmp_mib2_set_sysname_readonly (const u8_t * ocstr, const u16_t * ocstrlen)
```

see snmp_mib2_set_sysname but set pointer to readonly memory

2.5.10.6 Options

2.5.10.6.1 Macros

- #define LWIP_SNMP 0
- #define SNMP_USE_NETCONN 0
- #define SNMP_USE_RAW 1
- #define SNMP_STACK_SIZE DEFAULT_THREAD_STACKSIZE
- #define SNMP_THREAD_PRIO DEFAULT_THREAD_PRIO
- #define SNMP_TRAP_DESTINATIONS 1

str	if non-NULL then copy str pointer
len	points to string length, excluding zero terminator

ocstr	if non-NULL then copy str pointer
ocstrlen	points to string length, excluding zero terminator. if set to NULL it is assumed that ocstr is NULL-terminated.
bufsize	size of the buffer in bytes. (this is required because the buffer can be overwritten by snmp-set) if ocstrlen is NULL buffer needs space for terminating 0 byte. otherwise complete buffer is used for string. if bufsize is set to 0, the value is regarded as read-only.

ocstr	if non-NULL then copy str pointer
ocstrlen	points to string length, excluding zero terminator. if set to NULL it is assumed that ocstr is NULL-terminated.
bufsize	size of the buffer in bytes. (this is required because the buffer can be overwritten by snmp-set) if ocstrlen is NULL buffer needs space for terminating 0 byte. otherwise complete buffer is used for string. if bufsize is set to 0, the value is regarded as read-only.

- #define SNMP_SAFE_REQUESTS 1
- #define SNMP_MAX_OCTET_STRING_LEN 127
- #define SNMP_MAX_OBJ_ID_LEN 50
- #define SNMP_MIN_VALUE_SIZE (2 * sizeof(u32_t*)) /* size required to store the basic types (8 bytes for counter64) */
- #define SNMP_MAX_VALUE_SIZE LWIP_MAX(LWIP_MAX((SNMP_MAX_OCTET_STRING_LEN), sizeof(u32_t)*(SNMP_I
SNMP_MIN_VALUE_SIZE)
- #define SNMP_COMMUNITY "public"
- #define SNMP_COMMUNITY_WRITE "private"
- #define SNMP_COMMUNITY_TRAP "public"
- #define SNMP_MAX_COMMUNITY_STR_LEN LWIP_MAX(LWIP_MAX(sizeof(SNMP_COMMUNITY), sizeof(SNMP_COM
sizeof(SNMP_COMMUNITY_TRAP))
- #define SNMP_LWIP_ENTERPRISE_OID 26381
- #define SNMP_DEVICE_ENTERPRISE_OID {1, 3, 6, 1, 4, 1, SNMP_LWIP_ENTERPRISE_OID}
- #define SNMP_DEVICE_ENTERPRISE_OID_LEN 7
- #define SNMP_DEBUG LWIP_DBG_OFF
- #define SNMP_MIB_DEBUG LWIP_DBG_OFF
- #define SNMP_LWIP_MIB2 LWIP_SNMP
- #define SNMP_LWIP_MIB2_SYSDESC "lwIP"
- #define SNMP_LWIP_MIB2_SYSNAME "FQDN-unk"
- #define SNMP_LWIP_MIB2_SYSCONTACT ""
- #define SNMP_LWIP_MIB2_SYSLOCATION ""
- #define SNMP_LWIP_GETBULK_MAX_REPETITIONS 0

2.5.10.6.2 Detailed Description

2.5.10.6.3 Macro Definition Documentation

2.5.10.6.3.1 LWIP_SNMP

```
#define LWIP_SNMP   0
```

LWIP_SNMP==1: This enables the lwIP SNMP agent. UDP must be available for SNMP transport. If you want to use your own SNMP agent, leave this disabled. To integrate MIB2 of an external agent, you need to enable LWIP_MIB2_CALLBACKS and MIB2_STATS. This will give you the callbacks and statistics counters you need to get MIB2 working.

2.5.10.6.3.2 SNMP_COMMUNITY

```
#define SNMP_COMMUNITY   "public"
```

The snmp read-access community. Used for write-access and traps, too unless SNMP_COMMUNITY_WRITE or SNMP_COMMUNIT
are enabled, respectively.

2.5.10.6.3.3 SNMP_COMMUNITY_TRAP

```
#define SNMP_COMMUNITY_TRAP   "public"
```

The snmp community used for sending traps.

2.5.10.6.3.4 SNMP_COMMUNITY_WRITE

```
#define SNMP_COMMUNITY_WRITE   "private"
```

The snmp write-access community. Set this community to "" in order to disallow any write access.

2.5.10.6.3.5 SNMP_DEBUG

```
#define SNMP_DEBUG   LWIP_DBG_OFF
```

SNMP_DEBUG: Enable debugging for SNMP messages.

2.5.10.6.3.6 SNMP_DEVICE_ENTERPRISE_OID

```
#define SNMP_DEVICE_ENTERPRISE_OID   {1, 3, 6, 1, 4, 1, SNMP_LWIP_ENTERPRISE_OID}
```

IANA assigned enterprise ID for lwIP is 26381 **See also** http://www.iana.org/assignments/enterprise-numbers

Note
this enterprise ID is assigned to the lwIP project, all object identifiers living under this ID are assigned by the lwIP maintainers! don't change this define, use snmp_set_device_enterprise_oid()

If you need to create your own private MIB you'll need to apply for your own enterprise ID with IANA: http://www.iana.org/-numbers.html

2.5.10.6.3.7 SNMP_DEVICE_ENTERPRISE_OID_LEN

```
#define SNMP_DEVICE_ENTERPRISE_OID_LEN   7
```

Length of SNMP_DEVICE_ENTERPRISE_OID

2.5.10.6.3.8 SNMP_LWIP_ENTERPRISE_OID

```
#define SNMP_LWIP_ENTERPRISE_OID   26381
```

The OID identifiying the device. This may be the enterprise OID itself or any OID located below it in tree.

2.5.10.6.3.9 SNMP_LWIP_GETBULK_MAX_REPETITIONS

```
#define SNMP_LWIP_GETBULK_MAX_REPETITIONS   0
```

This value is used to limit the repetitions processed in GetBulk requests (value == 0 means no limitation). This may be useful to limit the load for a single request. According to SNMP RFC 1905 it is allowed to not return all requested variables from a GetBulk request if system load would be too high. so the effect is that the client will do more requests to gather all data. For the stack this could be useful in case that SNMP processing is done in TCP/IP thread. In this situation a request with many repetitions could block the thread for a longer time. Setting limit here will keep the stack more responsive.

2.5.10.6.3.10 SNMP_LWIP_MIB2

```
#define SNMP_LWIP_MIB2   LWIP_SNMP
```

Indicates if the MIB2 implementation of LWIP SNMP stack is used.

2.5.10.6.3.11 SNMP_LWIP_MIB2_SYSCONTACT

```
#define SNMP_LWIP_MIB2_SYSCONTACT   ""
```

Value return for sysContact field of MIB2. To make sysContact field settable, call snmp_mib2_set_syscontact() to provide the necessary buffers.

2.5.10.6.3.12 SNMP_LWIP_MIB2_SYSDESC

```
#define SNMP_LWIP_MIB2_SYSDESC   "lwIP"
```

Value return for sysDesc field of MIB2.

2.5.10.6.3.13 SNMP_LWIP_MIB2_SYSLOCATION

```
#define SNMP_LWIP_MIB2_SYSLOCATION   ""
```

Value return for sysLocation field of MIB2. To make sysLocation field settable, call snmp_mib2_set_syslocation() to provide the necessary buffers.

2.5.10.6.3.14 SNMP_LWIP_MIB2_SYSNAME

```
#define SNMP_LWIP_MIB2_SYSNAME   "FQDN-unk"
```

Value return for sysName field of MIB2. To make sysName field settable, call snmp_mib2_set_sysname() to provide the necessary buffers.

2.5.10.6.3.15 SNMP_MAX_COMMUNITY_STR_LEN

```
#define SNMP_MAX_COMMUNITY_STR_LEN   LWIP_MAX(LWIP_MAX(sizeof(SNMP_COMMUNITY), sizeof(SNMP
sizeof(SNMP_COMMUNITY_TRAP))
```

The maximum length of community string. If community names shall be adjusted at runtime via snmp_set_community() calls, enter here the possible maximum length (+1 for terminating null character).

2.5.10.6.3.16 SNMP_MAX_OBJ_ID_LEN

```
#define SNMP_MAX_OBJ_ID_LEN   50
```

The maximum number of Sub ID's inside an object identifier. Indirectly this also limits the maximum depth of SNMP tree.

2.5.10.6.3.17 SNMP_MAX_OCTET_STRING_LEN

```
#define SNMP_MAX_OCTET_STRING_LEN   127
```

The maximum length of strings used.

2.5.10.6.3.18 SNMP_MAX_VALUE_SIZE

```
#define SNMP_MAX_VALUE_SIZE   LWIP_MAX(LWIP_MAX((SNMP_MAX_OCTET_STRING_LEN), sizeof(u32_t)
SNMP_MIN_VALUE_SIZE)
```

The maximum size of a value.

2.5.10.6.3.19 SNMP_MIB_DEBUG

```
#define SNMP_MIB_DEBUG   LWIP_DBG_OFF
```

SNMP_MIB_DEBUG: Enable debugging for SNMP MIBs.

2.5.10.6.3.20 SNMP_MIN_VALUE_SIZE

```
#define SNMP_MIN_VALUE_SIZE   (2 * sizeof(u32_t*)) /* size required to store the basic
types (8 bytes for counter64) */
```

The minimum size of a value.

2.5.10.6.3.21 SNMP_SAFE_REQUESTS

```
#define SNMP_SAFE_REQUESTS   1
```

Only allow SNMP write actions that are 'safe' (e.g. disabling netifs is not a safe action and disabled when SNMP_SAFE_REQUESTS = 1). Unsafe requests are disabled by default!

2.5.10.6.3.22 SNMP_STACK_SIZE

```
#define SNMP_STACK_SIZE   DEFAULT_THREAD_STACKSIZE
```

SNMP_STACK_SIZE: Stack size of SNMP netconn worker thread

2.5.10.6.3.23 SNMP_THREAD_PRIO

```
#define SNMP_THREAD_PRIO   DEFAULT_THREAD_PRIO
```

SNMP_THREAD_PRIO: SNMP netconn worker thread priority

2.5.10.6.3.24 SNMP_TRAP_DESTINATIONS

```
#define SNMP_TRAP_DESTINATIONS   1
```

SNMP_TRAP_DESTINATIONS: Number of trap destinations. At least one trap destination is required

2.5.10.6.3.25 SNMP_USE_NETCONN

```
#define SNMP_USE_NETCONN   0
```

SNMP_USE_NETCONN: Use netconn API instead of raw API. Makes SNMP agent run in a worker thread, so blocking operations can be done in MIB calls.

2.5.10.6.3.26 SNMP_USE_RAW

```
#define SNMP_USE_RAW   1
```

SNMP_USE_RAW: Use raw API. SNMP agent does not run in a worker thread, so blocking operations should not be done in MIB calls.

2.5.11 SNTP

2.5.11.1 Modules

- Options

2.5.11.2 Functions

- void sntp_init (void)
- void sntp_stop (void)
- u8_t sntp_enabled (void)
- void sntp_setoperatingmode (u8_t operating_mode)
- u8_t sntp_getoperatingmode (void)
- u8_t sntp_getreachability (u8_t idx)
- void sntp_setserver (u8_t idx, const ip_addr_t *server)
- const ip_addr_t * sntp_getserver (u8_t idx)
- u8_t sntp_getkodreceived (u8_t idx)

2.5.11.3 Detailed Description

This is simple "SNTP" client for the lwIP raw API. It is a minimal implementation of SNTPv4 as specified in RFC 4330.

You need to increase MEMP_NUM_SYS_TIMEOUT by one if you use SNTP!

For a list of some public NTP servers, see this link: http://support.ntp.org/bin/view/Servers/NTPPoolServers

2.5.11.4 Function Documentation

2.5.11.4.1 sntp_enabled()

```
u8_t sntp_enabled (void )
```

Get enabled state.

idx	the index of the NTP server

2.5.11.4.2 sntp_getkodreceived()

```
u8_t sntp_getkodreceived (u8_t idx)
```

Check if a Kiss-of-Death has been received from this server (only valid for SNTP_MAX_SERVERS > 1).

Parameters

Returns 1 if a KoD has been received, 0 if not.

2.5.11.4.3 sntp_getoperatingmode()

```
u8_t sntp_getoperatingmode (void )
```

Gets the operating mode.

2.5.11.4.4 sntp_getreachability()

```
u8_t sntp_getreachability (u8_t idx)
```

Gets the server reachability shift register as described in RFC 5905.

Parameters

idx	the index of the NTP server

2.5.11.4.5 sntp_getserver()

```
const ip_addr_t * sntp_getserver (u8_t idx)
```

Obtain one of the currently configured by IP address (or DHCP) NTP servers

Parameters

idx	the index of the NTP server

Returns IP address of the indexed NTP server or "ip_addr_any" if the NTP server has not been configured by address (or at all).

2.5.11.4.6 sntp_init()

```
void sntp_init (void )
```

Initialize this module. Send out request instantly or after SNTP_STARTUP_DELAY(_FUNC).

2.5.11.4.7 sntp_setoperatingmode()

```
void sntp_setoperatingmode (u8_t operating_mode)
```

Sets the operating mode.

Parameters

operating_mode	one of the available operating modes

idx	the index of the NTP server to set must be < SNTP_MAX_SERVERS
server	IP address of the NTP server to set

2.5.11.4.8 sntp_setserver()

```
void sntp_setserver (u8_t idx, const ip_addr_t * server)
```

Initialize one of the NTP servers by IP address

Parameters

2.5.11.4.9 sntp_stop()

```
void sntp_stop (void )
```

Stop this module.

2.5.11.5 Options

2.5.11.5.1 Macros

- #define SNTP_SET_SYSTEM_TIME(sec)   LWIP_UNUSED_ARG(sec)
- #define SNTP_MAX_SERVERS LWIP_DHCP_MAX_NTP_SERVERS
- #define SNTP_GET_SERVERS_FROM_DHCP LWIP_DHCP_GET_NTP_SRV
- #define SNTP_GET_SERVERS_FROM_DHCPV6 LWIP_DHCP6_GET_NTP_SRV
- #define SNTP_SERVER_DNS   0
- #define SNTP_DEBUG LWIP_DBG_OFF
- #define SNTP_PORT LWIP_IANA_PORT_SNTP
- #define SNTP_CHECK_RESPONSE   0
- #define SNTP_COMP_ROUNDTRIP   0
- #define SNTP_STARTUP_DELAY   1
- #define SNTP_STARTUP_DELAY_FUNC   (LWIP_RAND() % 5000)
- #define SNTP_RECV_TIMEOUT   15000
- #define SNTP_UPDATE_DELAY   3600000
- #define SNTP_GET_SYSTEM_TIME(sec, us)   do { (sec) = 0; (us) = 0; } while(0)
- #define SNTP_RETRY_TIMEOUT SNTP_RECV_TIMEOUT
- #define SNTP_RETRY_TIMEOUT_MAX   (SNTP_RETRY_TIMEOUT * 10)
- #define SNTP_RETRY_TIMEOUT_EXP   1
- #define SNTP_MONITOR_SERVER_REACHABILITY   1

2.5.11.5.2 Detailed Description

2.5.11.5.3 Macro Definition Documentation

2.5.11.5.3.1 SNTP_CHECK_RESPONSE

```
#define SNTP_CHECK_RESPONSE   0
```

Sanity check: Define this to

- 0 to turn off sanity checks (default; smaller code)
- >= 1 to check address and port of the response packet to ensure the response comes from the server we sent the request to.
- >= 2 to check returned Originate Timestamp against Transmit Timestamp sent to the server (to ensure response to older request).
- >= 3

2.5.11.5.3.2 SNTP_COMP_ROUNDTRIP

```
#define SNTP_COMP_ROUNDTRIP   0
```

Enable round-trip delay compensation. Compensate for the round-trip delay by calculating the clock offset from the originate, receive, transmit and destination timestamps, as per RFC.

The calculation requires compiler support for 64-bit integers. Also, either SNTP_SET_SYSTEM_TIME_US or SNTP_SET_SYSTEM_ has to be implemented for setting the system clock with sub-second precision. Likewise, either SNTP_GET_SYSTEM_TIME or SNTP_GET_SYSTEM_TIME_NTP needs to be implemented with sub-second precision.

Although not strictly required, it makes sense to combine this option with SNTP_CHECK_RESPONSE >= 2 for sanity-checking of the received timestamps. Also, in order for the round-trip calculation to work, the difference between the local clock and the NTP server clock must not be larger than about 34 years. If that limit is exceeded, the implementation will fall back to setting the clock without compensation. In order to ensure that the local clock is always within the permitted range for compensation, even at first try, it may be necessary to store at least the current year in non-volatile memory.

2.5.11.5.3.3 SNTP_DEBUG

```
#define SNTP_DEBUG   LWIP_DBG_OFF
```

SNTP_DEBUG: Enable debugging for SNTP.

2.5.11.5.3.4 SNTP_GET_SERVERS_FROM_DHCP

```
#define SNTP_GET_SERVERS_FROM_DHCP   LWIP_DHCP_GET_NTP_SRV
```

Set this to 1 to implement the callback function called by dhcp when NTP servers are received.

2.5.11.5.3.5 SNTP_GET_SERVERS_FROM_DHCPV6

```
#define SNTP_GET_SERVERS_FROM_DHCPV6   LWIP_DHCP6_GET_NTP_SRV
```

Set this to 1 to implement the callback function called by dhcpv6 when NTP servers are received.

2.5.11.5.3.6 SNTP_GET_SYSTEM_TIME

```
#define SNTP_GET_SYSTEM_TIME( sec, us)   do { (sec) = 0; (us) = 0; } while(0)
```

SNTP macro to get system time, used with SNTP_CHECK_RESPONSE >= 2 to send in request and compare in response. Also used for round-trip delay compensation if SNTP_COMP_ROUNDTRIP != 0. Alternatively, define SNTP_GET_SYSTEM_TIME_NTP(frac) in order to work with native NTP timestamps instead.

2.5.11.5.3.7 SNTP_MAX_SERVERS

```
#define SNTP_MAX_SERVERS   LWIP_DHCP_MAX_NTP_SERVERS
```

The maximum number of SNTP servers that can be set

2.5.11.5.3.8 SNTP_MONITOR_SERVER_REACHABILITY

```
#define SNTP_MONITOR_SERVER_REACHABILITY   1
```

Keep a reachability shift register per server Default is on to conform to RFC.

2.5.11.5.3.9 SNTP_PORT

```
#define SNTP_PORT   LWIP_IANA_PORT_SNTP
```

SNTP server port

2.5.11.5.3.10 SNTP_RECV_TIMEOUT

```
#define SNTP_RECV_TIMEOUT   15000
```

SNTP receive timeout - in milliseconds Also used as retry timeout - this shouldn't be too low. Default is 15 seconds. Must not be below 15 seconds by specification (i.e. 15000)

2.5.11.5.3.11 SNTP_RETRY_TIMEOUT

```
#define SNTP_RETRY_TIMEOUT   SNTP_RECV_TIMEOUT
```

Default retry timeout (in milliseconds) if the response received is invalid. This is doubled with each retry until SNTP_RETRY_TIMEOU
is reached.

2.5.11.5.3.12 SNTP_RETRY_TIMEOUT_EXP

```
#define SNTP_RETRY_TIMEOUT_EXP   1
```

Increase retry timeout with every retry sent Default is on to conform to RFC.

2.5.11.5.3.13 SNTP_RETRY_TIMEOUT_MAX

```
#define SNTP_RETRY_TIMEOUT_MAX   (SNTP_RETRY_TIMEOUT * 10)
```

Maximum retry timeout (in milliseconds).

2.5.11.5.3.14 SNTP_SERVER_DNS

```
#define SNTP_SERVER_DNS   0
```

Set this to 1 to support DNS names (or IP address strings) to set sntp servers One server address/name can be defined as default if SNTP_SERVER_DNS == 1: #define SNTP_SERVER_ADDRESS "pool.ntp.org"

2.5.11.5.3.15 SNTP_SET_SYSTEM_TIME

```
#define SNTP_SET_SYSTEM_TIME( sec)   LWIP_UNUSED_ARG(sec)
```

SNTP macro to change system time in seconds Define SNTP_SET_SYSTEM_TIME_US(sec, us) to set the time in microseconds instead of this one if you need the additional precision. Alternatively, define SNTP_SET_SYSTEM_TIME_NTP(sec, frac) in order to work with native NTP timestamps instead.

2.5.11.5.3.16 SNTP_STARTUP_DELAY

```
#define SNTP_STARTUP_DELAY   1
```

According to the RFC, this shall be a random delay between 1 and 5 minutes (in milliseconds) to prevent load peaks. This can be defined to a random generation function, which must return the delay in milliseconds as u32_t. Turned off by default.

2.5.11.5.3.17 SNTP_STARTUP_DELAY_FUNC

```
#define SNTP_STARTUP_DELAY_FUNC   (LWIP_RAND() % 5000)
```

If you want the startup delay to be a function, define this to a function (including the brackets) and define SNTP_STARTUP_DELAY to 1.

2.5.11.5.3.18 SNTP_UPDATE_DELAY

```
#define SNTP_UPDATE_DELAY   3600000
```

SNTP update delay - in milliseconds Default is 1 hour. Must not be below 60 seconds by specification (i.e. 60000)

2.5.12 TFTP client/server

2.5.12.1 Modules

- Options

2.5.12.2 Data Structures

- struct tftp_context

2.5.12.3 Functions

- err_t tftp_init_server (const struct tftp_context *ctx)
- err_t tftp_init_client (const struct tftp_context *ctx)
- void tftp_cleanup (void)

2.5.12.4 Detailed Description

This is simple TFTP client/server for the lwIP raw API. You need to increase MEMP_NUM_SYS_TIMEOUT by one if you use TFTP!

2.5.12.5 Function Documentation

2.5.12.5.1 tftp_cleanup()

```
void tftp_cleanup (void )
```

Deinitialize ("turn off") TFTP client/server.

2.5.12.5.2 tftp_init_client()

```
err_t tftp_init_client (const struct tftp_context * ctx)
```

Initialize TFTP client.

Parameters

ctx	TFTP callback struct

2.5.12.5.3 tftp_init_server()

```
err_t tftp_init_server (const struct tftp_context * ctx)
```

Initialize TFTP server.

Parameters

ctx	TFTP callback struct

2.5.12.6 Options

2.5.12.6.1 Macros

- #define TFTP_DEBUG LWIP_DBG_OFF
- #define TFTP_PORT LWIP_IANA_PORT_TFTP
- #define TFTP_TIMEOUT_MSECS 10000
- #define TFTP_MAX_RETRIES 5
- #define TFTP_TIMER_MSECS (TFTP_TIMEOUT_MSECS / 10)
- #define TFTP_MAX_FILENAME_LEN 20
- #define TFTP_MAX_MODE_LEN 10

2.5.12.6.2 Detailed Description

2.5.12.6.3 Macro Definition Documentation

2.5.12.6.3.1 TFTP_DEBUG

```
#define TFTP_DEBUG   LWIP_DBG_OFF
```

Enable TFTP debug messages

2.5.12.6.3.2 TFTP_MAX_FILENAME_LEN

```
#define TFTP_MAX_FILENAME_LEN   20
```

Max. length of TFTP filename

2.5.12.6.3.3 TFTP_MAX_MODE_LEN

```
#define TFTP_MAX_MODE_LEN   10
```

Max. length of TFTP mode

2.5.12.6.3.4 TFTP_MAX_RETRIES

```
#define TFTP_MAX_RETRIES   5
```

Max. number of retries when a file is read from server

2.5.12.6.3.5 TFTP_PORT

```
#define TFTP_PORT   LWIP_IANA_PORT_TFTP
```

TFTP server port

2.5.12.6.3.6 TFTP_TIMEOUT_MSECS

```
#define TFTP_TIMEOUT_MSECS   10000
```

TFTP timeout

2.5.12.6.3.7 TFTP_TIMER_MSECS

```
#define TFTP_TIMER_MSECS   (TFTP_TIMEOUT_MSECS / 10)
```

TFTP timer cyclic interval

Chapter 3

Data Structure Documentation

3.1 _lwiperf_settings Struct Reference

3.1.1 Detailed Description

This is the Iperf settings struct sent from the client

The documentation for this struct was generated from the following file: src/apps/lwiperf/lwiperf.c

3.2 _lwiperf_state_tcp Struct Reference

3.2.1 Detailed Description

Connection handle for a TCP iperf session

The documentation for this struct was generated from the following file: src/apps/lwiperf/lwiperf.c

3.3 acd Struct Reference

```
#include <acd.h>
```

3.3.1 Data Fields

- struct acd * next
- ip4_addr_t ipaddr
- acd_state_enum_t state
- u8_t sent_num
- u16_t ttw
- u8_t lastconflict
- u8_t num_conflicts
- acd_conflict_callback_t acd_conflict_callback

3.3.2 Detailed Description

ACD state information per netif

3.3.3 Field Documentation

3.3.3.1 acd_conflict_callback

`acd_conflict_callback_t acd::acd_conflict_callback`

callback function -> let’s the acd user know if the address is good or if a conflict is detected

3.3.3.2 ipaddr

`ip4_addr_t acd::ipaddr`

the currently selected, probed, announced or used IP-Address

3.3.3.3 lastconflict

`u8_t acd::lastconflict`

ticks until a conflict can again be solved by defending

3.3.3.4 next

`struct acd* acd::next`

next acd module

3.3.3.5 num_conflicts

`u8_t acd::num_conflicts`

total number of probed/used IP-Addresses that resulted in a conflict

3.3.3.6 sent_num

`u8_t acd::sent_num`

sent number of probes or announces, dependent on state

3.3.3.7 state

`acd_state_enum_t acd::state`

current ACD state machine state

3.3.3.8 ttw

`u16_t acd::ttw`

ticks to wait, tick is ACD_TMR_INTERVAL long

The documentation for this struct was generated from the following file: src/include/lwip/acd.h

3.4 altcp_allocator_s Struct Reference

```
#include <altcp.h>
```

3.4.1 Data Fields

- altcp_new_fn alloc
- void * arg

3.4.2 Detailed Description

Struct containing an allocator and its state.

3.4.3 Field Documentation

3.4.3.1 alloc

```
altcp_new_fn altcp_allocator_s::alloc
```

Allocator function

3.4.3.2 arg

```
void* altcp_allocator_s::arg
```

Argument to allocator function

The documentation for this struct was generated from the following file: src/include/lwip/altcp.h

3.5 api_msg Struct Reference

```
#include <api_msg.h>
```

3.5.1 Data Fields

- struct netconn * conn
- err_t err
- union {
  - struct netbuf * b
  - struct {
  - } n
  - struct {
  - } bc
  - struct {
  - } ad
  - struct {
    - const struct netvector * vector

        * u16_t vector_cnt
        * size_t vector_off
        * size_t len
        * size_t offset
    - } w
    - struct {
    - } r
    - struct {
    - } sd
    - struct {
    - } jl
- } msg

3.5.2 Detailed Description

This struct includes everything that is necessary to execute a function for a netconn in another thread context (mainly used to process netconns in the tcpip_thread context to be thread safe).

3.5.3 Field Documentation

3.5.3.1 [struct]

```
struct { ...  } api_msg::ad
```

used for lwip_netconn_do_getaddr

3.5.3.2 b

```
struct netbuf* api_msg::b
```

used for lwip_netconn_do_send

3.5.3.3 [struct]

```
struct { ...  } api_msg::bc
```

used for lwip_netconn_do_bind and lwip_netconn_do_connect

3.5.3.4 conn

```
struct netconn* api_msg::conn
```

The netconn which to process - always needed: it includes the semaphore which is used to block the application thread until the function finished.

3.5.3.5 err

```
err_t api_msg::err
```

The return value of the function executed in tcpip_thread.

3.5.3.6 [struct]

`struct { ... } api_msg::jl`

used for lwip_netconn_do_join_leave_group

3.5.3.7 len

`size_t api_msg::len`

total length across vectors

3.5.3.8 [union]

`union { ... } api_msg::msg`

Depending on the executed function, one of these union members is used

3.5.3.9 [struct]

`struct { ... } api_msg::n`

used for lwip_netconn_do_newconn

3.5.3.10 offset

`size_t api_msg::offset`

offset into total length/output of bytes written when err == ERR_OK

3.5.3.11 [struct]

`struct { ... } api_msg::r`

used for lwip_netconn_do_recv

3.5.3.12 [struct]

`struct { ... } api_msg::sd`

used for lwip_netconn_do_close (/shutdown)

3.5.3.13 vector

`const struct netvector* api_msg::vector`

current vector to write

3.5.3.14 vector_cnt

`u16_t api_msg::vector_cnt`

number of unwritten vectors

3.5.3.15 vector_off

```
size_t api_msg::vector_off
```

offset into current vector

3.5.3.16 [struct]

```
struct { ... } api_msg::w
```

used for lwip_netconn_do_write

The documentation for this struct was generated from the following file: src/include/lwip/priv/api_msg.h

3.6 autoip Struct Reference

```
#include <autoip.h>
```

3.6.1 Data Fields

- ip4_addr_t llipaddr
- u8_t state
- u8_t tried_llipaddr
- struct acd acd

3.6.2 Detailed Description

AutoIP state information per netif

3.6.3 Field Documentation

3.6.3.1 acd

```
struct acd autoip::acd
```

acd struct

3.6.3.2 llipaddr

```
ip4_addr_t autoip::llipaddr
```

the currently selected, probed, announced or used LL IP-Address

3.6.3.3 state

```
u8_t autoip::state
```

current AutoIP state machine state

3.6.3.4 tried_llipaddr

`u8_t autoip::tried_llipaddr`

total number of probed/used Link Local IP-Addresses

The documentation for this struct was generated from the following file: src/include/lwip/autoip.h

3.7 bridgeif_initdata_s Struct Reference

`#include <bridgeif.h>`

3.7.1 Data Fields

- struct eth_addr ethaddr
- u8_t max_ports
- u16_t max_fdb_dynamic_entries
- u16_t max_fdb_static_entries

3.7.2 Detailed Description

Initialisation data for bridgeif_init. An instance of this type must be passed as parameter 'state' to netif_add when the bridge is added.

3.7.3 Field Documentation

3.7.3.1 ethaddr

`struct eth_addr bridgeif_initdata_s::ethaddr`

MAC address of the bridge (cannot use the netif's addresses)

3.7.3.2 max_fdb_dynamic_entries

`u16_t bridgeif_initdata_s::max_fdb_dynamic_entries`

Maximum number of dynamic/learning entries in the bridge's forwarding database. In the default implementation, this controls memory consumption only.

3.7.3.3 max_fdb_static_entries

`u16_t bridgeif_initdata_s::max_fdb_static_entries`

Maximum number of static forwarding entries. Influences memory consumption!

3.7.3.4 max_ports

`u8_t bridgeif_initdata_s::max_ports`

Maximum number of ports in the bridge (ports are stored in an array, this influences memory allocated for netif->state of the bridge netif).

The documentation for this struct was generated from the following file: src/include/netif/bridgeif.h

3.8 dhcp6_msg Struct Reference

```
#include <dhcp6.h>
```

3.8.1 Detailed Description

minimum set of fields of any DHCPv6 message

The documentation for this struct was generated from the following file: src/include/lwip/prot/dhcp6.h

3.9 dhcp_msg Struct Reference

```
#include <dhcp.h>
```

3.9.1 Detailed Description

minimum set of fields of any DHCP message

The documentation for this struct was generated from the following file: src/include/lwip/prot/dhcp.h

3.10 dns_answer Struct Reference

3.10.1 Detailed Description

DNS answer message structure. No packing needed: only used locally on the stack.

The documentation for this struct was generated from the following file: src/core/dns.c

3.11 dns_api_msg Struct Reference

```
#include <api_msg.h>
```

3.11.1 Data Fields

- const char * name
- ip_addr_t * addr
- u8_t dns_addrtype
- sys_sem_t * sem
- err_t * err

3.11.2 Detailed Description

As lwip_netconn_do_gethostbyname requires more arguments but doesn't require a netconn, it has its own struct (to avoid struct api_msg getting bigger than necessary). lwip_netconn_do_gethostbyname must be called using tcpip_callback instead of tcpip_apimsg (see netconn_gethostbyname).

3.11.3 Field Documentation

3.11.3.1 addr

`ip_addr_t* dns_api_msg::addr`

The resolved address is stored here

3.11.3.2 dns_addrtype

`u8_t dns_api_msg::dns_addrtype`

Type of resolve call

3.11.3.3 err

`err_t* dns_api_msg::err`

Errors are given back here

3.11.3.4 name

`const char* dns_api_msg::name`

Hostname to query or dotted IP address string

3.11.3.5 sem

`sys_sem_t* dns_api_msg::sem`

This semaphore is posted when the name is resolved, the application thread should wait on it.

The documentation for this struct was generated from the following file: src/include/lwip/priv/api_msg.h

3.12 dns_hdr Struct Reference

`#include <dns.h>`

3.12.1 Detailed Description

DNS message header

The documentation for this struct was generated from the following file: src/include/lwip/prot/dns.h

3.13 dns_query Struct Reference

3.13.1 Detailed Description

DNS query message structure. No packing needed: only used locally on the stack.

The documentation for this struct was generated from the following file: src/core/dns.c

3.14 dns_req_entry Struct Reference

3.14.1 Detailed Description

DNS request table entry: used when dns_gehostbyname cannot answer the request from the DNS table

The documentation for this struct was generated from the following file: src/core/dns.c

3.15 dns_table_entry Struct Reference

3.15.1 Detailed Description

DNS table entry

The documentation for this struct was generated from the following file: src/core/dns.c

3.16 eth_addr Struct Reference

```
#include <ethernet.h>
```

3.16.1 Detailed Description

An Ethernet MAC address

The documentation for this struct was generated from the following file: src/include/lwip/prot/ethernet.h

3.17 eth_hdr Struct Reference

```
#include <ethernet.h>
```

3.17.1 Detailed Description

Ethernet header

The documentation for this struct was generated from the following file: src/include/lwip/prot/ethernet.h

3.18 eth_vlan_hdr Struct Reference

```
#include <ethernet.h>
```

3.18.1 Detailed Description

VLAN header inserted between ethernet header and payload if 'type' in ethernet header is ETHTYPE_VLAN. See IEEE802.Q

The documentation for this struct was generated from the following file: src/include/lwip/prot/ethernet.h

3.19 etharp_hdr Struct Reference

```
#include <etharp.h>
```

3.19.1 Detailed Description

the ARP message, see RFC 826 ("Packet format")

The documentation for this struct was generated from the following file: src/include/lwip/prot/etharp.h

3.20 etharp_q_entry Struct Reference

```
#include <etharp.h>
```

3.20.1 Detailed Description

struct for queueing outgoing packets for unknown address defined here to be accessed by memp.h

The documentation for this struct was generated from the following file: src/include/lwip/etharp.h

3.21 gethostbyname_r_helper Struct Reference

3.21.1 Detailed Description

helper struct for gethostbyname_r to access the char* buffer

The documentation for this struct was generated from the following file: src/api/netdb.c

3.22 icmp6_echo_hdr Struct Reference

```
#include <icmp6.h>
```

3.22.1 Detailed Description

This is the ICMP6 header adapted for echo req/resp.

The documentation for this struct was generated from the following file: src/include/lwip/prot/icmp6.h

3.23 icmp6_hdr Struct Reference

```
#include <icmp6.h>
```

3.23.1 Detailed Description

This is the standard ICMP6 header.

The documentation for this struct was generated from the following file: src/include/lwip/prot/icmp6.h

3.24 icmp_echo_hdr Struct Reference

```
#include <icmp.h>
```

3.24.1 Detailed Description

This is the standard ICMP header only that the u32_t data is split to two u16_t like ICMP echo needs it.

The documentation for this struct was generated from the following file: src/include/lwip/prot/icmp.h

3.25 icmp_hdr Struct Reference

```
#include <icmp.h>
```

3.25.1 Detailed Description

The standard ICMP header (unspecified 32 bit data)

The documentation for this struct was generated from the following file: src/include/lwip/prot/icmp.h

3.26 ieee_802154_hdr Struct Reference

```
#include <ieee802154.h>
```

3.26.1 Data Fields

- u16_t frame_control
- u8_t sequence_number
- u16_t destination_pan_id
- u8_t destination_address [8]
- u16_t source_pan_id
- u8_t source_address [8]

3.26.2 Detailed Description

General MAC frame format This shows the full featured header, mainly for documentation. Some fields are omitted or shortened to achieve frame compression.

3.26.3 Field Documentation

3.26.3.1 destination_address

```
u8_t ieee_802154_hdr::destination_address[8]
```

Destination Address is omitted if Destination Addressing Mode is 0

3.26.3.2 destination_pan_id

```
u16_t ieee_802154_hdr::destination_pan_id
```

Destination PAN ID is omitted if Destination Addressing Mode is 0

3.26.3.3 frame_control

```
u16_t ieee_802154_hdr::frame_control
```

See IEEE_802154_FC_* defines

3.26.3.4 sequence_number

```
u8_t ieee_802154_hdr::sequence_number
```

Sequence number is omitted if IEEE_802154_FC_SEQNO_SUPPR is set in frame_control

3.26.3.5 source_address

```
u8_t ieee_802154_hdr::source_address[8]
```

Source Address is omitted if Source Addressing Mode is 0

3.26.3.6 source_pan_id

```
u16_t ieee_802154_hdr::source_pan_id
```

Source PAN ID is omitted if Source Addressing Mode is 0 or if IEEE_802154_FC_PANID_COMPR is set in frame control

The documentation for this struct was generated from the following file: src/include/netif/ieee802154.h

3.27 igmp_group Struct Reference

```
#include <igmp.h>
```

3.27.1 Data Fields

- struct igmp_group * next
- ip4_addr_t group_address
- u8_t last_reporter_flag
- u8_t group_state
- u16_t timer
- u8_t use

3.27.2 Detailed Description

igmp group structure - there is a list of groups for each interface these should really be linked from the interface, but if we keep them separate we will not affect the lwip original code too much

There will be a group for the all systems group address but this will not run the state machine as it is used to kick off reports from all the other groups

3.27.3 Field Documentation

3.27.3.1 group_address

```
ip4_addr_t igmp_group::group_address
```

multicast address

3.27.3.2 group_state

```
u8_t igmp_group::group_state
```

current state of the group

3.27.3.3 last_reporter_flag

```
u8_t igmp_group::last_reporter_flag
```

signifies we were the last person to report

3.27.3.4 next

```
struct igmp_group* igmp_group::next
```

next link

3.27.3.5 timer

```
u16_t igmp_group::timer
```

timer for reporting, negative is OFF

3.27.3.6 use

```
u8_t igmp_group::use
```

counter of simultaneous uses

The documentation for this struct was generated from the following file: src/include/lwip/igmp.h

3.28 igmp_msg Struct Reference

```
#include <igmp.h>
```

3.28.1 Detailed Description

IGMP packet format.

The documentation for this struct was generated from the following file: src/include/lwip/prot/igmp.h

3.29 ip4_addr Struct Reference

```
#include <ip4_addr.h>
```

3.29.1 Detailed Description

This is the aligned version of ip4_addr_t, used as local variable, on the stack, etc.

The documentation for this struct was generated from the following file: src/include/lwip/ip4_addr.h

3.30 ip4_addr_packed Struct Reference

```
#include <ip4.h>
```

3.30.1 Detailed Description

This is the packed version of ip4_addr_t, used in network headers that are itself packed

The documentation for this struct was generated from the following file: src/include/lwip/prot/ip4.h

3.31 ip4_addr_wordaligned Struct Reference

```
#include <etharp.h>
```

3.31.1 Detailed Description

struct ip4_addr_wordaligned is used in the definition of the ARP packet format in order to support compilers that don't have structure packing.

The documentation for this struct was generated from the following file: src/include/lwip/prot/etharp.h

3.32 ip6_addr Struct Reference

```
#include <ip6_addr.h>
```

3.32.1 Detailed Description

This is the aligned version of ip6_addr_t, used as local variable, on the stack, etc.

The documentation for this struct was generated from the following file: src/include/lwip/ip6_addr.h

3.33 ip6_addr_packed Struct Reference

```
#include <ip6.h>
```

3.33.1 Detailed Description

This is the packed version of ip6_addr_t, used in network headers that are itself packed

The documentation for this struct was generated from the following file: src/include/lwip/prot/ip6.h

3.34 ip6_hdr Struct Reference

```
#include <ip6.h>
```

3.34.1 Data Fields

- u32_t _v_tc_fl
- u16_t _plen
- u8_t _nexth
- u8_t _hoplim
- ip6_addr_p_t src

3.34.2 Detailed Description

The IPv6 header.

3.34.3 Field Documentation

3.34.3.1 _hoplim

`u8_t ip6_hdr::_hoplim`

hop limit

3.34.3.2 _nexth

`u8_t ip6_hdr::_nexth`

next header

3.34.3.3 _plen

`u16_t ip6_hdr::_plen`

payload length

3.34.3.4 _v_tc_fl

`u32_t ip6_hdr::_v_tc_fl`

version / traffic class / flow label

3.34.3.5 src

`ip6_addr_p_t ip6_hdr::src`

source and destination IP addresses

The documentation for this struct was generated from the following file: src/include/lwip/prot/ip6.h

3.35 ip6_reass_helper Struct Reference

3.35.1 Detailed Description

This is a helper struct which holds the starting offset and the ending offset of this fragment to easily chain the fragments. It has the same packing requirements as the IPv6 header, since it replaces the Fragment Header in memory in incoming fragments to keep track of the various fragments.

The documentation for this struct was generated from the following file: src/core/ipv6/ip6_frag.c

3.36 ip6_reassdata Struct Reference

```
#include <ip6_frag.h>
```

3.36.1 Detailed Description

IPv6 reassembly helper struct. This is exported because memp needs to know the size.

The documentation for this struct was generated from the following file: src/include/lwip/ip6_frag.h

3.37 ip_addr Struct Reference

```
#include <ip_addr.h>
```

3.37.1 Data Fields

- u8_t type

3.37.2 Detailed Description

A union struct for both IP version's addresses. ATTENTION: watch out for its size when adding IPv6 address scope!

3.37.3 Field Documentation

3.37.3.1 type

```
u8_t ip_addr::type
```

lwip_ip_addr_type

The documentation for this struct was generated from the following file: src/include/lwip/ip_addr.h

3.38 ip_globals Struct Reference

```
#include <ip.h>
```

3.38.1 Data Fields

- struct netif * current_netif
- struct netif * current_input_netif
- const struct ip_hdr * current_ip4_header
- struct ip6_hdr * current_ip6_header
- u16_t current_ip_header_tot_len
- ip_addr_t current_iphdr_src
- ip_addr_t current_iphdr_dest

3.38.2 Detailed Description

Global variables of this module, kept in a struct for efficient access using base+index.

3.38.3 Field Documentation

3.38.3.1 current_input_netif

```
struct netif* ip_globals::current_input_netif
```

The interface that received the packet for the current callback invocation.

3.38.3.2 current_ip4_header

```
const struct ip_hdr* ip_globals::current_ip4_header
```

Header of the input packet currently being processed.

3.38.3.3 current_ip6_header

```
struct ip6_hdr* ip_globals::current_ip6_header
```

Header of the input IPv6 packet currently being processed.

3.38.3.4 current_ip_header_tot_len

```
u16_t ip_globals::current_ip_header_tot_len
```

Total header length of current_ip4/6_header (i.e. after this, the UDP/TCP header starts)

3.38.3.5 current_iphdr_dest

```
ip_addr_t ip_globals::current_iphdr_dest
```

Destination IP address of current_header

3.38.3.6 current_iphdr_src

```
ip_addr_t ip_globals::current_iphdr_src
```

Source IP address of current_header

3.38.3.7 current_netif

```
struct netif* ip_globals::current_netif
```

The interface that accepted the packet for the current callback invocation.

The documentation for this struct was generated from the following file: src/include/lwip/ip.h

3.39 ip_reass_helper Struct Reference

3.39.1 Detailed Description

This is a helper struct which holds the starting offset and the ending offset of this fragment to easily chain the fragments. It has the same packing requirements as the IP header, since it replaces the IP header in memory in incoming fragments (after copying it) to keep track of the various fragments. (-> If the IP header doesn't need packing, this struct doesn't need packing, too.)

The documentation for this struct was generated from the following file: src/core/ipv4/ip4_frag.c

3.40 ip_reassdata Struct Reference

```
#include <ip4_frag.h>
```

3.40.1 Detailed Description

IP reassembly helper struct. This is exported because memp needs to know the size.

The documentation for this struct was generated from the following file: src/include/lwip/ip4_frag.h

3.41 netif_ext_callback_args_t::ipv4_changed_s Struct Reference

```
#include <netif.h>
```

3.41.1 Data Fields

- const ip_addr_t * old_address

3.41.2 Detailed Description

Args to LWIP_NSC_IPV4_ADDRESS_CHANGED|LWIP_NSC_IPV4_GATEWAY_CHANGED|LWIP_NSC_IPV4_NETMASK_CH
callback

3.41.3 Field Documentation

3.41.3.1 old_address

```
const ip_addr_t* netif_ext_callback_args_t::ipv4_changed_s::old_address
```

Old IPv4 address

The documentation for this struct was generated from the following file: src/include/lwip/netif.h

3.42 netif_ext_callback_args_t::ipv6_addr_state_changed_s Struct Reference

```
#include <netif.h>
```

3.42.1 Data Fields

- s8_t addr_index
- u8_t old_state
- const ip_addr_t * address

3.42.2 Detailed Description

Args to LWIP_NSC_IPV6_ADDR_STATE_CHANGED callback

3.42.3 Field Documentation

3.42.3.1 addr_index

`s8_t netif_ext_callback_args_t::ipv6_addr_state_changed_s::addr_index`

Index of affected IPv6 address

3.42.3.2 address

`const ip_addr_t* netif_ext_callback_args_t::ipv6_addr_state_changed_s::address`

Affected IPv6 address

3.42.3.3 old_state

`u8_t netif_ext_callback_args_t::ipv6_addr_state_changed_s::old_state`

Old IPv6 address state

The documentation for this struct was generated from the following file: src/include/lwip/netif.h

3.43 netif_ext_callback_args_t::ipv6_set_s Struct Reference

`#include <netif.h>`

3.43.1 Data Fields

- s8_t addr_index
- const ip_addr_t * old_address

3.43.2 Detailed Description

Args to LWIP_NSC_IPV6_SET callback

3.43.3 Field Documentation

3.43.3.1 addr_index

`s8_t netif_ext_callback_args_t::ipv6_set_s::addr_index`

Index of changed IPv6 address

3.43.3.2 old_address

`const ip_addr_t* netif_ext_callback_args_t::ipv6_set_s::old_address`

Old IPv6 address

The documentation for this struct was generated from the following file: src/include/lwip/netif.h

3.44 netif_ext_callback_args_t::link_changed_s Struct Reference

```
#include <netif.h>
```

3.44.1 Data Fields

- u8_t state

3.44.2 Detailed Description

Args to LWIP_NSC_LINK_CHANGED callback

3.44.3 Field Documentation

3.44.3.1 state

```
u8_t netif_ext_callback_args_t::link_changed_s::state
```

1: up; 0: down

The documentation for this struct was generated from the following file: src/include/lwip/netif.h

3.45 lowpan6_ieee802154_data Struct Reference

3.45.1 Data Fields

- struct lowpan6_reass_helper * reass_list
- ip6_addr_t lowpan6_context [10]
- u16_t tx_datagram_tag
- u16_t ieee_802154_pan_id
- u8_t tx_frame_seq_num

3.45.2 Detailed Description

This struct keeps track of per-netif state

3.45.3 Field Documentation

3.45.3.1 ieee_802154_pan_id

```
u16_t lowpan6_ieee802154_data::ieee_802154_pan_id
```

local PAN ID for IEEE 802.15.4 header

3.45.3.2 lowpan6_context

```
ip6_addr_t lowpan6_ieee802154_data::lowpan6_context[10]
```

address context for compression

3.45.3.3 reass_list

`struct lowpan6_reass_helper* lowpan6_ieee802154_data::reass_list`

fragment reassembly list

3.45.3.4 tx_datagram_tag

`u16_t lowpan6_ieee802154_data::tx_datagram_tag`

Datagram Tag for fragmentation

3.45.3.5 tx_frame_seq_num

`u8_t lowpan6_ieee802154_data::tx_frame_seq_num`

Sequence Number for IEEE 802.15.4 transmission

The documentation for this struct was generated from the following file: src/netif/lowpan6.c

3.46 lowpan6_link_addr Struct Reference

```
#include <lowpan6_common.h>
```

3.46.1 Detailed Description

Helper define for a link layer address, which can be encoded as 0, 2 or 8 bytes

The documentation for this struct was generated from the following file: src/include/netif/lowpan6_common.h

3.47 lowpan6_reass_helper Struct Reference

3.47.1 Detailed Description

This is a helper struct for reassembly of fragments (IEEE 802.15.4 limits to 127 bytes)

The documentation for this struct was generated from the following file: src/netif/lowpan6.c

3.48 lwip_cyclic_timer Struct Reference

```
#include <timeouts.h>
```

3.48.1 Detailed Description

This struct contains information about a stack-internal timer function that has to be called at a defined interval

The documentation for this struct was generated from the following file: src/include/lwip/timeouts.h

3.49 lwip_select_cb Struct Reference

```
#include <sockets_priv.h>
```

3.49.1 Data Fields

- struct lwip_select_cb * next
- struct lwip_select_cb * prev
- fd_set * readset
- fd_set * writeset
- fd_set * exceptset
- struct pollfd * poll_fds
- nfds_t poll_nfds
- int sem_signalled
- sys_sem_t sem

3.49.2 Detailed Description

Description for a task waiting in select

3.49.3 Field Documentation

3.49.3.1 exceptset

```
fd_set* lwip_select_cb::exceptset
```

unimplemented: exceptset passed to select

3.49.3.2 next

```
struct lwip_select_cb* lwip_select_cb::next
```

Pointer to the next waiting task

3.49.3.3 poll_fds

```
struct pollfd* lwip_select_cb::poll_fds
```

fds passed to poll; NULL if select

3.49.3.4 poll_nfds

```
nfds_t lwip_select_cb::poll_nfds
```

nfds passed to poll; 0 if select

3.49.3.5 prev

```
struct lwip_select_cb* lwip_select_cb::prev
```

Pointer to the previous waiting task

3.49.3.6 readset

```
fd_set* lwip_select_cb::readset
```

readset passed to select

3.49.3.7 sem

```
sys_sem_t lwip_select_cb::sem
```

semaphore to wake up a task waiting for select

3.49.3.8 sem_signalled

```
int lwip_select_cb::sem_signalled
```

don't signal the same semaphore twice: set to 1 when signalled

3.49.3.9 writeset

```
fd_set* lwip_select_cb::writeset
```

writeset passed to select

The documentation for this struct was generated from the following file: src/include/lwip/priv/sockets_priv.h

3.50 lwip_sock Struct Reference

```
#include <sockets_priv.h>
```

3.50.1 Data Fields

- struct netconn * conn
- union lwip_sock_lastdata lastdata
- s16_t rcvevent
- u16_t sendevent
- u16_t errevent
- u8_t select_waiting

3.50.2 Detailed Description

Contains all internal pointers and states used for a socket

3.50.3 Field Documentation

3.50.3.1 conn

```
struct netconn* lwip_sock::conn
```

sockets currently are built on netconns, each socket has one netconn

3.50.3.2 errevent

```
u16_t lwip_sock::errevent
```

error happened for this socket, set by event_callback(), tested by select

3.50.3.3 lastdata

```
union lwip_sock_lastdata lwip_sock::lastdata
```

data that was left from the previous read

3.50.3.4 rcvevent

```
s16_t lwip_sock::rcvevent
```

number of times data was received, set by event_callback(), tested by the receive and select functions

3.50.3.5 select_waiting

```
u8_t lwip_sock::select_waiting
```

counter of how many threads are waiting for this socket using select

3.50.3.6 sendevent

```
u16_t lwip_sock::sendevent
```

number of times data was ACKed (free send buffer), set by event_callback(), tested by select

The documentation for this struct was generated from the following file: src/include/lwip/priv/sockets_priv.h

3.51 mdns_delayed_msg Struct Reference

```
#include <mdns_priv.h>
```

3.51.1 Data Fields

- u8_t multicast_msg_waiting
- u8_t multicast_timeout
- u8_t multicast_probe_timeout
- struct mdns_outmsg delayed_msg_multicast
- u8_t multicast_timeout_25TTL
- u8_t unicast_msg_in_use
- struct mdns_outmsg delayed_msg_unicast

3.51.2 Detailed Description

Delayed msg info

3.51.3 Field Documentation

3.51.3.1 delayed_msg_multicast

`struct mdns_outmsg mdns_delayed_msg::delayed_msg_multicast`

Output msg used for delayed multicast responses

3.51.3.2 delayed_msg_unicast

`struct mdns_outmsg mdns_delayed_msg::delayed_msg_unicast`

Output msg used for delayed unicast responses

3.51.3.3 multicast_msg_waiting

`u8_t mdns_delayed_msg::multicast_msg_waiting`

Signals if a multicast msg needs to be send out

3.51.3.4 multicast_probe_timeout

`u8_t mdns_delayed_msg::multicast_probe_timeout`

Multicast timeout only for probe answers

3.51.3.5 multicast_timeout

`u8_t mdns_delayed_msg::multicast_timeout`

Multicast timeout for all multicast traffic except probe answers

3.51.3.6 multicast_timeout_25TTL

`u8_t mdns_delayed_msg::multicast_timeout_25TTL`

Prefer multicast over unicast timeout -> 25% of TTL = we take 30s as general delay.

3.51.3.7 unicast_msg_in_use

`u8_t mdns_delayed_msg::unicast_msg_in_use`

Only send out new unicast message if previous was send

The documentation for this struct was generated from the following file: src/include/lwip/apps/mdns_priv.h

3.52 mdns_host Struct Reference

`#include <mdns_priv.h>`

3.52.1 Data Fields

- char name [63+1]
- struct mdns_service * services [1]
- u8_t sent_num
- mdns_resp_state_enum_t state
- struct mdns_delayed_msg ipv4
- struct mdns_delayed_msg ipv6
- u32_t conflict_time [15]
- u8_t rate_limit_activated
- u8_t index
- u8_t num_conflicts

3.52.2 Detailed Description

Description of a host/netif

3.52.3 Field Documentation

3.52.3.1 conflict_time

```
u32_t mdns_host::conflict_time[15]
```

Timestamp of probe conflict saved in list

3.52.3.2 index

```
u8_t mdns_host::index
```

List index for timestamps

3.52.3.3 ipv4

```
struct mdns_delayed_msg mdns_host::ipv4
```

delayed msg struct for IPv4

3.52.3.4 ipv6

```
struct mdns_delayed_msg mdns_host::ipv6
```

delayed msg struct for IPv6

3.52.3.5 name

```
char mdns_host::name[63+1]
```

Hostname

3.52.3.6 num_conflicts

```
u8_t mdns_host::num_conflicts
```

number of conflicts since startup

3.52.3.7 rate_limit_activated

```
u8_t mdns_host::rate_limit_activated
```

Rate limit flag

3.52.3.8 sent_num

```
u8_t mdns_host::sent_num
```

Number of probes/announces sent for the current name

3.52.3.9 services

```
struct mdns_service* mdns_host::services[1]
```

Pointer to services

3.52.3.10 state

```
mdns_resp_state_enum_t mdns_host::state
```

State of the mdns responder

The documentation for this struct was generated from the following file: src/include/lwip/apps/mdns_priv.h

3.53 mdns_outmsg Struct Reference

```
#include <mdns_priv.h>
```

3.53.1 Data Fields

- u16_t tx_id
- u8_t flags
- ip_addr_t dest_addr
- u8_t cache_flush
- u8_t unicast_reply_requested
- u8_t legacy_query
- u8_t probe_query_recv
- struct mdns_request * query

3.53.2 Detailed Description

mDNS output message

3.53.3 Field Documentation

3.53.3.1 cache_flush

`u8_t mdns_outmsg::cache_flush`

If all answers in packet should set cache_flush bit

3.53.3.2 dest_addr

`ip_addr_t mdns_outmsg::dest_addr`

Destination IP/port if sent unicast

3.53.3.3 flags

`u8_t mdns_outmsg::flags`

dns flags

3.53.3.4 legacy_query

`u8_t mdns_outmsg::legacy_query`

If legacy query. (tx_id needed, and write question again in reply before answer)

3.53.3.5 probe_query_recv

`u8_t mdns_outmsg::probe_query_recv`

If the query is a probe msg we need to respond immediately. Independent of the QU or QM flag.

3.53.3.6 query

`struct mdns_request* mdns_outmsg::query`

Search query to send

3.53.3.7 tx_id

`u16_t mdns_outmsg::tx_id`

Identifier. Used in legacy queries

3.53.3.8 unicast_reply_requested

`u8_t mdns_outmsg::unicast_reply_requested`

If reply should be sent unicast (as requested)

The documentation for this struct was generated from the following file: src/include/lwip/apps/mdns_priv.h

3.54 mdns_outpacket Struct Reference

`#include <mdns_priv.h>`

3.54.1 Data Fields

- struct pbuf * pbuf
- u16_t write_offset
- u16_t questions
- u16_t answers
- u16_t authoritative
- u16_t additional
- u16_t domain_offsets [10]

3.54.2 Detailed Description

mDNS output packet

3.54.3 Field Documentation

3.54.3.1 additional

`u16_t mdns_outpacket::additional`

Number of additional answers written

3.54.3.2 answers

`u16_t mdns_outpacket::answers`

Number of normal answers written

3.54.3.3 authoritative

`u16_t mdns_outpacket::authoritative`

Number of authoritative answers written

3.54.3.4 domain_offsets

`u16_t mdns_outpacket::domain_offsets[10]`

Offsets for written domain names in packet. Used for compression

3.54.3.5 pbuf

`struct pbuf* mdns_outpacket::pbuf`

Packet data

3.54.3.6 questions

`u16_t mdns_outpacket::questions`

Number of questions written

3.54.3.7 write_offset

`u16_t mdns_outpacket::write_offset`

Current write offset in packet

The documentation for this struct was generated from the following file: src/include/lwip/apps/mdns_priv.h

3.55 mdns_packet Struct Reference

3.55.1 Data Fields

- ip_addr_t source_addr
- u16_t recv_unicast
- struct pbuf * pbuf
- u16_t parse_offset
- u16_t tx_id
- u16_t questions
- u16_t questions_left
- u16_t answers
- u16_t answers_left
- u16_t authoritative
- u16_t authoritative_left
- u16_t additional
- u16_t additional_left
- struct mdns_packet * next_answer
- struct mdns_packet * next_tc_question

3.55.2 Detailed Description

Information about received packet

3.55.3 Field Documentation

3.55.3.1 additional

`u16_t mdns_packet::additional`

Number of additional answers in packet

3.55.3.2 additional_left

`u16_t mdns_packet::additional_left`

Number of unparsed additional answers

3.55.3.3 answers

`u16_t mdns_packet::answers`

Number of answers in packet

3.55.3.4 answers_left

`u16_t mdns_packet::answers_left`

Number of unparsed answers

3.55.3.5 authoritative

`u16_t mdns_packet::authoritative`

Number of authoritative answers in packet

3.55.3.6 authoritative_left

`u16_t mdns_packet::authoritative_left`

Number of unparsed authoritative answers

3.55.3.7 next_answer

`struct mdns_packet* mdns_packet::next_answer`

Chained list of known answer received after a truncated question

3.55.3.8 next_tc_question

`struct mdns_packet* mdns_packet::next_tc_question`

Chained list of truncated question that are waiting

3.55.3.9 parse_offset

`u16_t mdns_packet::parse_offset`

Current parsing offset in packet

3.55.3.10 pbuf

`struct pbuf* mdns_packet::pbuf`

Packet data

3.55.3.11 questions

`u16_t mdns_packet::questions`

Number of questions in packet, read from packet header

3.55.3.12 questions_left

`u16_t mdns_packet::questions_left`

Number of unparsed questions

3.55.3.13 recv_unicast

`u16_t mdns_packet::recv_unicast`

If packet was received unicast

3.55.3.14 source_addr

`ip_addr_t mdns_packet::source_addr`

Sender IP/port

3.55.3.15 tx_id

`u16_t mdns_packet::tx_id`

Identifier. Used in legacy queries

The documentation for this struct was generated from the following file: src/apps/mdns/mdns.c

3.56 mdns_request Struct Reference

`#include <mdns_priv.h>`

3.56.1 Data Fields

- char name [63+1]
- struct mdns_domain service
- search_result_fn_t result_fn
- u16_t proto
- u8_t qtype
- u16_t only_ptr

3.56.2 Detailed Description

Description of a search request

3.56.3 Field Documentation

3.56.3.1 name

`char mdns_request::name[63+1]`

Name of service, like ’myweb’

3.56.3.2 only_ptr

`u16_t mdns_request::only_ptr`

PTR only request.

3.56.3.3 proto

`u16_t mdns_request::proto`

Protocol, TCP or UDP

3.56.3.4 qtype

`u8_t mdns_request::qtype`

Query type (PTR, SRV, ...)

3.56.3.5 result_fn

`search_result_fn_t mdns_request::result_fn`

Callback function called for each response

3.56.3.6 service

`struct mdns_domain mdns_request::service`

Type of service, like '_http' or '_services._dns-sd'

The documentation for this struct was generated from the following file: src/include/lwip/apps/mdns_priv.h

3.57 mdns_rr_info Struct Reference

`#include <mdns.h>`

3.57.1 Detailed Description

Domain, type and class. Shared between questions and answers

The documentation for this struct was generated from the following file: src/include/lwip/apps/mdns.h

3.58 mdns_service Struct Reference

`#include <mdns_priv.h>`

3.58.1 Data Fields

- struct mdns_domain txtdata
- char name [63+1]
- char service [63+1]
- service_get_txt_fn_t txt_fn
- u16_t proto
- u16_t port

3.58.2 Detailed Description

Description of a service

3.58.3 Field Documentation

3.58.3.1 name

`char mdns_service::name[63+1]`

Name of service, like ’myweb’

3.58.3.2 port

`u16_t mdns_service::port`

Port of the service

3.58.3.3 proto

`u16_t mdns_service::proto`

Protocol, TCP or UDP

3.58.3.4 service

`char mdns_service::service[63+1]`

Type of service, like ’_http’

3.58.3.5 txt_fn

`service_get_txt_fn_t mdns_service::txt_fn`

Callback function and userdata to update txtdata buffer

3.58.3.6 txtdata

`struct mdns_domain mdns_service::txtdata`

TXT record to answer with

The documentation for this struct was generated from the following file: src/include/lwip/apps/mdns_priv.h

3.59 mem Struct Reference

3.59.1 Data Fields

- mem_size_t next
- mem_size_t prev
- u8_t used

3.59.2 Detailed Description

The heap is made up as a list of structs of this type. This does not have to be aligned since for getting its size, we only use the macro SIZEOF_STRUCT_MEM, which automatically aligns.

3.59.3 Field Documentation

3.59.3.1 next

```
mem_size_t mem::next
```

index (-> ram[next]) of the next struct

3.59.3.2 prev

```
mem_size_t mem::prev
```

index (-> ram[prev]) of the previous struct

3.59.3.3 used

```
u8_t mem::used
```

1: this area is used; 0: this area is unused

The documentation for this struct was generated from the following file: src/core/mem.c

3.60 memp_desc Struct Reference

```
#include <memp_priv.h>
```

3.60.1 Data Fields

- struct stats_mem * stats
- u16_t size
- u16_t num
- u8_t * base
- struct memp ** tab

3.60.2 Detailed Description

Memory pool descriptor

3.60.3 Field Documentation

3.60.3.1 base

```
u8_t* memp_desc::base
```

Base address

3.60.3.2 num

```
u16_t memp_desc::num
```

Number of elements

3.60.3.3 size

```
u16_t memp_desc::size
```

Element size

3.60.3.4 stats

```
struct stats_mem* memp_desc::stats
```

Statistics

3.60.3.5 tab

```
struct memp** memp_desc::tab
```

First free element of each pool. Elements form a linked list.

The documentation for this struct was generated from the following file: src/include/lwip/priv/memp_priv.h

3.61 mld_group Struct Reference

```
#include <mld6.h>
```

3.61.1 Data Fields

- struct mld_group * next
- ip6_addr_t group_address
- u8_t last_reporter_flag
- u8_t group_state
- u16_t timer
- u8_t use

3.61.2 Detailed Description

MLD group

3.61.3 Field Documentation

3.61.3.1 group_address

```
ip6_addr_t mld_group::group_address
```

multicast address

3.61.3.2 group_state

`u8_t mld_group::group_state`

current state of the group

3.61.3.3 last_reporter_flag

`u8_t mld_group::last_reporter_flag`

signifies we were the last person to report

3.61.3.4 next

`struct mld_group* mld_group::next`

next link

3.61.3.5 timer

`u16_t mld_group::timer`

timer for reporting

3.61.3.6 use

`u8_t mld_group::use`

counter of simultaneous uses

The documentation for this struct was generated from the following file: src/include/lwip/mld6.h

3.62 mld_header Struct Reference

`#include <mld6.h>`

3.62.1 Detailed Description

Multicast listener report/query/done message header.

The documentation for this struct was generated from the following file: src/include/lwip/prot/mld6.h

3.63 mqtt_client_s Struct Reference

`#include <mqtt_priv.h>`

3.63.1 Data Fields

- u16_t cyclic_tick
- u16_t pkt_id_seq
- u16_t inpub_pkt_id
- u8_t conn_state
- void * connect_arg
- struct mqtt_request_t * pend_req_queue
- mqtt_incoming_data_cb_t data_cb
- u32_t msg_idx
- struct mqtt_ringbuf_t output

3.63.2 Detailed Description

MQTT client

3.63.3 Field Documentation

3.63.3.1 conn_state

```
u8_t mqtt_client_s::conn_state
```

Connection state

3.63.3.2 connect_arg

```
void* mqtt_client_s::connect_arg
```

Connection callback

3.63.3.3 cyclic_tick

```
u16_t mqtt_client_s::cyclic_tick
```

Timers and timeouts

3.63.3.4 data_cb

```
mqtt_incoming_data_cb_t mqtt_client_s::data_cb
```

Incoming data callback

3.63.3.5 inpub_pkt_id

```
u16_t mqtt_client_s::inpub_pkt_id
```

Packet identifier of pending incoming publish

3.63.3.6 msg_idx

```
u32_t mqtt_client_s::msg_idx
```

Input

3.63.3.7 output

```
struct mqtt_ringbuf_t mqtt_client_s::output
```

Output ring-buffer

3.63.3.8 pend_req_queue

```
struct mqtt_request_t* mqtt_client_s::pend_req_queue
```

Pending requests to server

3.63.3.9 pkt_id_seq

```
u16_t mqtt_client_s::pkt_id_seq
```

Packet identifier generator

The documentation for this struct was generated from the following file: src/include/lwip/apps/mqtt_priv.h

3.64 mqtt_connect_client_info_t Struct Reference

```
#include <mqtt.h>
```

3.64.1 Data Fields

- const char * client_id
- const char * client_user
- const char * client_pass
- u16_t keep_alive
- const char * will_topic
- const char * will_msg
- u8_t will_msg_len
- u8_t will_qos
- u8_t will_retain
- struct altcp_tls_config * tls_config

3.64.2 Detailed Description

Client information and connection parameters

3.64.3 Field Documentation

3.64.3.1 client_id

`const char* mqtt_connect_client_info_t::client_id`

Client identifier, must be set by caller

3.64.3.2 client_pass

`const char* mqtt_connect_client_info_t::client_pass`

Password, set to NULL if not used

3.64.3.3 client_user

`const char* mqtt_connect_client_info_t::client_user`

User name, set to NULL if not used

3.64.3.4 keep_alive

`u16_t mqtt_connect_client_info_t::keep_alive`

keep alive time in seconds, 0 to disable keep alive functionality

3.64.3.5 tls_config

`struct altcp_tls_config* mqtt_connect_client_info_t::tls_config`

TLS configuration for secure connections

3.64.3.6 will_msg

`const char* mqtt_connect_client_info_t::will_msg`

will_msg, see will_topic

3.64.3.7 will_msg_len

`u8_t mqtt_connect_client_info_t::will_msg_len`

will_msg length, 0 to compute length from will_msg string

3.64.3.8 will_qos

`u8_t mqtt_connect_client_info_t::will_qos`

will_qos, see will_topic

3.64.3.9 will_retain

`u8_t mqtt_connect_client_info_t::will_retain`

will_retain, see will_topic

3.64.3.10 will_topic

```
const char* mqtt_connect_client_info_t::will_topic
```

will topic, set to NULL if will is not to be used, will_msg, will_qos and will retain are then ignored

The documentation for this struct was generated from the following file: src/include/lwip/apps/mqtt.h

3.65 mqtt_request_t Struct Reference

```
#include <mqtt_priv.h>
```

3.65.1 Data Fields

- struct mqtt_request_t * next
- mqtt_request_cb_t cb
- u16_t pkt_id
- u16_t timeout_diff

3.65.2 Detailed Description

Pending request item, binds application callback to pending server requests

3.65.3 Field Documentation

3.65.3.1 cb

```
mqtt_request_cb_t mqtt_request_t::cb
```

Callback to upper layer

3.65.3.2 next

```
struct mqtt_request_t* mqtt_request_t::next
```

Next item in list, NULL means this is the last in chain, next pointing at itself means request is unallocated

3.65.3.3 pkt_id

```
u16_t mqtt_request_t::pkt_id
```

MQTT packet identifier

3.65.3.4 timeout_diff

```
u16_t mqtt_request_t::timeout_diff
```

Expire time relative to element before this

The documentation for this struct was generated from the following file: src/include/lwip/apps/mqtt_priv.h

3.66 mqtt_ringbuf_t Struct Reference

```
#include <mqtt_priv.h>
```

3.66.1 Detailed Description

Ring buffer

The documentation for this struct was generated from the following file: src/include/lwip/apps/mqtt_priv.h

3.67 na_header Struct Reference

```
#include <nd6.h>
```

3.67.1 Detailed Description

Neighbor advertisement message header.

The documentation for this struct was generated from the following file: src/include/lwip/prot/nd6.h

3.68 nd6_neighbor_cache_entry Struct Reference

```
#include <nd6_priv.h>
```

3.68.1 Data Fields

- struct nd6_q_entry * q

3.68.2 Detailed Description

Struct for tables.

3.68.3 Field Documentation

3.68.3.1 q

```
struct nd6_q_entry* nd6_neighbor_cache_entry::q
```

Pointer to queue of pending outgoing packets on this entry.

The documentation for this struct was generated from the following file: src/include/lwip/priv/nd6_priv.h

3.69 nd6_q_entry Struct Reference

```
#include <nd6_priv.h>
```

3.69.1 Detailed Description

struct for queueing outgoing packets for unknown address defined here to be accessed by memp.h

The documentation for this struct was generated from the following file: src/include/lwip/priv/nd6_priv.h

3.70 netbios_answer Struct Reference

3.70.1 Data Fields

- u8_t name_size
- u8_t query_name [(16 *2)+1]
- u8_t number_of_names
- u8_t answer_name [16]
- u16_t answer_name_flags
- u8_t unit_id [6]
- u8_t jumpers
- u8_t test_result
- u16_t version_number
- u16_t period_of_statistics
- u16_t number_of_crcs
- u16_t number_of_alignment_errors
- u16_t number_of_collisions
- u16_t number_of_send_aborts
- u32_t number_of_good_sends
- u32_t number_of_good_receives
- u16_t number_of_retransmits
- u16_t number_of_no_resource_condition
- u16_t number_of_free_command_blocks
- u16_t total_number_of_command_blocks
- u16_t max_total_number_of_command_blocks
- u16_t number_of_pending_sessions
- u16_t max_number_of_pending_sessions
- u16_t max_total_sessions_possible
- u16_t session_data_packet_size

3.70.2 Detailed Description

The NBNS Structure Responds to a Name Query

3.70.3 Field Documentation

3.70.3.1 answer_name

```
u8_t netbios_answer::answer_name[16]
```

node name

3.70.3.2 answer_name_flags

`u16_t netbios_answer::answer_name_flags`

node flags

3.70.3.3 jumpers

`u8_t netbios_answer::jumpers`

Jumpers

3.70.3.4 max_number_of_pending_sessions

`u16_t netbios_answer::max_number_of_pending_sessions`

Statistics

3.70.3.5 max_total_number_of_command_blocks

`u16_t netbios_answer::max_total_number_of_command_blocks`

Statistics

3.70.3.6 max_total_sessions_possible

`u16_t netbios_answer::max_total_sessions_possible`

Statistics

3.70.3.7 name_size

`u8_t netbios_answer::name_size`

the length of the next string

3.70.3.8 number_of_alignment_errors

`u16_t netbios_answer::number_of_alignment_errors`

Statistics

3.70.3.9 number_of_collisions

`u16_t netbios_answer::number_of_collisions`

Statistics

3.70.3.10 number_of_crcs

`u16_t netbios_answer::number_of_crcs`

Statistics

3.70.3.11 number_of_free_command_blocks

`u16_t netbios_answer::number_of_free_command_blocks`

Statistics

3.70.3.12 number_of_good_receives

`u32_t netbios_answer::number_of_good_receives`

Statistics

3.70.3.13 number_of_good_sends

`u32_t netbios_answer::number_of_good_sends`

Statistics

3.70.3.14 number_of_names

`u8_t netbios_answer::number_of_names`

number of names

3.70.3.15 number_of_no_resource_condition

`u16_t netbios_answer::number_of_no_resource_condition`

Statistics

3.70.3.16 number_of_pending_sessions

`u16_t netbios_answer::number_of_pending_sessions`

Statistics

3.70.3.17 number_of_retransmits

`u16_t netbios_answer::number_of_retransmits`

Statistics

3.70.3.18 number_of_send_aborts

`u16_t netbios_answer::number_of_send_aborts`

Statistics

3.70.3.19 period_of_statistics

`u16_t netbios_answer::period_of_statistics`

Period of statistics

3.70.3.20 query_name

`u8_t netbios_answer::query_name[(16 *2)+1]`

WARNING!!! this item may be of a different length (we use this struct for transmission)

3.70.3.21 session_data_packet_size

`u16_t netbios_answer::session_data_packet_size`

Statistics

3.70.3.22 test_result

`u8_t netbios_answer::test_result`

Test result

3.70.3.23 total_number_of_command_blocks

`u16_t netbios_answer::total_number_of_command_blocks`

Statistics

3.70.3.24 unit_id

`u8_t netbios_answer::unit_id[6]`

Unit ID

3.70.3.25 version_number

`u16_t netbios_answer::version_number`

Version number

The documentation for this struct was generated from the following file: src/apps/netbiosns/netbiosns.c

3.71 netbios_hdr Struct Reference

3.71.1 Detailed Description

NetBIOS message header

The documentation for this struct was generated from the following file: src/apps/netbiosns/netbiosns.c

3.72 netbios_name_hdr Struct Reference

3.72.1 Detailed Description

NetBIOS message name part

The documentation for this struct was generated from the following file: src/apps/netbiosns/netbiosns.c

3.73 netbios_question_hdr Struct Reference

3.73.1 Detailed Description

NetBIOS message question part

The documentation for this struct was generated from the following file: src/apps/netbiosns/netbiosns.c

3.74 netbios_resp Struct Reference

3.74.1 Detailed Description

NetBIOS message

The documentation for this struct was generated from the following file: src/apps/netbiosns/netbiosns.c

3.75 netbuf Struct Reference

```
#include <netbuf.h>
```

3.75.1 Detailed Description

"Network buffer" - contains data and addressing info

The documentation for this struct was generated from the following file: src/include/lwip/netbuf.h

3.76 netconn Struct Reference

```
#include <api.h>
```

3.76.1 Data Fields

- enum netconn_type type
- enum netconn_state state
- union {
- } pcb
- err_t pending_err
- sys_sem_t op_completed
- sys_mbox_t recvmbox
- sys_mbox_t acceptmbox
- s32_t send_timeout
- int recv_bufsize
- int recv_avail
- s16_t linger

- u8_t flags
- struct api_msg * current_msg
- netconn_callback callback

3.76.2 Detailed Description

A netconn descriptor

3.76.3 Field Documentation

3.76.3.1 acceptmbox

```
sys_mbox_t netconn::acceptmbox
```

mbox where new connections are stored until processed by the application thread

3.76.3.2 callback

```
netconn_callback netconn::callback
```

A callback function that is informed about events for this netconn

3.76.3.3 current_msg

```
struct api_msg* netconn::current_msg
```

TCP: when data passed to netconn_write doesn't fit into the send buffer, this temporarily stores the message. Also used during connect and close.

3.76.3.4 flags

```
u8_t netconn::flags
```

flags holding more netconn-internal state, see NETCONN_FLAG_* defines

3.76.3.5 linger

```
s16_t netconn::linger
```

values <0 mean linger is disabled, values > 0 are seconds to linger

3.76.3.6 op_completed

```
sys_sem_t netconn::op_completed
```

sem that is used to synchronously execute functions in the core context

3.76.3.7 [union]

```
union { ... } netconn::pcb
```

the lwIP internal protocol control block

3.76.3.8 pending_err

```
err_t netconn::pending_err
```

the last asynchronous unreported error this netconn had

3.76.3.9 recv_avail

```
int netconn::recv_avail
```

number of bytes currently in recvmbox to be received, tested against recv_bufsize to limit bytes on recvmbox for UDP and RAW, used for FIONREAD

3.76.3.10 recv_bufsize

```
int netconn::recv_bufsize
```

maximum amount of bytes queued in recvmbox not used for TCP: adjust TCP_WND instead!

3.76.3.11 recvmbox

```
sys_mbox_t netconn::recvmbox
```

mbox where received packets are stored until they are fetched by the netconn application thread (can grow quite big)

3.76.3.12 send_timeout

```
s32_t netconn::send_timeout
```

timeout to wait for sending data (which means enqueueing data for sending in internal buffers) in milliseconds

3.76.3.13 state

```
enum netconn_state netconn::state
```

current state of the netconn

3.76.3.14 type

```
enum netconn_type netconn::type
```

type of the netconn (TCP, UDP or RAW)

The documentation for this struct was generated from the following file: src/include/lwip/api.h

3.77 netif Struct Reference

```
#include <netif.h>
```

3.77.1 Data Fields

- struct netif * next
- ip_addr_t ip_addr
- ip_addr_t ip6_addr [3]
- u8_t ip6_addr_state [3]
- u32_t ip6_addr_valid_life [3]
- netif_input_fn input
- netif_output_fn output
- netif_linkoutput_fn linkoutput
- netif_output_ip6_fn output_ip6
- netif_status_callback_fn status_callback
- netif_status_callback_fn link_callback
- netif_status_callback_fn remove_callback
- void * state
- u16_t mtu
- u16_t mtu6
- u8_t hwaddr [6U]
- u8_t hwaddr_len
- u8_t flags
- char name [2]
- u8_t num
- u8_t ip6_autoconfig_enabled
- u8_t rs_count
- u8_t link_type
- u32_t link_speed
- u32_t ts
- struct stats_mib2_netif_ctrs mib2_counters
- netif_igmp_mac_filter_fn igmp_mac_filter
- netif_mld_mac_filter_fn mld_mac_filter

3.77.2 Detailed Description

Generic data structure used for all lwIP network interfaces. The following fields should be filled in by the initialization function for the device driver: hwaddr_len, hwaddr[], mtu, flags

3.77.3 Field Documentation

3.77.3.1 flags

`u8_t netif::flags`

flags (**See also** Flags)

3.77.3.2 hwaddr

`u8_t netif::hwaddr[6U]`

link level hardware address of this interface

3.77.3.3 hwaddr_len

`u8_t netif::hwaddr_len`

number of bytes used in hwaddr

3.77.3.4 igmp_mac_filter

`netif_igmp_mac_filter_fn netif::igmp_mac_filter`

This function could be called to add or delete an entry in the multicast filter table of the ethernet MAC.

3.77.3.5 input

`netif_input_fn netif::input`

This function is called by the network device driver to pass a packet up the TCP/IP stack.

3.77.3.6 ip6_addr

`ip_addr_t netif::ip6_addr[3]`

Array of IPv6 addresses for this netif.

3.77.3.7 ip6_addr_state

`u8_t netif::ip6_addr_state[3]`

The state of each IPv6 address (Tentative, Preferred, etc). **See also** ip6_addr.h

3.77.3.8 ip6_addr_valid_life

`u32_t netif::ip6_addr_valid_life[3]`

Remaining valid and preferred lifetime of each IPv6 address, in seconds. For valid lifetimes, the special value of IP6_ADDR_LIFE_STAT
(0) indicates the address is static and has no lifetimes.

3.77.3.9 ip6_autoconfig_enabled

`u8_t netif::ip6_autoconfig_enabled`

is this netif enabled for IPv6 autoconfiguration

3.77.3.10 ip_addr

`ip_addr_t netif::ip_addr`

IP address configuration in network byte order

3.77.3.11 link_callback

`netif_status_callback_fn netif::link_callback`

This function is called when the netif link is set to up or down

3.77.3.12 link_speed

`u32_t netif::link_speed`

(estimate) link speed

3.77.3.13 link_type

`u8_t netif::link_type`

link type (from "snmp_ifType" enum from snmp_mib2.h)

3.77.3.14 linkoutput

`netif_linkoutput_fn netif::linkoutput`

This function is called by ethernet_output() when it wants to send a packet on the interface. This function outputs the pbuf as-is on the link medium.

3.77.3.15 mib2_counters

`struct stats_mib2_netif_ctrs netif::mib2_counters`

counters

3.77.3.16 mld_mac_filter

`netif_mld_mac_filter_fn netif::mld_mac_filter`

This function could be called to add or delete an entry in the IPv6 multicast filter table of the ethernet MAC.

3.77.3.17 mtu

`u16_t netif::mtu`

maximum transfer unit (in bytes)

3.77.3.18 mtu6

`u16_t netif::mtu6`

maximum transfer unit (in bytes), updated by RA

3.77.3.19 name

`char netif::name[2]`

descriptive abbreviation

3.77.3.20 next

`struct netif* netif::next`

pointer to next in linked list

3.77.3.21 num

`u8_t netif::num`

number of this interface. Used for Interface Identification API and NETIF related, as well as for IPv6 zones

3.77.3.22 output

`netif_output_fn netif::output`

This function is called by the IP module when it wants to send a packet on the interface. This function typically first resolves the hardware address, then sends the packet. For ethernet physical layer, this is usually etharp_output()

3.77.3.23 output_ip6

`netif_output_ip6_fn netif::output_ip6`

This function is called by the IPv6 module when it wants to send a packet on the interface. This function typically first resolves the hardware address, then sends the packet. For ethernet physical layer, this is usually ethip6_output()

3.77.3.24 remove_callback

`netif_status_callback_fn netif::remove_callback`

This function is called when the netif has been removed

3.77.3.25 rs_count

`u8_t netif::rs_count`

Number of Router Solicitation messages that remain to be sent.

3.77.3.26 state

`void* netif::state`

This field can be set by the device driver and could point to state information for the device.

3.77.3.27 status_callback

`netif_status_callback_fn netif::status_callback`

This function is called when the netif state is set to up or down

3.77.3.28 ts

```
u32_t netif::ts
```

timestamp at last change made (up/down)

The documentation for this struct was generated from the following file: src/include/lwip/netif.h

3.78 netif_ext_callback_args_t Union Reference

```
#include <netif.h>
```

3.78.1 Data Structures

- struct ipv4_changed_s
- struct ipv6_addr_state_changed_s
- struct ipv6_set_s
- struct link_changed_s
- struct status_changed_s

3.78.2 Detailed Description

Argument supplied to netif_ext_callback_fn.

The documentation for this union was generated from the following file: src/include/lwip/netif.h

3.79 netvector Struct Reference

```
#include <api.h>
```

3.79.1 Data Fields

- const void * ptr
- size_t len

3.79.2 Detailed Description

This vector type is passed to netconn_write_vectors_partly to send multiple buffers at once. ATTENTION: This type has to directly map struct iovec since one is casted into the other!

3.79.3 Field Documentation

3.79.3.1 len

```
size_t netvector::len
```

size of the application data to send

3.79.3.2 ptr

```
const void* netvector::ptr
```

pointer to the application buffer that contains the data to send

The documentation for this struct was generated from the following file: src/include/lwip/api.h

3.80 ns_header Struct Reference

```
#include <nd6.h>
```

3.80.1 Detailed Description

Neighbor solicitation message header.

The documentation for this struct was generated from the following file: src/include/lwip/prot/nd6.h

3.81 pbuf Struct Reference

```
#include <pbuf.h>
```

3.81.1 Data Fields

- struct pbuf * next
- void * payload
- u16_t tot_len
- u16_t len
- u8_t type_internal
- u8_t flags
- u8_t ref
- u8_t if_idx

3.81.2 Detailed Description

Main packet buffer struct

3.81.3 Field Documentation

3.81.3.1 flags

```
u8_t pbuf::flags
```

misc flags

3.81.3.2 if_idx

```
u8_t pbuf::if_idx
```

For incoming packets, this contains the input netif's index

3.81.3.3 len

```
u16_t pbuf::len
```

length of this buffer

3.81.3.4 next

```
struct pbuf* pbuf::next
```

next pbuf in singly linked pbuf chain

3.81.3.5 payload

```
void* pbuf::payload
```

pointer to the actual data in the buffer

3.81.3.6 ref

```
u8_t pbuf::ref
```

the reference count always equals the number of pointers that refer to this pbuf. This can be pointers from an application, the stack itself, or pbuf->next pointers from a chain.

3.81.3.7 tot_len

```
u16_t pbuf::tot_len
```

total length of this buffer and all next buffers in chain belonging to the same packet.

For non-queue packet chains this is the invariant: p->tot_len == p->len + (p->next? p->next->tot_len: 0)

3.81.3.8 type_internal

```
u8_t pbuf::type_internal
```

a bit field indicating pbuf type and allocation sources (see PBUF_TYPE_FLAG_*, PBUF_ALLOC_FLAG_* and PBUF_TYPE_ALLOC

The documentation for this struct was generated from the following file: src/include/lwip/pbuf.h

3.82 pbuf_custom Struct Reference

```
#include <pbuf.h>
```

3.82.1 Data Fields

- struct pbuf pbuf
- pbuf_free_custom_fn custom_free_function

3.82.2 Detailed Description

A custom pbuf: like a pbuf, but following a function pointer to free it.

3.82.3 Field Documentation

3.82.3.1 custom_free_function

`pbuf_free_custom_fn pbuf_custom::custom_free_function`

This function is called when pbuf_free deallocates this pbuf(_custom)

3.82.3.2 pbuf

`struct pbuf pbuf_custom::pbuf`

The actual pbuf

The documentation for this struct was generated from the following file: src/include/lwip/pbuf.h

3.83 pbuf_custom_ref Struct Reference

`#include <ip6_frag.h>`

3.83.1 Data Fields

- struct pbuf_custom pc
- struct pbuf * original

3.83.2 Detailed Description

A custom pbuf that holds a reference to another pbuf, which is freed when this custom pbuf is freed. This is used to create a custom PBUF_REF that points into the original pbuf.

3.83.3 Field Documentation

3.83.3.1 original

`struct pbuf * pbuf_custom_ref::original`

pointer to the original pbuf that is referenced

3.83.3.2 pc

`struct pbuf_custom pbuf_custom_ref::pc`

'base class'

The documentation for this struct was generated from the following files: src/include/lwip/ip4_frag.hsrc/include/lwip/ip6_frag.h

3.84 pbuf_rom Struct Reference

`#include <pbuf.h>`

3.84.1 Data Fields

- struct pbuf * next
- const void * payload

3.84.2 Detailed Description

Helper struct for const-correctness only. The only meaning of this one is to provide a const payload pointer for PBUF_ROM type.

3.84.3 Field Documentation

3.84.3.1 next

```
struct pbuf* pbuf_rom::next
```

next pbuf in singly linked pbuf chain

3.84.3.2 payload

```
const void* pbuf_rom::payload
```

pointer to the actual data in the buffer

The documentation for this struct was generated from the following file: src/include/lwip/pbuf.h

3.85 raw_pcb Struct Reference

```
#include <raw.h>
```

3.85.1 Data Fields

- u8_t mcast_ifindex
- u8_t mcast_ttl
- raw_recv_fn recv

3.85.2 Detailed Description

the RAW protocol control block

3.85.3 Field Documentation

3.85.3.1 mcast_ifindex

```
u8_t raw_pcb::mcast_ifindex
```

outgoing network interface for multicast packets, by interface index (if nonzero)

3.85.3.2 mcast_ttl

`u8_t raw_pcb::mcast_ttl`

TTL for outgoing multicast packets

3.85.3.3 recv

`raw_recv_fn raw_pcb::recv`

receive callback function

The documentation for this struct was generated from the following file: src/include/lwip/raw.h

3.86 redirect_header Struct Reference

`#include <nd6.h>`

3.86.1 Detailed Description

Redirect message header.

The documentation for this struct was generated from the following file: src/include/lwip/prot/nd6.h

3.87 rs_header Struct Reference

`#include <nd6.h>`

3.87.1 Detailed Description

Router solicitation message header.

The documentation for this struct was generated from the following file: src/include/lwip/prot/nd6.h

3.88 smtp_send_request Struct Reference

`#include <smtp.h>`

3.88.1 Data Fields

- u8_t static_data

3.88.2 Detailed Description

This structure is used as argument for smtp_send_mail_int(), which in turn can be used with tcpip_callback() to send mail from interrupt context, e.g. like this: struct smtp_send_request *req; (to be filled) tcpip_try_callback(smtp_send_mail_int, (void*)req);

For member description, see parameter description of smtp_send_mail(). When using with tcpip_callback, this structure has to stay allocated (e.g. using mem_malloc/mem_free) until its 'callback_fn' is called.

3.88.3 Field Documentation

3.88.3.1 static_data

```
u8_t smtp_send_request::static_data
```

If this is != 0, data is *not* copied into an extra buffer but used from the pointers supplied in this struct. This means less memory usage, but data must stay untouched until the callback function is called.

The documentation for this struct was generated from the following file: src/include/lwip/apps/smtp.h

3.89 smtp_session Struct Reference

3.89.1 Data Fields

- enum smtp_session_state state
- u16_t timer
- char tx_buf [255+1]
- const char * from
- u16_t from_len
- const char * to
- u16_t to_len
- const char * subject
- u16_t subject_len
- const char * body
- u16_t body_len
- u16_t body_sent
- smtp_result_fn callback_fn
- void * callback_arg
- char * username
- char * pass
- char auth_plain [32+32+3]
- size_t auth_plain_len

3.89.2 Detailed Description

struct keeping the body and state of an smtp session

3.89.3 Field Documentation

3.89.3.1 auth_plain

```
char smtp_session::auth_plain[32+ 32+3]
```

Username and password combined as necessary for PLAIN authentication

3.89.3.2 auth_plain_len

```
size_t smtp_session::auth_plain_len
```

Length of smtp_auth_plain string (cannot use strlen since it includes \0)

3.89.3.3 body

```
const char* smtp_session::body
```

this is the body of the mail to be sent

3.89.3.4 body_len

```
u16_t smtp_session::body_len
```

this is the length of the body to be sent

3.89.3.5 body_sent

```
u16_t smtp_session::body_sent
```

amount of data from body already sent

3.89.3.6 callback_arg

```
void* smtp_session::callback_arg
```

argument for callback function

3.89.3.7 callback_fn

```
smtp_result_fn smtp_session::callback_fn
```

callback function to call when closed

3.89.3.8 from

```
const char* smtp_session::from
```

source email address

3.89.3.9 from_len

```
u16_t smtp_session::from_len
```

size of the sourceemail address

3.89.3.10 pass

```
char* smtp_session::pass
```

Password to use for this request

3.89.3.11 state

```
enum smtp_session_state smtp_session::state
```

keeping the state of the smtp session

3.89.3.12 subject

```
const char* smtp_session::subject
```

subject of the email

3.89.3.13 subject_len

```
u16_t smtp_session::subject_len
```

length of the subject string

3.89.3.14 timer

```
u16_t smtp_session::timer
```

timeout handling, if this reaches 0, the connection is closed

3.89.3.15 to

```
const char* smtp_session::to
```

target email address

3.89.3.16 to_len

```
u16_t smtp_session::to_len
```

size of the target email address

3.89.3.17 tx_buf

```
char smtp_session::tx_buf[255+1]
```

helper buffer for transmit, not used for sending body

3.89.3.18 username

```
char* smtp_session::username
```

Username to use for this request

The documentation for this struct was generated from the following file: src/apps/smtp/smtp.c

3.90 snmp_leaf_node Struct Reference

```
#include <snmp_core.h>
```

3.90.1 Data Fields

- struct snmp_node node

3.90.2 Detailed Description

SNMP leaf node

3.90.3 Field Documentation

3.90.3.1 node

```
struct snmp_node snmp_leaf_node::node
```

inherited "base class" members

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_core.h

3.91 snmp_mib Struct Reference

```
#include <snmp_core.h>
```

3.91.1 Detailed Description

represents a single mib with its base oid and root node

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_core.h

3.92 snmp_next_oid_state Struct Reference

```
#include <snmp_core.h>
```

3.92.1 Detailed Description

state for next_oid_init / next_oid_check functions

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_core.h

3.93 snmp_node Struct Reference

```
#include <snmp_core.h>
```

3.93.1 Data Fields

- u8_t node_type
- u32_t oid

3.93.2 Detailed Description

node "base class" layout, the mandatory fields for a node

3.93.3 Field Documentation

3.93.3.1 node_type

`u8_t snmp_node::node_type`

one out of SNMP_NODE_TREE or any leaf node type (like SNMP_NODE_SCALAR)

3.93.3.2 oid

`u32_t snmp_node::oid`

the number assigned to this node which used as part of the full OID

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_core.h

3.94 snmp_node_instance Struct Reference

`#include <snmp_core.h>`

3.94.1 Data Fields

- const struct snmp_node * node
- struct snmp_obj_id instance_oid
- u8_t asn1_type
- snmp_access_t access
- node_instance_get_value_method get_value
- node_instance_set_test_method set_test
- node_instance_set_value_method set_value
- node_instance_release_method release_instance
- union snmp_variant_value reference
- u32_t reference_len

3.94.2 Detailed Description

SNMP node instance

3.94.3 Field Documentation

3.94.3.1 access

`snmp_access_t snmp_node_instance::access`

one out of instance access types defined above (SNMP_NODE_INSTANCE_READ_ONLY,...)

3.94.3.2 asn1_type

`u8_t snmp_node_instance::asn1_type`

ASN type for this object (see snmp_asn1.h for definitions)

3.94.3.3 get_value

```
node_instance_get_value_method snmp_node_instance::get_value
```

returns object value for the given object identifier. Return values <0 to indicate an error

3.94.3.4 instance_oid

```
struct snmp_obj_id snmp_node_instance::instance_oid
```

prefilled with the instance id requested; for get_instance() this is the exact oid requested; for get_next_instance() this is the relative starting point, stack expects relative oid of next node here

3.94.3.5 node

```
const struct snmp_node* snmp_node_instance::node
```

prefilled with the node, get_instance() is called on; may be changed by user to any value to pass an arbitrary node between calls to get_instance() and get_value/test_value/set_value

3.94.3.6 reference

```
union snmp_variant_value snmp_node_instance::reference
```

reference to pass arbitrary value between calls to get_instance() and get_value/test_value/set_value

3.94.3.7 reference_len

```
u32_t snmp_node_instance::reference_len
```

see reference (if reference is a pointer, the length of underlying data may be stored here or anything else)

3.94.3.8 release_instance

```
node_instance_release_method snmp_node_instance::release_instance
```

called in any case when the instance is not required anymore by stack (useful for freeing memory allocated in get_instance/get_next_insta
methods)

3.94.3.9 set_test

```
node_instance_set_test_method snmp_node_instance::set_test
```

tests length and/or range BEFORE setting

3.94.3.10 set_value

```
node_instance_set_value_method snmp_node_instance::set_value
```

sets object value, only called when set_test() was successful

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_core.h

3.95 snmp_obj_id Struct Reference

```
#include <snmp_core.h>
```

3.95.1 Detailed Description

internal object identifier representation

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_core.h

3.96 snmp_oid_range Struct Reference

```
#include <snmp_core.h>
```

3.96.1 Detailed Description

OID range structure

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_core.h

3.97 snmp_scalar_array_node Struct Reference

```
#include <snmp_scalar.h>
```

3.97.1 Data Fields

- struct snmp_leaf_node node

3.97.2 Detailed Description

basic scalar array node

3.97.3 Field Documentation

3.97.3.1 node

```
struct snmp_leaf_node snmp_scalar_array_node::node
```

inherited "base class" members

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_scalar.h

3.98 snmp_scalar_array_node_def Struct Reference

```
#include <snmp_scalar.h>
```

3.98.1 Detailed Description

scalar array node - a tree node which contains scalars only as children

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_scalar.h

3.99 snmp_scalar_node Struct Reference

```
#include <snmp_scalar.h>
```

3.99.1 Data Fields

- struct snmp_leaf_node node

3.99.2 Detailed Description

basic scalar node

3.99.3 Field Documentation

3.99.3.1 node

```
struct snmp_leaf_node snmp_scalar_node::node
```

inherited "base class" members

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_scalar.h

3.100 snmp_table_col_def Struct Reference

```
#include <snmp_table.h>
```

3.100.1 Detailed Description

default (customizable) read/write table

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_table.h

3.101 snmp_table_node Struct Reference

```
#include <snmp_table.h>
```

3.101.1 Data Fields

- struct snmp_leaf_node node
- node_instance_get_value_method get_value
- node_instance_set_test_method set_test
- node_instance_set_value_method set_value

3.101.2 Detailed Description

table node

3.101.3 Field Documentation

3.101.3.1 get_value

`node_instance_get_value_method snmp_table_node::get_value`

returns object value for the given object identifier

3.101.3.2 node

`struct snmp_leaf_node snmp_table_node::node`

inherited "base class" members

3.101.3.3 set_test

`node_instance_set_test_method snmp_table_node::set_test`

tests length and/or range BEFORE setting

3.101.3.4 set_value

`node_instance_set_value_method snmp_table_node::set_value`

sets object value, only called when set_test() was successful

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_table.h

3.102 snmp_table_simple_node Struct Reference

```
#include <snmp_table.h>
```

3.102.1 Detailed Description

simple read-only table node

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_table.h

3.103 snmp_threadsync_instance Struct Reference

```
#include <snmp_threadsync.h>
```

3.103.1 Detailed Description

Thread sync instance. Needed EXACTLY once for every thread to be synced into.

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_threadsync.h

3.104 snmp_threadsync_node Struct Reference

```
#include <snmp_threadsync.h>
```

3.104.1 Detailed Description

SNMP thread sync proxy leaf node

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_threadsync.h

3.105 snmp_tree_node Struct Reference

```
#include <snmp_core.h>
```

3.105.1 Data Fields

- struct snmp_node node

3.105.2 Detailed Description

SNMP tree node

3.105.3 Field Documentation

3.105.3.1 node

```
struct snmp_node snmp_tree_node::node
```

inherited "base class" members

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_core.h

3.106 snmp_varbind Struct Reference

```
#include <snmp.h>
```

3.106.1 Data Fields

- struct snmp_varbind * next
- struct snmp_obj_id oid
- u8_t type
- u16_t value_len
- void * value

3.106.2 Detailed Description

SNMP variable binding descriptor (publicly needed for traps)

3.106.3 Field Documentation

3.106.3.1 next

```
struct snmp_varbind* snmp_varbind::next
```

pointer to next varbind, NULL for last in list

3.106.3.2 oid

`struct snmp_obj_id snmp_varbind::oid`

object identifier

3.106.3.3 type

`u8_t snmp_varbind::type`

value ASN1 type

3.106.3.4 value

`void* snmp_varbind::value`

object value

3.106.3.5 value_len

`u16_t snmp_varbind::value_len`

object value length

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp.h

3.107 snmp_varbind_len Struct Reference

`#include <snmp_msg.h>`

3.107.1 Detailed Description

A helper struct keeping length information about varbinds

The documentation for this struct was generated from the following file: src/apps/snmp/snmp_msg.h

3.108 snmp_variant_value Union Reference

`#include <snmp_core.h>`

3.108.1 Detailed Description

SNMP variant value, used as reference in struct snmp_node_instance and table implementation

The documentation for this union was generated from the following file: src/include/lwip/apps/snmp_core.h

3.109 sntp_msg Struct Reference

3.109.1 Detailed Description

SNTP packet format (without optional fields) Timestamps are coded as 64 bits:

- signed 32 bits seconds since Feb 07, 2036, 06:28:16 UTC (epoch 1)
- unsigned 32 bits seconds fraction (2^32 = 1 second)

The documentation for this struct was generated from the following file: src/apps/sntp/sntp.c

3.110 sntp_server Struct Reference

3.110.1 Data Fields

- u8_t reachability

3.110.2 Detailed Description

Names/Addresses of servers

3.110.3 Field Documentation

3.110.3.1 reachability

```
u8_t sntp_server::reachability
```

Reachability shift register as described in RFC 5905

The documentation for this struct was generated from the following file: src/apps/sntp/sntp.c

3.111 sntp_time Struct Reference

3.111.1 Detailed Description

64-bit NTP timestamp, in network byte order.

The documentation for this struct was generated from the following file: src/apps/sntp/sntp.c

3.112 sntp_timestamps Struct Reference

3.112.1 Detailed Description

Timestamps to be extracted from the NTP header.

The documentation for this struct was generated from the following file: src/apps/sntp/sntp.c

3.113 sockaddr_aligned Union Reference

3.113.1 Detailed Description

A struct sockaddr replacement that has the same alignment as sockaddr_in/ sockaddr_in6 if instantiated.

The documentation for this union was generated from the following file: src/api/sockets.c

3.114 stats_ Struct Reference

```
#include <stats.h>
```

3.114.1 Data Fields

- struct stats_proto link
- struct stats_proto etharp
- struct stats_proto ip_frag
- struct stats_proto ip
- struct stats_proto icmp
- struct stats_igmp igmp
- struct stats_proto udp
- struct stats_proto tcp
- struct stats_mem mem
- struct stats_mem * memp [MEMP_MAX]
- struct stats_sys sys
- struct stats_proto ip6
- struct stats_proto icmp6
- struct stats_proto ip6_frag
- struct stats_igmp mld6
- struct stats_proto nd6
- struct stats_mib2 mib2

3.114.2 Detailed Description

lwIP stats container

3.114.3 Field Documentation

3.114.3.1 etharp

`struct stats_proto stats_::etharp`

ARP

3.114.3.2 icmp

`struct stats_proto stats_::icmp`

ICMP

3.114.3.3 icmp6

`struct stats_proto stats_::icmp6`

ICMP6

3.114.3.4 igmp

`struct stats_igmp stats_::igmp`

IGMP

3.114.3.5 ip

`struct stats_proto stats_::ip`

IP

3.114.3.6 ip6

`struct stats_proto stats_::ip6`

IPv6

3.114.3.7 ip6_frag

`struct stats_proto stats_::ip6_frag`

IPv6 fragmentation

3.114.3.8 ip_frag

`struct stats_proto stats_::ip_frag`

Fragmentation

3.114.3.9 link

`struct stats_proto stats_::link`

Link level

3.114.3.10 mem

`struct stats_mem stats_::mem`

Heap

3.114.3.11 memp

`struct stats_mem* stats_::memp[MEMP_MAX]`

Internal memory pools

3.114.3.12 mib2

`struct stats_mib2 stats_::mib2`

SNMP MIB2

3.114.3.13 mld6

```
struct stats_igmp stats_::mld6
```

Multicast listener discovery

3.114.3.14 nd6

```
struct stats_proto stats_::nd6
```

Neighbor discovery

3.114.3.15 sys

```
struct stats_sys stats_::sys
```

System

3.114.3.16 tcp

```
struct stats_proto stats_::tcp
```

TCP

3.114.3.17 udp

```
struct stats_proto stats_::udp
```

UDP

The documentation for this struct was generated from the following file: src/include/lwip/stats.h

3.115 stats_igmp Struct Reference

```
#include <stats.h>
```

3.115.1 Detailed Description

IGMP stats

The documentation for this struct was generated from the following file: src/include/lwip/stats.h

3.116 stats_mem Struct Reference

```
#include <stats.h>
```

3.116.1 Detailed Description

Memory stats

The documentation for this struct was generated from the following file: src/include/lwip/stats.h

3.117 stats_mib2 Struct Reference

```
#include <stats.h>
```

3.117.1 Detailed Description

SNMP MIB2 stats

The documentation for this struct was generated from the following file: src/include/lwip/stats.h

3.118 stats_mib2_netif_ctrs Struct Reference

```
#include <stats.h>
```

3.118.1 Data Fields

- u32_t ifinoctets
- u32_t ifinucastpkts
- u32_t ifinnucastpkts
- u32_t ifindiscards
- u32_t ifinerrors
- u32_t ifinunknownprotos
- u32_t ifoutoctets
- u32_t ifoutucastpkts
- u32_t ifoutnucastpkts
- u32_t ifoutdiscards
- u32_t ifouterrors

3.118.2 Detailed Description

SNMP MIB2 interface stats

3.118.3 Field Documentation

3.118.3.1 ifindiscards

```
u32_t stats_mib2_netif_ctrs::ifindiscards
```

The number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space

3.118.3.2 ifinerrors

```
u32_t stats_mib2_netif_ctrs::ifinerrors
```

For packet-oriented interfaces, the number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol. For character- oriented or fixed-length interfaces, the number of inbound transmission units that contained errors preventing them from being deliverable to a higher-layer protocol.

3.118.3.3 ifinnucastpkts

`u32_t stats_mib2_netif_ctrs::ifinnucastpkts`

The number of packets, delivered by this sub-layer to a higher (sub-)layer, which were addressed to a multicast or broadcast address at this sub-layer

3.118.3.4 ifinoctets

`u32_t stats_mib2_netif_ctrs::ifinoctets`

The total number of octets received on the interface, including framing characters

3.118.3.5 ifinucastpkts

`u32_t stats_mib2_netif_ctrs::ifinucastpkts`

The number of packets, delivered by this sub-layer to a higher (sub-)layer, which were not addressed to a multicast or broadcast address at this sub-layer

3.118.3.6 ifinunknownprotos

`u32_t stats_mib2_netif_ctrs::ifinunknownprotos`

For packet-oriented interfaces, the number of packets received via the interface which were discarded because of an unknown or unsupported protocol. For character-oriented or fixed-length interfaces that support protocol multiplexing the number of transmission units received via the interface which were discarded because of an unknown or unsupported protocol. For any interface that does not support protocol multiplexing, this counter will always be 0

3.118.3.7 ifoutdiscards

`u32_t stats_mib2_netif_ctrs::ifoutdiscards`

The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space.

3.118.3.8 ifouterrors

`u32_t stats_mib2_netif_ctrs::ifouterrors`

For packet-oriented interfaces, the number of outbound packets that could not be transmitted because of errors. For character-oriented or fixed-length interfaces, the number of outbound transmission units that could not be transmitted because of errors.

3.118.3.9 ifoutnucastpkts

`u32_t stats_mib2_netif_ctrs::ifoutnucastpkts`

The total number of packets that higher-level protocols requested be transmitted, and which were addressed to a multicast or broadcast address at this sub-layer, including those that were discarded or not sent.

3.118.3.10 ifoutoctets

`u32_t stats_mib2_netif_ctrs::ifoutoctets`

The total number of octets transmitted out of the interface, including framing characters.

3.118.3.11 ifoutucastpkts

```
u32_t stats_mib2_netif_ctrs::ifoutucastpkts
```

The total number of packets that higher-level protocols requested be transmitted, and which were not addressed to a multicast or broadcast address at this sub-layer, including those that were discarded or not sent.

The documentation for this struct was generated from the following file: src/include/lwip/stats.h

3.119 stats_proto Struct Reference

```
#include <stats.h>
```

3.119.1 Detailed Description

Protocol related stats

The documentation for this struct was generated from the following file: src/include/lwip/stats.h

3.120 stats_sys Struct Reference

```
#include <stats.h>
```

3.120.1 Detailed Description

System stats

The documentation for this struct was generated from the following file: src/include/lwip/stats.h

3.121 stats_syselem Struct Reference

```
#include <stats.h>
```

3.121.1 Detailed Description

System element stats

The documentation for this struct was generated from the following file: src/include/lwip/stats.h

3.122 netif_ext_callback_args_t::status_changed_s Struct Reference

```
#include <netif.h>
```

3.122.1 Data Fields

- u8_t state

3.122.2 Detailed Description

Args to LWIP_NSC_STATUS_CHANGED callback

3.122.3 Field Documentation

3.122.3.1 state

`u8_t netif_ext_callback_args_t::status_changed_s::state`

1: up; 0: down

The documentation for this struct was generated from the following file: src/include/lwip/netif.h

3.123 tCGI Struct Reference

`#include <httpd.h>`

3.123.1 Detailed Description

Structure defining the base filename (URL) of a CGI and the associated function which is to be called when that URL is requested.

The documentation for this struct was generated from the following file: src/include/lwip/apps/httpd.h

3.124 tcp_ext_arg_callbacks Struct Reference

`#include <tcp.h>`

3.124.1 Data Fields

- tcp_extarg_callback_pcb_destroyed_fn destroy
- tcp_extarg_callback_passive_open_fn passive_open

3.124.2 Detailed Description

A table of callback functions that is invoked for ext arguments

3.124.3 Field Documentation

3.124.3.1 destroy

`tcp_extarg_callback_pcb_destroyed_fn tcp_ext_arg_callbacks::destroy`

tcp_extarg_callback_pcb_destroyed_fn

3.124.3.2 passive_open

`tcp_extarg_callback_passive_open_fn tcp_ext_arg_callbacks::passive_open`

tcp_extarg_callback_passive_open_fn

The documentation for this struct was generated from the following file: src/include/lwip/tcp.h

3.125 tcp_pcb Struct Reference

`#include <tcp.h>`

3.125.1 Data Fields

- ip_addr_t local_ip
- struct tcp_pcb * next

3.125.2 Detailed Description

the TCP protocol control block

3.125.3 Field Documentation

3.125.3.1 local_ip

`ip_addr_t tcp_pcb::local_ip`

common PCB members

3.125.3.2 next

`struct tcp_pcb* tcp_pcb::next`

protocol specific PCB members

The documentation for this struct was generated from the following file: src/include/lwip/tcp.h

3.126 tcp_pcb_listen Struct Reference

`#include <tcp.h>`

3.126.1 Data Fields

- ip_addr_t local_ip
- struct tcp_pcb_listen * next

3.126.2 Detailed Description

the TCP protocol control block for listening pcbs

3.126.3 Field Documentation

3.126.3.1 local_ip

`ip_addr_t tcp_pcb_listen::local_ip`

Common members of all PCB types

3.126.3.2 next

`struct tcp_pcb_listen* tcp_pcb_listen::next`

Protocol specific PCB members

The documentation for this struct was generated from the following file: src/include/lwip/tcp.h

3.127 tftp_context Struct Reference

```
#include <tftp_common.h>
```

3.127.1 Data Fields

- void *(* open)(const char *fname, const char *mode, u8_t write)
- void(* close)(void *handle)
- int(* read)(void *handle, void *buf, int bytes)
- int(* write)(void *handle, struct pbuf *p)
- void(* error)(void *handle, int err, const char *msg, int size)

3.127.2 Detailed Description

TFTP context containing callback functions for TFTP transfers

3.127.3 Field Documentation

3.127.3.1 close

```
void(* tftp_context::close) (void *handle)
```

Close file handle

Parameters

handle	File handle returned by open()/tftp_put()/tftp_get()

3.127.3.2 error

```
void(* tftp_context::error) (void *handle, int err, const char *msg, int size)
```

Error indication from client or response from server

Parameters

handle	File handle set by open()/tftp_get()/tftp_put()
err	error code from client or server
msg	error message from client or server
size	size of msg

3.127.3.3 open

```
void *(* tftp_context::open) (const char *fname, const char *mode, u8_t write)
```

Open file for read/write (server mode only).

Parameters

Returns File handle supplied to other functions

fname	Filename
mode	Mode string from TFTP RFC 1350 (netascii, octet, mail)
write	Flag indicating read (0) or write (!= 0) access

3.127.3.4 read

```
int(* tftp_context::read) (void *handle, void *buf, int bytes)
```

Read from file

Parameters

handle	File handle returned by open()/tftp_put()/tftp_get()
buf	Target buffer to copy read data to
bytes	Number of bytes to copy to buf

Returns >= 0: Success; < 0: Error

3.127.3.5 write

```
int(* tftp_context::write) (void *handle, struct pbuf *p)
```

Write to file

Parameters

handle	File handle returned by open()/tftp_put()/tftp_get()
pbuf	PBUF adjusted such that payload pointer points to the beginning of write data. In other words, TFTP headers are stripped off.

Returns >= 0: Success; < 0: Error

The documentation for this struct was generated from the following file: src/include/lwip/apps/tftp_common.h

3.128 threadsync_data Struct Reference

```
#include <snmp_threadsync.h>
```

3.128.1 Detailed Description

Thread sync runtime data. For internal usage only.

The documentation for this struct was generated from the following file: src/include/lwip/apps/snmp_threadsync.h

3.129 udp_pcb Struct Reference

```
#include <udp.h>
```

3.129.1 Data Fields

- ip_addr_t local_ip
- u16_t local_port
- ip4_addr_t mcast_ip4
- u8_t mcast_ifindex
- u8_t mcast_ttl
- u16_t chksum_len_rx
- udp_recv_fn recv
- void * recv_arg

3.129.2 Detailed Description

the UDP protocol control block

3.129.3 Field Documentation

3.129.3.1 chksum_len_rx

`u16_t udp_pcb::chksum_len_rx`

used for UDP_LITE only

3.129.3.2 local_ip

`ip_addr_t udp_pcb::local_ip`

Common members of all PCB types

3.129.3.3 local_port

`u16_t udp_pcb::local_port`

ports are in host byte order

3.129.3.4 mcast_ifindex

`u8_t udp_pcb::mcast_ifindex`

outgoing network interface for multicast packets, by interface index (if nonzero)

3.129.3.5 mcast_ip4

`ip4_addr_t udp_pcb::mcast_ip4`

outgoing network interface for multicast packets, by IPv4 address (if not ’any’)

3.129.3.6 mcast_ttl

`u8_t udp_pcb::mcast_ttl`

TTL for outgoing multicast packets

3.129.3.7 recv

`udp_recv_fn udp_pcb::recv`

receive callback function

3.129.3.8 recv_arg

`void* udp_pcb::recv_arg`

user-supplied argument for the recv callback

The documentation for this struct was generated from the following file: src/include/lwip/udp.h

3.130 zepif_init Struct Reference

`#include <zepif.h>`

3.130.1 Data Fields

- u16_t zep_src_udp_port
- u16_t zep_dst_udp_port
- const ip_addr_t * zep_src_ip_addr
- const ip_addr_t * zep_dst_ip_addr
- const struct netif * zep_netif
- u8_t addr [6]

3.130.2 Detailed Description

Pass this struct as ’state’ to netif_add to control the behaviour of this netif. If NULL is passed, default behaviour is chosen

3.130.3 Field Documentation

3.130.3.1 addr

`u8_t zepif_init::addr[6]`

MAC address of the 6LowPAN device

3.130.3.2 zep_dst_ip_addr

`const ip_addr_t* zepif_init::zep_dst_ip_addr`

The IP address to sed ZEP frames to (NULL = BROADCAST)

3.130.3.3 zep_dst_udp_port

`u16_t zepif_init::zep_dst_udp_port`

The UDP port used to ZEP frames to (0 = default)

3.130.3.4 zep_netif

```
const struct netif* zepif_init::zep_netif
```

If != NULL, the udp pcb is bound to this netif

3.130.3.5 zep_src_ip_addr

```
const ip_addr_t* zepif_init::zep_src_ip_addr
```

The IP address to sed ZEP frames from (NULL = ANY)

3.130.3.6 zep_src_udp_port

```
u16_t zepif_init::zep_src_udp_port
```

The UDP port used to ZEP frames from (0 = default)

The documentation for this struct was generated from the following file: src/include/netif/zepif.h

Chapter 4

File Documentation

4.1 src/api/api_lib.c File Reference

```
#include "lwip/opt.h"#include "lwip/api.h"#include "lwip/memp.h"#include "lwip/ip.h"# ↩
    include "lwip/raw.h"#include "lwip/udp.h"#include "lwip/priv/api_msg.h"#include "lwip/ ↩
    priv/tcp_priv.h"#include "lwip/priv/tcpip_priv.h"#include "path/to/my/lwip_hooks.h"# ↩
    include <string.h>
```

4.1.1 Functions

- struct netconn * netconn_new_with_proto_and_callback (enum netconn_type t, u8_t proto, netconn_callback callback)
- err_t netconn_prepare_delete (struct netconn *conn)
- err_t netconn_delete (struct netconn *conn)
- err_t netconn_getaddr (struct netconn *conn, ip_addr_t *addr, u16_t *port, u8_t local)
- err_t netconn_bind (struct netconn *conn, const ip_addr_t *addr, u16_t port)
- err_t netconn_bind_if (struct netconn *conn, u8_t if_idx)
- err_t netconn_connect (struct netconn *conn, const ip_addr_t *addr, u16_t port)
- err_t netconn_disconnect (struct netconn *conn)
- err_t netconn_listen_with_backlog (struct netconn *conn, u8_t backlog)
- err_t netconn_accept (struct netconn *conn, struct netconn **new_conn)
- err_t netconn_recv_tcp_pbuf (struct netconn *conn, struct pbuf **new_buf)
- err_t netconn_recv_tcp_pbuf_flags (struct netconn *conn, struct pbuf **new_buf, u8_t apiflags)
- err_t netconn_recv_udp_raw_netbuf (struct netconn *conn, struct netbuf **new_buf)
- err_t netconn_recv_udp_raw_netbuf_flags (struct netconn *conn, struct netbuf **new_buf, u8_t apiflags)
- err_t netconn_recv (struct netconn *conn, struct netbuf **new_buf)
- err_t netconn_sendto (struct netconn *conn, struct netbuf *buf, const ip_addr_t *addr, u16_t port)
- err_t netconn_send (struct netconn *conn, struct netbuf *buf)
- err_t netconn_write_partly (struct netconn *conn, const void *dataptr, size_t size, u8_t apiflags, size_t *bytes_written)

- err_t netconn_write_vectors_partly (struct netconn *conn, struct netvector *vectors, u16_t vectorcnt, u8_t apiflags, size_t *bytes_written)
- err_t netconn_close (struct netconn *conn)
- err_t netconn_err (struct netconn *conn)
- err_t netconn_shutdown (struct netconn *conn, u8_t shut_rx, u8_t shut_tx)
- err_t netconn_join_leave_group (struct netconn *conn, const ip_addr_t *multiaddr, const ip_addr_t *netif_addr, enum netconn_igmp join_or_leave)
- err_t netconn_join_leave_group_netif (struct netconn *conn, const ip_addr_t *multiaddr, u8_t if_idx, enum netconn_igmp join_or_leave)
- err_t netconn_gethostbyname_addrtype (const char *name, ip_addr_t *addr, u8_t dns_addrtype)

4.1.2 Detailed Description

Sequential API External module

4.1.3 Function Documentation

4.1.3.1 netconn_getaddr()

```
err_t netconn_getaddr (struct netconn * conn, ip_addr_t * addr, u16_t * port, u8_t local)
```

Get the local or remote IP address and port of a netconn. For RAW netconns, this returns the protocol instead of a port!

Parameters

conn	the netconn to query
addr	a pointer to which to save the IP address
port	a pointer to which to save the port (or protocol for RAW)
local	1 to get the local IP address, 0 to get the remote one

Returns ERR_CONN for invalid connections ERR_OK if the information was retrieved

4.1.3.2 netconn_new_with_proto_and_callback()

```
struct netconn * netconn_new_with_proto_and_callback (enum netconn_type t, u8_t proto,
netconn_callback callback)
```

Create a new netconn (of a specific type) that has a callback function. The corresponding pcb is also created.

Parameters

t	the type of 'connection' to create (

See also enum netconn_type)

Parameters

proto	the IP protocol for RAW IP pcbs
callback	a function to call on status changes (RX available, TX'ed)

Returns a newly allocated struct netconn or NULL on memory error

4.1.3.3 netconn_recv_udp_raw_netbuf()

```
err_t netconn_recv_udp_raw_netbuf (struct netconn * conn, struct netbuf ** new_buf)
```

Receive data (in form of a netbuf) from a UDP or RAW netconn

Parameters

conn	the netconn from which to receive data
new_buf	pointer where a new netbuf is stored when received data

Returns ERR_OK if data has been received, an error code otherwise (timeout, memory error or another error) ERR_ARG if conn is not a UDP/RAW netconn

4.1.3.4 netconn_recv_udp_raw_netbuf_flags()

```
err_t netconn_recv_udp_raw_netbuf_flags (struct netconn * conn, struct netbuf ** new_buf,
u8_t apiflags)
```

Receive data (in form of a netbuf) from a UDP or RAW netconn

Parameters

conn	the netconn from which to receive data
new_buf	pointer where a new netbuf is stored when received data
apiflags	flags that control function behaviour. For now only: • NETCONN_DONTBLOCK: only read data that is available now, don't wait for more data

Returns ERR_OK if data has been received, an error code otherwise (timeout, memory error or another error) ERR_ARG if conn is not a UDP/RAW netconn

4.1.3.5 netconn_write_vectors_partly()

```
err_t netconn_write_vectors_partly (struct netconn * conn, struct netvector * vectors,
u16_t vectorcnt, u8_t apiflags, size_t * bytes_written)
```

Send vectorized data atomically over a TCP netconn.

Parameters

conn	the TCP netconn over which to send data
vectors	array of vectors containing data to send
vectorcnt	number of vectors in the array
apiflags	combination of following flags : • NETCONN_COPY: data will be copied into memory belonging to the stack • NETCONN_MORE: for TCP connection, PSH flag will be set on last segment sent • NETCONN_DONTBLOCK: only write the data if all data can be written at once
bytes_written	pointer to a location that receives the number of written bytes

Returns ERR_OK if data was sent, any other err_t on error

4.2 src/api/api_msg.c File Reference

```
#include "lwip/opt.h"#include "lwip/priv/api_msg.h"#include "lwip/ip.h"#include "lwip/ ↩
    ip_addr.h"#include "lwip/udp.h"#include "lwip/tcp.h"#include "lwip/raw.h"#include "lwip/ ↩
    memp.h"#include "lwip/igmp.h"#include "lwip/dns.h"#include "lwip/mld6.h"#include "lwip/ ↩
    priv/tcpip_priv.h"#include <string.h>
```

4.2.1 Functions

- void lwip_netconn_do_newconn (void *m)
- struct netconn * netconn_alloc (enum netconn_type t, netconn_callback callback)
- void netconn_free (struct netconn *conn)
- void lwip_netconn_do_delconn (void *m)
- void lwip_netconn_do_bind (void *m)
- void lwip_netconn_do_bind_if (void *m)
- void lwip_netconn_do_connect (void *m)
- void lwip_netconn_do_disconnect (void *m)
- void lwip_netconn_do_listen (void *m)
- void lwip_netconn_do_send (void *m)
- void lwip_netconn_do_recv (void *m)
- void lwip_netconn_do_accepted (void *m)
- void lwip_netconn_do_write (void *m)
- void lwip_netconn_do_getaddr (void *m)
- void lwip_netconn_do_close (void *m)
- void lwip_netconn_do_join_leave_group (void *m)
- void lwip_netconn_do_join_leave_group_netif (void *m)
- void lwip_netconn_do_gethostbyname (void *arg)

4.2.2 Detailed Description

Sequential API Internal module

4.2.3 Function Documentation

4.2.3.1 lwip_netconn_do_accepted()

```
void lwip_netconn_do_accepted (void * m)
```

Indicate that a TCP pcb has been accepted Called from netconn_accept

Parameters

m	the api_msg pointing to the connection

4.2.3.2 lwip_netconn_do_bind()

```
void lwip_netconn_do_bind (void * m)
```

Bind a pcb contained in a netconn Called from netconn_bind.

Parameters

m	the api_msg pointing to the connection and containing the IP address and port to bind to

4.2.3.3 lwip_netconn_do_bind_if()

```
void lwip_netconn_do_bind_if (void * m)
```

Bind a pcb contained in a netconn to an interface Called from netconn_bind_if.

Parameters

m	the api_msg pointing to the connection and containing the IP address and port to bind to

4.2.3.4 lwip_netconn_do_close()

```
void lwip_netconn_do_close (void * m)
```

Close or half-shutdown a TCP pcb contained in a netconn Called from netconn_close In contrast to closing sockets, the netconn is not deallocated.

Parameters

4.2.3.5 lwip_netconn_do_connect()

```
void lwip_netconn_do_connect (void * m)
```

Connect a pcb contained inside a netconn Called from netconn_connect.

Parameters

4.2.3.6 lwip_netconn_do_delconn()

```
void lwip_netconn_do_delconn (void * m)
```

Delete the pcb inside a netconn. Called from netconn_delete.

Parameters

4.2.3.7 lwip_netconn_do_disconnect()

```
void lwip_netconn_do_disconnect (void * m)
```

Disconnect a pcb contained inside a netconn Only used for UDP netconns. Called from netconn_disconnect.

Parameters

m	the api_msg pointing to the connection

m	the api_msg pointing to the connection and containing the IP address and port to connect to

4.2.3.8 lwip_netconn_do_getaddr()

```
void lwip_netconn_do_getaddr (void * m)
```

Return a connection's local or remote address Called from netconn_getaddr

Parameters

4.2.3.9 lwip_netconn_do_gethostbyname()

```
void lwip_netconn_do_gethostbyname (void * arg)
```

Execute a DNS query Called from netconn_gethostbyname

Parameters

4.2.3.10 lwip_netconn_do_join_leave_group()

```
void lwip_netconn_do_join_leave_group (void * m)
```

Join multicast groups for UDP netconns. Called from netconn_join_leave_group

Parameters

4.2.3.11 lwip_netconn_do_join_leave_group_netif()

```
void lwip_netconn_do_join_leave_group_netif (void * m)
```

Join multicast groups for UDP netconns. Called from netconn_join_leave_group_netif

Parameters

4.2.3.12 lwip_netconn_do_listen()

```
void lwip_netconn_do_listen (void * m)
```

Set a TCP pcb contained in a netconn into listen mode Called from netconn_listen.

Parameters

4.2.3.13 lwip_netconn_do_newconn()

```
void lwip_netconn_do_newconn (void * m)
```

Create a new pcb of a specific type inside a netconn. Called from netconn_new_with_proto_and_callback.

Parameters

m	the api_msg pointing to the connection

m	the api_msg pointing to the connection to disconnect

m	the api_msg pointing to the connection

4.2.3.14 lwip_netconn_do_recv()

```
void lwip_netconn_do_recv (void * m)
```

Indicate data has been received from a TCP pcb contained in a netconn Called from netconn_recv

Parameters

4.2.3.15 lwip_netconn_do_send()

```
void lwip_netconn_do_send (void * m)
```

Send some data on a RAW or UDP pcb contained in a netconn Called from netconn_send

Parameters

4.2.3.16 lwip_netconn_do_write()

```
void lwip_netconn_do_write (void * m)
```

Send some data on a TCP pcb contained in a netconn Called from netconn_write

Parameters

4.2.3.17 netconn_alloc()

```
struct netconn * netconn_alloc (enum netconn_type t, netconn_callback callback)
```

Create a new netconn (of a specific type) that has a callback function. The corresponding pcb is NOT created!

Parameters

See also enum netconn_type)

Parameters

Returns a newly allocated struct netconn or NULL on memory error

4.2.3.18 netconn_free()

```
void netconn_free (struct netconn * conn)
```

Delete a netconn and all its resources. The pcb is NOT freed (since we might not be in the right thread context do this).

Parameters

4.3 src/api/err.c File Reference

```
#include "lwip/err.h"#include "lwip/def.h"#include "lwip/sys.h"#include "lwip/errno.h"
```

arg	the dns_api_msg pointing to the query

m	the api_msg pointing to the connection

m	the api_msg pointing to the connection

4.3.1 Detailed Description

Error Management module

4.4 src/api/if_api.c File Reference

```
#include "lwip/opt.h"#include "lwip/errno.h"#include "lwip/if_api.h"#include "lwip/netifapi ↩
    .h"#include "lwip/priv/sockets_priv.h"
```

4.4.1 Functions

- char * lwip_if_indextoname (unsigned int ifindex, char *ifname)
- unsigned int lwip_if_nametoindex (const char *ifname)

4.4.2 Detailed Description

Interface Identification APIs from: RFC 3493: Basic Socket Interface Extensions for IPv6 Section 4: Interface Identification

4.5 src/api/netbuf.c File Reference

```
#include "lwip/opt.h"#include "lwip/netbuf.h"#include "lwip/memp.h"#include <string.h>
```

4.5.1 Functions

- struct netbuf * netbuf_new (void)
- void netbuf_delete (struct netbuf *buf)
- void * netbuf_alloc (struct netbuf *buf, u16_t size)
- void netbuf_free (struct netbuf *buf)
- err_t netbuf_ref (struct netbuf *buf, const void *dataptr, u16_t size)
- void netbuf_chain (struct netbuf *head, struct netbuf *tail)
- err_t netbuf_data (struct netbuf *buf, void **dataptr, u16_t *len)
- s8_t netbuf_next (struct netbuf *buf)
- void netbuf_first (struct netbuf *buf)

m	the api_msg pointing to the connection

m	the api_msg describing the connection type

m	the api_msg pointing to the connection

4.5.2 Detailed Description

Network buffer management

4.6 src/api/netdb.c File Reference

```
#include "lwip/netdb.h"#include "lwip/err.h"#include "lwip/errno.h"#include "lwip/mem.h"# ↩
    include "lwip/memp.h"#include "lwip/ip_addr.h"#include "lwip/api.h"#include "lwip/dns.h ↩
    "#include <string.h>#include <stdlib.h>
```

4.6.1 Data Structures

- struct gethostbyname_r_helper

4.6.2 Macros

- #define LWIP_DNS_API_HOSTENT_STORAGE 0

4.6.3 Functions

- struct hostent * lwip_gethostbyname (const char *name)
- int lwip_gethostbyname_r (const char *name, struct hostent *ret, char *buf, size_t buflen, struct hostent **result, int *h_errnop)
- void lwip_freeaddrinfo (struct addrinfo *ai)
- int lwip_getaddrinfo (const char *nodename, const char *servname, const struct addrinfo *hints, struct addrinfo **res)

4.6.4 Variables

- int h_errno

4.6.5 Detailed Description

API functions for name resolving

m	the api_msg pointing to the connection

m	the api_msg pointing to the connection

t	the type of 'connection' to create (

4.6.6 Macro Definition Documentation

4.6.6.1 LWIP_DNS_API_HOSTENT_STORAGE

```
#define LWIP_DNS_API_HOSTENT_STORAGE   0
```

LWIP_DNS_API_HOSTENT_STORAGE: if set to 0 (default), lwip_gethostbyname() returns the same global variable for all calls (in all threads). When set to 1, your port should provide a function struct hostent* sys_thread_hostent(struct hostent* h); which have to do a copy of "h" and return a pointer ont the "per-thread" copy.

4.6.7 Function Documentation

4.6.7.1 lwip_freeaddrinfo()

```
void lwip_freeaddrinfo (struct addrinfo * ai)
```

Frees one or more addrinfo structures returned by getaddrinfo(), along with any additional storage associated with those structures. If the ai_next field of the structure is not null, the entire list of structures is freed.

Parameters

4.6.7.2 lwip_getaddrinfo()

```
int lwip_getaddrinfo (const char * nodename, const char * servname, const struct addrinfo
* hints, struct addrinfo ** res)
```

Translates the name of a service location (for example, a host name) and/or a service name and returns a set of socket addresses and associated information to be used in creating a socket with which to address the specified service. Memory for the result is allocated internally and must be freed by calling lwip_freeaddrinfo()!

Due to a limitation in dns_gethostbyname, only the first address of a host is returned. Also, service names are not supported (only port numbers)!

Parameters

Returns 0 on success, non-zero on failure

4.6.7.3 lwip_gethostbyname()

```
struct hostent * lwip_gethostbyname (const char * name)
```

Returns an entry containing addresses of address family AF_INET for the host with name name. Due to dns_gethostbyname limitations, only one address is returned.

Parameters

Returns an entry containing addresses of address family AF_INET for the host with name name

callback	a function to call on status changes (RX available, TX'ed)

conn	the netconn to free

ai	struct addrinfo to free

4.6.7.4 lwip_gethostbyname_r()

```
int lwip_gethostbyname_r (const char * name, struct hostent * ret, char * buf, size_t bufl
struct hostent ** result, int * h_errnop)
```

Thread-safe variant of lwip_gethostbyname: instead of using a static buffer, this function takes buffer and errno pointers as arguments and uses these for the result.

Parameters

Returns 0 on success, non-zero on error, additional error information is stored in *h_errnop instead of h_errno to be thread-safe

4.6.8 Variable Documentation

4.6.8.1 h_errno

```
int h_errno
```

h_errno is exported in netdb.h for access by applications.

4.7 src/api/netifapi.c File Reference

```
#include "lwip/opt.h"#include "lwip/etharp.h"#include "lwip/netifapi.h"#include "lwip/memp. ↩
    h"#include "lwip/priv/tcpip_priv.h"#include <string.h>
```

4.7.1 Functions

- err_t netifapi_arp_add (const ip4_addr_t *ipaddr, struct eth_addr *ethaddr, enum netifapi_arp_entry type)
- err_t netifapi_arp_remove (const ip4_addr_t *ipaddr, enum netifapi_arp_entry type)
- err_t netifapi_netif_add (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw, void *state, netif_init_fn init, netif_input_fn input)
- err_t netifapi_netif_set_addr (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw)
- err_t netifapi_netif_common (struct netif *netif, netifapi_void_fn voidfunc, netifapi_errt_fn errtfunc)
- err_t netifapi_netif_name_to_index (const char *name, u8_t *idx)
- err_t netifapi_netif_index_to_name (u8_t idx, char *name)

nodename	descriptive name or address string of the host (may be NULL -> local address)
servname	port number as string of NULL
hints	structure containing input values that set socktype and protocol
res	pointer to a pointer where to store the result (set to NULL on failure)

name	the hostname to resolve

name	the hostname to resolve
ret	pre-allocated struct where to store the result
buf	pre-allocated buffer where to store additional data
buflen	the size of buf
result	pointer to a hostent pointer that is set to ret on success and set to zero on error
h_errnop	pointer to an int where to store errors (instead of modifying the global h_errno)

4.7.2 Detailed Description

Network Interface Sequential API module

4.7.3 Function Documentation

4.7.3.1 netifapi_arp_add()

```
err_t netifapi_arp_add (const ip4_addr_t * ipaddr, struct eth_addr * ethaddr, enum netifap
type)
```

Add or update an entry in the ARP cache. For an update, ipaddr is used to find the cache entry.

Parameters

ipaddr	IPv4 address of cache entry
ethaddr	hardware address mapped to ipaddr
type	type of ARP cache entry

Returns ERR_OK: entry added/updated, else error from err_t

4.7.3.2 netifapi_arp_remove()

```
err_t netifapi_arp_remove (const ip4_addr_t * ipaddr, enum netifapi_arp_entry type)
```

Remove an entry in the ARP cache identified by ipaddr

Parameters

Returns ERR_OK: entry removed, else error from err_t

4.7.3.3 netifapi_netif_common()

```
err_t netifapi_netif_common (struct netif * netif, netifapi_void_fn voidfunc, netifapi_err
errtfunc)
```

call the "errtfunc" (or the "voidfunc" if "errtfunc" is NULL) in a thread-safe way by running that function inside the tcpip_thread context.

Note
use only for functions where there is only "netif" parameter.

ipaddr	IPv4 address of cache entry
type	type of ARP cache entry

4.8 src/api/sockets.c File Reference

```
#include "lwip/opt.h"#include "lwip/sockets.h"#include "lwip/priv/sockets_priv.h"#include " ↩
    lwip/api.h"#include "lwip/igmp.h"#include "lwip/inet.h"#include "lwip/tcp.h"#include " ↩
    lwip/raw.h"#include "lwip/udp.h"#include "lwip/memp.h"#include "lwip/pbuf.h"#include " ↩
    lwip/netif.h"#include "lwip/priv/tcpip_priv.h"#include "lwip/mld6.h"#include <string.h># ↩
    include "path/to/my/lwip_hooks.h"
```

4.8.1 Data Structures

- union sockaddr_aligned

4.8.2 Enumerations

- enum lwip_pollscan_opts { LWIP_POLLSCAN_CLEAR = 1 , LWIP_POLLSCAN_INC_WAIT = 2 , LWIP_POLLSCAN_DEC_WAI
  = 4 }

4.8.3 Functions

- void lwip_socket_thread_init (void)
- void lwip_socket_thread_cleanup (void)
- int lwip_listen (int s, int backlog)
- int lwip_shutdown (int s, int how)
- int lwip_fcntl (int s, int cmd, int val)

4.8.4 Detailed Description

Sockets BSD-Like API module

4.8.5 Enumeration Type Documentation

4.8.5.1 lwip_pollscan_opts

`enum lwip_pollscan_opts`

Options for the lwip_pollscan function.

LWIP_POLLSCAN_CLEAR	Clear revents in each struct pollfd.
LWIP_POLLSCAN_INC_WAIT	Increment select_waiting in each struct lwip_sock.
LWIP_POLLSCAN_DEC_WAIT	Decrement select_waiting in each struct lwip_sock.

4.8.6 Function Documentation

4.8.6.1 lwip_fcntl()

```
int lwip_fcntl (int s, int cmd, int val)
```

A minimal implementation of fcntl. Currently only the commands F_GETFL and F_SETFL are implemented. The flag O_NONBLOCK and access modes are supported for F_GETFL, only the flag O_NONBLOCK is implemented for F_SETFL.

4.8.6.2 lwip_listen()

```
int lwip_listen (int s, int backlog)
```

Set a socket into listen mode. The socket may not have been used for another connection previously.

Parameters

s	the socket to set to listening mode
backlog	(ATTENTION: needs TCP_LISTEN_BACKLOG=1)

Returns 0 on success, non-zero on failure

4.8.6.3 lwip_shutdown()

```
int lwip_shutdown (int s, int how)
```

Close one end of a full-duplex connection.

4.8.6.4 lwip_socket_thread_cleanup()

```
void lwip_socket_thread_cleanup (void )
```

LWIP_NETCONN_SEM_PER_THREAD==1: destroy thread-local semaphore

4.8.6.5 lwip_socket_thread_init()

```
void lwip_socket_thread_init (void )
```

LWIP_NETCONN_SEM_PER_THREAD==1: initialize thread-local semaphore

4.9 src/api/tcpip.c File Reference

```
#include "lwip/opt.h"#include "lwip/priv/tcpip_priv.h"#include "lwip/sys.h"#include "lwip/ ↩
    memp.h"#include "lwip/mem.h"#include "lwip/init.h"#include "lwip/ip.h"#include "lwip/ ↩
    pbuf.h"#include "lwip/etharp.h"#include "netif/ethernet.h"
```

4.9.1 Functions

- err_t tcpip_inpkt (struct pbuf *p, struct netif *inp, netif_input_fn input_fn)
- err_t tcpip_input (struct pbuf *p, struct netif *inp)
- err_t tcpip_callback (tcpip_callback_fn function, void *ctx)

- err_t tcpip_try_callback (tcpip_callback_fn function, void *ctx)
- err_t tcpip_send_msg_wait_sem (tcpip_callback_fn fn, void *apimsg, sys_sem_t *sem)
- err_t tcpip_api_call (tcpip_api_call_fn fn, struct tcpip_api_call_data *call)
- struct tcpip_callback_msg * tcpip_callbackmsg_new (tcpip_callback_fn function, void *ctx)
- void tcpip_callbackmsg_delete (struct tcpip_callback_msg *msg)
- err_t tcpip_callbackmsg_trycallback (struct tcpip_callback_msg *msg)
- err_t tcpip_callbackmsg_trycallback_fromisr (struct tcpip_callback_msg *msg)
- err_t tcpip_callback_wait (tcpip_callback_fn function, void *ctx)
- void tcpip_init (tcpip_init_done_fn initfunc, void *arg)
- err_t pbuf_free_callback (struct pbuf *p)
- err_t mem_free_callback (void *m)

4.9.2 Variables

- sys_mutex_t lock_tcpip_core

4.9.3 Detailed Description

Sequential API Main thread module

4.9.4 Function Documentation

4.9.4.1 mem_free_callback()

`err_t mem_free_callback (void * m)`

A simple wrapper function that allows you to free heap memory from interrupt context.

Parameters

m	the heap memory to free

Returns ERR_OK if callback could be enqueued, an err_t if not

4.9.4.2 pbuf_free_callback()

`err_t pbuf_free_callback (struct pbuf * p)`

A simple wrapper function that allows you to free a pbuf from interrupt context.

Parameters

p	The pbuf (chain) to be dereferenced.

Returns ERR_OK if callback could be enqueued, an err_t if not

4.9.4.3 tcpip_api_call()

```
err_t tcpip_api_call (tcpip_api_call_fn fn, struct tcpip_api_call_data * call)
```

Synchronously calls function in TCPIP thread and waits for its completion. It is recommended to use LWIP_TCPIP_CORE_LOCKING (preferred) or LWIP_NETCONN_SEM_PER_THREAD. If not, a semaphore is created and destroyed on every call which is usually an expensive/slow operation.

Parameters

fn	Function to call
call	Call parameters

Returns Return value from tcpip_api_call_fn

4.9.4.4 tcpip_callback_wait()

```
err_t tcpip_callback_wait (tcpip_callback_fn function, void * ctx)
```

Sends a message to TCPIP thread to call a function. Caller thread blocks until the function returns. It is recommended to use LWIP_TCPIP_CORE_LOCKING (preferred) or LWIP_NETCONN_SEM_PER_THREAD. If not, a semaphore is created and destroyed on every call which is usually an expensive/slow operation.

Parameters

function	the function to call
ctx	parameter passed to f

Returns ERR_OK if the function was called, another err_t if not

4.9.4.5 tcpip_inpkt()

```
err_t tcpip_inpkt (struct pbuf * p, struct netif * inp, netif_input_fn input_fn)
```

Pass a received packet to tcpip_thread for input processing

Parameters

p	the received packet
inp	the network interface on which the packet was received
input_fn	input function to call

4.9.4.6 tcpip_send_msg_wait_sem()

```
err_t tcpip_send_msg_wait_sem (tcpip_callback_fn fn, void * apimsg, sys_sem_t * sem)
```

Sends a message to TCPIP thread to call a function. Caller thread blocks on on a provided semaphore, which is NOT automatically signalled by TCPIP thread, this has to be done by the user. It is recommended to use LWIP_TCPIP_CORE_LOCKING since this is the way with least runtime overhead.

Parameters

Returns ERR_OK if the function was called, another err_t if not

fn	function to be called from TCPIP thread
apimsg	argument to API function
sem	semaphore to wait on

4.9.5 Variable Documentation

4.9.5.1 lock_tcpip_core

`sys_mutex_t lock_tcpip_core`

The global semaphore to lock the stack.

4.10 src/apps/altcp_tls/altcp_tls_mbedtls.c File Reference

```
#include "lwip/opt.h"#include "lwip/sys.h"#include "lwip/apps/altcp_tls_mbedtls_opts.h"
```

4.10.1 Detailed Description

Application layered TCP/TLS connection API (to be used from TCPIP thread)

This file provides a TLS layer using mbedTLS

This version is currently compatible with the 2.x.x branch (current LTS).

4.11 src/apps/altcp_tls/altcp_tls_mbedtls_mem.c File Reference

```
#include "lwip/opt.h"#include "lwip/apps/altcp_tls_mbedtls_opts.h"
```

4.11.1 Detailed Description

Application layered TCP connection API (to be used from TCPIP thread)

This file contains memory management functions for a TLS layer using mbedTLS.

ATTENTION: For production usage, you might want to override this file with your own implementation since this implementation simply uses the lwIP heap without caring for fragmentation or leaving heap for other parts of lwIP!

4.12 src/apps/altcp_tls/altcp_tls_mbedtls_mem.h File Reference

```
#include "lwip/opt.h"#include "lwip/apps/altcp_tls_mbedtls_opts.h"
```

4.12.1 Detailed Description

Application layered TCP/TLS connection API (to be used from TCPIP thread)

This file contains memory management function prototypes for a TLS layer using mbedTLS.

Memory management contains:

- allocating/freeing altcp_mbedtls_state_t
- allocating/freeing memory used in the mbedTLS library

4.13 src/apps/altcp_tls/altcp_tls_mbedtls_structs.h File Reference

```
#include "lwip/opt.h"#include "lwip/apps/altcp_tls_mbedtls_opts.h"
```

4.13.1 Detailed Description

Application layered TCP/TLS connection API (to be used from TCPIP thread)

This file contains structure definitions for a TLS layer using mbedTLS.

4.14 src/apps/http/altcp_proxyconnect.c File Reference

```
#include "lwip/apps/altcp_proxyconnect.h"#include "lwip/altcp.h"#include "lwip/priv/ ↩
    altcp_priv.h"#include "lwip/altcp_tcp.h"#include "lwip/altcp_tls.h"#include "lwip/mem.h ↩
    "#include "lwip/init.h"#include <stdio.h>
```

4.14.1 Macros

- #define ALTCP_PROXYCONNECT_CLIENT_AGENT "lwIP/" LWIP_VERSION_STRING " (http://savannah.nongnu.org/projects/

4.14.2 Functions

- struct altcp_pcb * altcp_proxyconnect_new (struct altcp_proxyconnect_config *config, struct altcp_pcb *inner_pcb)
- struct altcp_pcb * altcp_proxyconnect_new_tcp (struct altcp_proxyconnect_config *config, u8_t ip_type)
- struct altcp_pcb * altcp_proxyconnect_alloc (void *arg, u8_t ip_type)
- struct altcp_pcb * altcp_proxyconnect_tls_alloc (void *arg, u8_t ip_type)

4.14.3 Detailed Description

Application layered TCP connection API that executes a proxy-connect.

This file provides a starting layer that executes a proxy-connect e.g. to set up TLS connections through a http proxy.

4.14.4 Macro Definition Documentation

4.14.4.1 ALTCP_PROXYCONNECT_CLIENT_AGENT

```
#define ALTCP_PROXYCONNECT_CLIENT_AGENT   "lwIP/" LWIP_VERSION_STRING " (http://savannah.n
```

This string is passed in the HTTP header as "User-Agent: "

4.14.5 Function Documentation

4.14.5.1 altcp_proxyconnect_alloc()

```
struct altcp_pcb * altcp_proxyconnect_alloc (void * arg, u8_t ip_type)
```

Allocator function to allocate a proxy connect altcp pcb connecting directly via tcp to the proxy.

The returned pcb is a chain: altcp_proxyconnect - altcp_tcp - tcp pcb

This function is meant for use with altcp_new.

Parameters

arg	struct altcp_proxyconnect_config that contains the proxy settings
ip_type	IP type of the connection (lwip_ip_addr_type)

4.14.5.2 altcp_proxyconnect_new()

```
struct altcp_pcb * altcp_proxyconnect_new (struct altcp_proxyconnect_config * config, stru
altcp_pcb * inner_pcb)
```

Allocate a new altcp layer connecting through a proxy. This function gets the inner pcb passed.

Parameters

config	struct altcp_proxyconnect_config that contains the proxy settings
inner_pcb	pcb that makes the connection to the proxy (i.e. tcp pcb)

4.14.5.3 altcp_proxyconnect_new_tcp()

```
struct altcp_pcb * altcp_proxyconnect_new_tcp (struct altcp_proxyconnect_config * config,
u8_t ip_type)
```

Allocate a new altcp layer connecting through a proxy. This function allocates the inner pcb as tcp pcb, resulting in a direct tcp connection to the proxy.

Parameters

config	struct altcp_proxyconnect_config that contains the proxy settings
ip_type	IP type of the connection (lwip_ip_addr_type)

4.14.5.4 altcp_proxyconnect_tls_alloc()

```
struct altcp_pcb * altcp_proxyconnect_tls_alloc (void * arg, u8_t ip_type)
```

Allocator function to allocate a TLS connection through a proxy.

The returned pcb is a chain: altcp_tls - altcp_proxyconnect - altcp_tcp - tcp pcb

This function is meant for use with altcp_new.

Parameters

4.15 src/apps/http/http_client.c File Reference

```
#include "lwip/apps/http_client.h"#include "lwip/altcp_tcp.h"#include "lwip/dns.h"#include  ←
    "lwip/debug.h"#include "lwip/mem.h"#include "lwip/altcp_tls.h"#include "lwip/init.h"# ←
    include <stdio.h>#include <stdlib.h>#include <string.h>
```

4.15.1 Macros

- #define HTTPC_DEBUG LWIP_DBG_OFF
- #define HTTPC_DEBUG_REQUEST 0
- #define HTTPC_CLIENT_AGENT "lwIP/" LWIP_VERSION_STRING " (http://savannah.nongnu.org/projects/lwip)"

arg	struct altcp_proxyconnect_tls_config that contains the proxy settings and tls settings
ip_type	IP type of the connection (lwip_ip_addr_type)

4.15.2 Functions

- err_t httpc_get_file (const ip_addr_t *server_addr, u16_t port, const char *uri, const httpc_connection_t *settings, altcp_recv_fn recv_fn, void *callback_arg, httpc_state_t **connection)
- err_t httpc_get_file_dns (const char *server_name, u16_t port, const char *uri, const httpc_connection_t *settings, altcp_recv_fn recv_fn, void *callback_arg, httpc_state_t **connection)

4.15.3 Detailed Description

HTTP client

4.15.4 Macro Definition Documentation

4.15.4.1 HTTPC_CLIENT_AGENT

```
#define HTTPC_CLIENT_AGENT   "lwIP/" LWIP_VERSION_STRING " (http://savannah.nongnu.org/pro
```

This string is passed in the HTTP header as "User-Agent: "

4.15.4.2 HTTPC_DEBUG

```
#define HTTPC_DEBUG   LWIP_DBG_OFF
```

HTTPC_DEBUG: Enable debugging for HTTP client.

4.15.4.3 HTTPC_DEBUG_REQUEST

```
#define HTTPC_DEBUG_REQUEST   0
```

Set this to 1 to keep server name and uri in request state

4.16 src/apps/http/httpd.c File Reference

```
#include "lwip/init.h"#include "lwip/apps/httpd.h"#include "lwip/debug.h"#include "lwip/ ←
    stats.h"#include "lwip/apps/fs.h"#include "httpd_structs.h"#include "lwip/def.h"#include ←
     "lwip/altcp.h"#include "lwip/altcp_tcp.h"#include "lwip/altcp_tls.h"#include "path/to/ ←
    my/lwip_hooks.h"#include <string.h>#include <stdlib.h>#include <stdio.h>
```

4.16.1 Macros

- #define MIN_REQ_LEN   7
- #define HTTP_IS_DATA_VOLATILE(hs)   (HTTP_IS_DYNAMIC_FILE(hs) ? TCP_WRITE_FLAG_COPY : 0)
- #define HTTP_IS_HDR_VOLATILE(hs, ptr)   0

4.16.2 Functions

- void httpd_post_data_recved (void *connection, u16_t recved_len)
- void httpd_init (void)
- void httpd_inits (struct altcp_tls_config *conf)
- void http_set_ssi_handler (tSSIHandler ssi_handler, const char **tags, int num_tags)
- void http_set_cgi_handlers (const tCGI *cgis, int num_handlers)

4.16.3 Detailed Description

LWIP HTTP server implementation

4.16.4 Macro Definition Documentation

4.16.4.1 HTTP_IS_DATA_VOLATILE

```
#define HTTP_IS_DATA_VOLATILE( hs)    (HTTP_IS_DYNAMIC_FILE(hs) ?  TCP_WRITE_FLAG_COPY :
0)
```

tcp_write does not have to copy data when sent from rom-file-system directly

4.16.4.2 HTTP_IS_HDR_VOLATILE

```
#define HTTP_IS_HDR_VOLATILE( hs, ptr)    0
```

Default: dynamic headers are sent from ROM (non-dynamic headers are handled like file data)

4.16.4.3 MIN_REQ_LEN

```
#define MIN_REQ_LEN    7
```

Minimum length for a valid HTTP/0.9 request: "GET /\r\n" -> 7 bytes

4.17 src/apps/lwiperf/lwiperf.c File Reference

```
#include "lwip/apps/lwiperf.h"#include "lwip/tcp.h"#include "lwip/sys.h"#include "lwip/inet ↩
    .h"#include <string.h>
```

4.17.1 Data Structures

- struct _lwiperf_settings
- struct _lwiperf_state_tcp

4.17.2 Macros

- #define LWIPERF_TCP_MAX_IDLE_SEC   10U
- #define LWIPERF_SERVER_IP_TYPE IPADDR_TYPE_ANY
- #define LWIPERF_CHECK_RX_DATA   0

4.17.3 Typedefs

- typedef struct _lwiperf_settings lwiperf_settings_t
- typedef struct _lwiperf_state_tcp lwiperf_state_tcp_t

4.17.4 Functions

- void * lwiperf_start_tcp_server_default (lwiperf_report_fn report_fn, void *report_arg)
- void * lwiperf_start_tcp_server (const ip_addr_t *local_addr, u16_t local_port, lwiperf_report_fn report_fn, void *report_arg)
- void * lwiperf_start_tcp_client_default (const ip_addr_t *remote_addr, lwiperf_report_fn report_fn, void *report_arg)
- void * lwiperf_start_tcp_client (const ip_addr_t *remote_addr, u16_t remote_port, enum lwiperf_client_type type, lwiperf_report_fn report_fn, void *report_arg)
- void lwiperf_abort (void *lwiperf_session)

4.17.5 Detailed Description

lwIP iPerf server implementation

4.17.6 Macro Definition Documentation

4.17.6.1 LWIPERF_CHECK_RX_DATA

```
#define LWIPERF_CHECK_RX_DATA   0
```

If this is 1, check that received data has the correct format

4.17.6.2 LWIPERF_SERVER_IP_TYPE

```
#define LWIPERF_SERVER_IP_TYPE   IPADDR_TYPE_ANY
```

Change this if you don't want to lwiperf to listen to any IP version

4.17.6.3 LWIPERF_TCP_MAX_IDLE_SEC

```
#define LWIPERF_TCP_MAX_IDLE_SEC   10U
```

Specify the idle timeout (in seconds) after that the test fails

4.17.7 Typedef Documentation

4.17.7.1 lwiperf_settings_t

```
typedef struct _lwiperf_settings lwiperf_settings_t
```

This is the Iperf settings struct sent from the client

4.17.7.2 lwiperf_state_tcp_t

```
typedef struct _lwiperf_state_tcp lwiperf_state_tcp_t
```

Connection handle for a TCP iperf session

4.18 src/apps/mdns/mdns.c File Reference

```
#include "lwip/apps/mdns.h"#include "lwip/apps/mdns_priv.h"#include "lwip/apps/mdns_domain. ↩
    h"#include "lwip/apps/mdns_out.h"#include "lwip/netif.h"#include "lwip/udp.h"#include " ↩
    lwip/ip_addr.h"#include "lwip/mem.h"#include "lwip/memp.h"#include "lwip/prot/dns.h"# ↩
    include "lwip/prot/iana.h"#include "lwip/timeouts.h"#include "lwip/sys.h"#include < ↩
    string.h>#include <stdio.h>#include "lwip/igmp.h"#include "lwip/mld6.h"
```

4.18.1 Data Structures

- struct mdns_packet

4.18.2 Macros

- #define MDNS_RESPONSE_DELAY_MAX 120
- #define MDNS_PROBE_COUNT 3

4.18.3 Functions

- struct mdns_host * netif_mdns_data (struct netif *netif)
- struct udp_pcb * get_mdns_pcb (void)
- err_t mdns_resp_add_netif (struct netif *netif, const char *hostname)
- err_t mdns_resp_remove_netif (struct netif *netif)
- err_t mdns_resp_rename_netif (struct netif *netif, const char *hostname)
- int mdns_resp_netif_active (struct netif *netif)
- s8_t mdns_resp_add_service (struct netif *netif, const char *name, const char *service, enum mdns_sd_proto proto, u16_t port, service_get_txt_fn_t txt_fn, void *txt_data)
- err_t mdns_resp_del_service (struct netif *netif, u8_t slot)
- err_t mdns_resp_rename_service (struct netif *netif, u8_t slot, const char *name)
- err_t mdns_resp_add_service_txtitem (struct mdns_service *service, const char *txt, u8_t txt_len)
- void mdns_search_stop (u8_t request_id)
- err_t mdns_search_service (const char *name, const char *service, enum mdns_sd_proto proto, struct netif *netif, search_result_fn_t result_fn, void *arg, u8_t *request_id)
- void mdns_resp_announce (struct netif *netif)
- void mdns_resp_register_name_result_cb (mdns_name_result_cb_t cb)
- void mdns_resp_restart_delay (struct netif *netif, uint32_t delay)
- void mdns_resp_restart (struct netif *netif)
- void mdns_resp_init (void)
- void * mdns_get_service_txt_userdata (struct netif *netif, s8_t slot)

4.18.4 Detailed Description

MDNS responder implementation

4.18.5 Macro Definition Documentation

4.18.5.1 MDNS_PROBE_COUNT

```
#define MDNS_PROBE_COUNT   3
```

Probing & announcing defines

4.18.5.2 MDNS_RESPONSE_DELAY_MAX

```
#define MDNS_RESPONSE_DELAY_MAX   120
```

Delayed response defines

4.18.6 Function Documentation

4.18.6.1 get_mdns_pcb()

```
struct udp_pcb * get_mdns_pcb (void )
```

Construction to access the mdns udp pcb.

Returns udp_pcb struct of mdns

4.18.6.2 mdns_resp_register_name_result_cb()

```
void mdns_resp_register_name_result_cb (mdns_name_result_cb_t cb)
```

Register a callback function that is called if probing is completed successfully or with a conflict.

4.18.6.3 netif_mdns_data()

```
struct mdns_host * netif_mdns_data (struct netif * netif)
```

Construction to make mdns struct accessible from mdns_out.c TODO: can we add the mdns struct to the netif like we do for dhcp, autoip,...? Then this is not needed any more.

Parameters

netif	The network interface

Returns mdns struct

4.19 src/apps/mdns/mdns_domain.c File Reference

```
#include "lwip/apps/mdns.h"#include "lwip/apps/mdns_domain.h"#include "lwip/apps/mdns_priv. ↩
    h"#include "lwip/prot/dns.h"#include <string.h>#include "lwip/prot/ip6.h"
```

4.19.1 Functions

- err_t mdns_domain_add_label (struct mdns_domain *domain, const char *label, u8_t len)
- err_t mdns_domain_add_domain (struct mdns_domain *domain, struct mdns_domain *source)
- err_t mdns_domain_add_string (struct mdns_domain *domain, const char *source)
- u16_t mdns_readname (struct pbuf *p, u16_t offset, struct mdns_domain *domain)
- void mdns_domain_debug_print (struct mdns_domain *domain)
- int mdns_domain_eq (struct mdns_domain *a, struct mdns_domain *b)
- err_t mdns_build_reverse_v4_domain (struct mdns_domain *domain, const ip4_addr_t *addr)
- err_t mdns_build_reverse_v6_domain (struct mdns_domain *domain, const ip6_addr_t *addr)
- err_t mdns_build_host_domain (struct mdns_domain *domain, struct mdns_host *mdns)
- err_t mdns_build_dnssd_domain (struct mdns_domain *domain)
- err_t mdns_build_service_domain (struct mdns_domain *domain, struct mdns_service *service, int include_name)
- err_t mdns_build_request_domain (struct mdns_domain *domain, struct mdns_request *request, int include_name)
- u16_t mdns_compress_domain (struct pbuf *pbuf, u16_t *offset, struct mdns_domain *domain)
- err_t mdns_write_domain (struct mdns_outpacket *outpkt, struct mdns_domain *domain)

4.19.2 Detailed Description

MDNS responder implementation - domain related functionalities

4.19.3 Function Documentation

4.19.3.1 mdns_build_dnssd_domain()

```
err_t mdns_build_dnssd_domain (struct mdns_domain * domain)
```

Build the lookup-all-services special DNS-SD domain name

Parameters

domain	Where to write the domain name

Returns ERR_OK if domain _services._dns-sd._udp.local. was written, an err_t otherwise

4.19.3.2 mdns_build_host_domain()

```
err_t mdns_build_host_domain (struct mdns_domain * domain, struct mdns_host * mdns)
```

Build the <hostname>.local. domain name

Parameters

domain	Where to write the domain name
mdns	TMDNS netif descriptor.

Returns ERR_OK if domain <hostname>.local. was written, an err_t otherwise

4.19.3.3 mdns_build_request_domain()

```
err_t mdns_build_request_domain (struct mdns_domain * domain, struct mdns_request * reques
int include_name)
```

Build domain name for a request

Parameters

domain	Where to write the domain name
request	The request struct, containing service name, type and protocol
include_name	Whether to include the service name in the domain

Returns ERR_OK if domain was written. If service name is included, <name>.<type>.<proto>.local. will be written, otherwise <type>.<proto>.local. An err_t is returned on error.

4.19.3.4 mdns_build_reverse_v4_domain()

```
err_t mdns_build_reverse_v4_domain (struct mdns_domain * domain, const ip4_addr_t * addr)
```

Build domain for reverse lookup of IPv4 address like 12.0.168.192.in-addr.arpa. for 192.168.0.12

Parameters

domain	Where to write the domain name
addr	Pointer to an IPv4 address to encode

Returns ERR_OK if domain was written, an err_t otherwise

4.19.3.5 mdns_build_reverse_v6_domain()

```
err_t mdns_build_reverse_v6_domain (struct mdns_domain * domain, const ip6_addr_t * addr)
```

Build domain for reverse lookup of IP address like b.a.9.8.7.6.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. for 2001:db8::567:89ab

Parameters

domain	Where to write the domain name
addr	Pointer to an IPv6 address to encode

Returns ERR_OK if domain was written, an err_t otherwise

4.19.3.6 mdns_build_service_domain()

```
err_t mdns_build_service_domain (struct mdns_domain * domain, struct mdns_service * servic
int include_name)
```

Build domain name for a service

Parameters

Returns ERR_OK if domain was written. If service name is included, <name>.<type>.<proto>.local. will be written, otherwise <type>.<proto>.local. An err_t is returned on error.

domain	Where to write the domain name
service	The service struct, containing service name, type and protocol
include_name	Whether to include the service name in the domain

4.19.3.7 mdns_compress_domain()

```
u16_t mdns_compress_domain (struct pbuf * pbuf, u16_t * offset, struct mdns_domain * domai
```

Return bytes needed to write before jump for best result of compressing supplied domain against domain in outpacket starting at specified offset. If a match is found, offset is updated to where to jump to

Parameters

pbuf	Pointer to pbuf with the partially constructed DNS packet
offset	Start position of a domain written earlier. If this location is suitable for compression, the pointer is updated to where in the domain to jump to.
domain	The domain to write

Returns Number of bytes to write of the new domain before writing a jump to the offset. If compression can not be done against this previous domain name, the full new domain length is returned.

4.19.3.8 mdns_domain_add_domain()

```
err_t mdns_domain_add_domain (struct mdns_domain * domain, struct mdns_domain * source)
```

Add a partial domain to a domain

Parameters

domain	The domain to add a label to
source	The domain to add, like <\x09_services\007_dns-sd\000>

Returns ERR_OK on success, an err_t otherwise if label too long

4.19.3.9 mdns_domain_add_label()

```
err_t mdns_domain_add_label (struct mdns_domain * domain, const char * label, u8_t len)
```

Add a label part to a domain

Parameters

Returns ERR_OK on success, an err_t otherwise if label too long

4.19.3.10 mdns_domain_add_string()

```
err_t mdns_domain_add_string (struct mdns_domain * domain, const char * source)
```

Add a string domain to a domain

Parameters

Returns ERR_OK on success, an err_t otherwise if label too long

domain	The domain to add a label to
label	The label to add, like <hostname>, 'local', 'com' or "
len	The length of the label

domain	The domain to add a label to
source	The string to add, like <_services._dns-sd>

4.19.3.11 mdns_domain_debug_print()

```
void mdns_domain_debug_print (struct mdns_domain * domain)
```

Print domain name to debug output

Parameters

domain	The domain name

4.19.3.12 mdns_domain_eq()

```
int mdns_domain_eq (struct mdns_domain * a, struct mdns_domain * b)
```

Return 1 if contents of domains match (case-insensitive)

Parameters

Returns 1 if domains are equal ignoring case, 0 otherwise

4.19.3.13 mdns_readname()

```
u16_t mdns_readname (struct pbuf * p, u16_t offset, struct mdns_domain * domain)
```

Read possibly compressed domain name from packet buffer

Parameters

Returns The new offset after the domain, or MDNS_READNAME_ERROR if reading failed

4.19.3.14 mdns_write_domain()

```
err_t mdns_write_domain (struct mdns_outpacket * outpkt, struct mdns_domain * domain)
```

Write domain to outpacket. Compression will be attempted, unless domain->skip_compression is set.

Parameters

Returns ERR_OK on success, an err_t otherwise

4.20 src/apps/mdns/mdns_out.c File Reference

```
#include "lwip/apps/mdns_out.h"#include "lwip/apps/mdns_priv.h"#include "lwip/apps/ ↩
    mdns_domain.h"#include "lwip/prot/dns.h"#include "lwip/prot/iana.h"#include "lwip/udp.h ↩
    "#include <string.h>#include "lwip/prot/ip6.h"
```

a	Domain name to compare 1
b	Domain name to compare 2

p	The packet
offset	start position of domain name in packet
domain	The domain name destination

4.20.1 Functions

- void mdns_prepare_txtdata (struct mdns_service *service)
- err_t mdns_create_outpacket (struct netif *netif, struct mdns_outmsg *msg, struct mdns_outpacket *outpkt)
- err_t mdns_send_outpacket (struct mdns_outmsg *msg, struct netif *netif)
- void mdns_multicast_timeout_reset_ipv4 (void *arg)
- void mdns_multicast_probe_timeout_reset_ipv4 (void *arg)
- void mdns_multicast_timeout_25ttl_reset_ipv4 (void *arg)
- void mdns_send_multicast_msg_delayed_ipv4 (void *arg)
- void mdns_send_unicast_msg_delayed_ipv4 (void *arg)
- void mdns_start_multicast_timeouts_ipv4 (struct netif *netif)
- void mdns_multicast_timeout_reset_ipv6 (void *arg)
- void mdns_multicast_probe_timeout_reset_ipv6 (void *arg)
- void mdns_multicast_timeout_25ttl_reset_ipv6 (void *arg)
- void mdns_send_multicast_msg_delayed_ipv6 (void *arg)
- void mdns_send_unicast_msg_delayed_ipv6 (void *arg)
- void mdns_start_multicast_timeouts_ipv6 (struct netif *netif)
- void mdns_set_timeout (struct netif *netif, u32_t msecs, sys_timeout_handler handler, u8_t *busy_flag)
- err_t mdns_send_request (struct mdns_request *req, struct netif *netif, const ip_addr_t *destination)

4.20.2 Detailed Description

MDNS responder implementation - output related functionalities

4.20.3 Function Documentation

4.20.3.1 mdns_create_outpacket()

```
err_t mdns_create_outpacket (struct netif * netif, struct mdns_outmsg * msg, struct mdns_o
* outpkt)
```

Create packet with chosen answers as a reply

Add all selected answers / questions Add additional answers based on the selected answers

outpkt	The outpacket to write to
domain	The domain name to write

4.20.3.2 mdns_multicast_probe_timeout_reset_ipv4()

```
void mdns_multicast_probe_timeout_reset_ipv4 (void * arg)
```

Called by timeouts when timer is passed, allows direct multicast IPv4 probe response traffic again and sends out probe response if one was pending

Parameters

arg	pointer to netif of timeout.

4.20.3.3 mdns_multicast_probe_timeout_reset_ipv6()

```
void mdns_multicast_probe_timeout_reset_ipv6 (void * arg)
```

Called by timeouts when timer is passed, allows direct multicast IPv6 probe response traffic again and sends out probe response if one was pending

Parameters

arg	pointer to netif of timeout.

4.20.3.4 mdns_multicast_timeout_25ttl_reset_ipv4()

```
void mdns_multicast_timeout_25ttl_reset_ipv4 (void * arg)
```

Called by timeouts when timer is passed, allows to send an answer on a QU question via multicast.

Parameters

arg	pointer to netif of timeout.

4.20.3.5 mdns_multicast_timeout_25ttl_reset_ipv6()

```
void mdns_multicast_timeout_25ttl_reset_ipv6 (void * arg)
```

Called by timeouts when timer is passed, allows to send an answer on a QU question via multicast.

Parameters

4.20.3.6 mdns_multicast_timeout_reset_ipv4()

```
void mdns_multicast_timeout_reset_ipv4 (void * arg)
```

Called by timeouts when timer is passed, allows multicast IPv4 traffic again.

Parameters

4.20.3.7 mdns_multicast_timeout_reset_ipv6()

```
void mdns_multicast_timeout_reset_ipv6 (void * arg)
```

Called by timeouts when timer is passed, allows multicast IPv6 traffic again.

Parameters

arg	pointer to netif of timeout.

arg	pointer to netif of timeout.

4.20.3.8 mdns_prepare_txtdata()

`void mdns_prepare_txtdata (struct mdns_service * service)`

Call user supplied function to setup TXT data

Parameters

4.20.3.9 mdns_send_multicast_msg_delayed_ipv4()

`void mdns_send_multicast_msg_delayed_ipv4 (void * arg)`

Called by timeouts when timer is passed, sends out delayed multicast IPv4 response.

Parameters

4.20.3.10 mdns_send_multicast_msg_delayed_ipv6()

`void mdns_send_multicast_msg_delayed_ipv6 (void * arg)`

Called by timeouts when timer is passed, sends out delayed multicast IPv6 response.

Parameters

4.20.3.11 mdns_send_outpacket()

`err_t mdns_send_outpacket (struct mdns_outmsg * msg, struct netif * netif)`

Send chosen answers as a reply

Create the packet Send the packet

4.20.3.12 mdns_send_request()

```
err_t mdns_send_request (struct mdns_request * req, struct netif * netif, const ip_addr_t
* destination)
```

Send search request containing all our known data

Parameters

4.20.3.13 mdns_send_unicast_msg_delayed_ipv4()

`void mdns_send_unicast_msg_delayed_ipv4 (void * arg)`

Called by timeouts when timer is passed, sends out delayed unicast IPv4 response.

Parameters

arg	pointer to netif of timeout.

service	The service to build TXT record for

arg	pointer to netif of timeout.

4.20.3.14 mdns_send_unicast_msg_delayed_ipv6()

```
void mdns_send_unicast_msg_delayed_ipv6 (void * arg)
```

Called by timeouts when timer is passed, sends out delayed unicast IPv6 response.

Parameters

4.20.3.15 mdns_set_timeout()

```
void mdns_set_timeout (struct netif * netif, u32_t msecs, sys_timeout_handler handler,
u8_t * busy_flag)
```

Sets a timer that calls the handler when finished. Depending on the busy_flag the timer is restarted or started. The flag is set before return. Sys_timeout does not give us this functionality.

Parameters

4.20.3.16 mdns_start_multicast_timeouts_ipv4()

```
void mdns_start_multicast_timeouts_ipv4 (struct netif * netif)
```

Start all multicast timeouts for IPv4 Timeouts started:

- do not multicast within one second
- do not multicast a probe response within 250ms
- send a multicast answer on a QU question if not send recently.

Parameters

4.20.3.17 mdns_start_multicast_timeouts_ipv6()

```
void mdns_start_multicast_timeouts_ipv6 (struct netif * netif)
```

Start all multicast timeouts for IPv6 Timeouts started:

- do not multicast within one second
- do not multicast a probe response within 250ms
- send a multicast answer on a QU question if not send recently.

Parameters

arg	pointer to netif of timeout.

req	The request to send
netif	The network interface to send on
destination	The target address to send to (usually multicast address)

arg	pointer to netif of timeout.

4.21 src/apps/mqtt/mqtt.c File Reference

```
#include "lwip/apps/mqtt.h"#include "lwip/apps/mqtt_priv.h"#include "lwip/timeouts.h"# ↩
    include "lwip/ip_addr.h"#include "lwip/mem.h"#include "lwip/err.h"#include "lwip/pbuf.h ↩
    "#include "lwip/altcp.h"#include "lwip/altcp_tcp.h"#include "lwip/altcp_tls.h"#include < ↩
    string.h>
```

4.21.1 Macros

- #define MQTT_DEBUG LWIP_DBG_OFF
- #define MQTT_CTL_PACKET_TYPE(fixed_hdr_byte0) ((fixed_hdr_byte0 & 0xf0) >> 4)
- #define mqtt_ringbuf_free(rb) (MQTT_OUTPUT_RINGBUF_SIZE - mqtt_ringbuf_len(rb))
- #define mqtt_ringbuf_linear_read_length(rb) LWIP_MIN(mqtt_ringbuf_len(rb), (MQTT_OUTPUT_RINGBUF_SIZE - (rb)->get))

4.21.2 Enumerations

- enum
- enum mqtt_message_type
- enum mqtt_connect_flag

4.21.3 Functions

- err_t mqtt_publish (mqtt_client_t *client, const char *topic, const void *payload, u16_t payload_length, u8_t qos, u8_t retain, mqtt_request_cb_t cb, void *arg)
- err_t mqtt_sub_unsub (mqtt_client_t *client, const char *topic, u8_t qos, mqtt_request_cb_t cb, void *arg, u8_t sub)
- void mqtt_set_inpub_callback (mqtt_client_t *client, mqtt_incoming_publish_cb_t pub_cb, mqtt_incoming_data_cb_t data_cb, void *arg)
- mqtt_client_t * mqtt_client_new (void)
- void mqtt_client_free (mqtt_client_t *client)
- err_t mqtt_client_connect (mqtt_client_t *client, const ip_addr_t *ip_addr, u16_t port, mqtt_connection_cb_t cb, void *arg, const struct mqtt_connect_client_info_t *client_info)
- void mqtt_disconnect (mqtt_client_t *client)
- u8_t mqtt_client_is_connected (mqtt_client_t *client)

arg	pointer to netif of timeout.

netif	Network interface info
msecs	Time value to set
handler	Callback function to call
busy_flag	Pointer to flag that displays if the timer is running or not.

netif	network interface to start timeouts on

4.21.4 Detailed Description

MQTT client

4.21.5 Macro Definition Documentation

4.21.5.1 MQTT_CTL_PACKET_TYPE

```
#define MQTT_CTL_PACKET_TYPE( fixed_hdr_byte0)   ((fixed_hdr_byte0 & 0xf0) >> 4)
```

Helpers to extract control packet type and qos from first byte in fixed header

4.21.5.2 MQTT_DEBUG

```
#define MQTT_DEBUG   LWIP_DBG_OFF
```

MQTT_DEBUG: Default is off.

4.21.5.3 mqtt_ringbuf_free

```
#define mqtt_ringbuf_free( rb)   (MQTT_OUTPUT_RINGBUF_SIZE - mqtt_ringbuf_len(rb))
```

Return number of bytes free in ring buffer

4.21.5.4 mqtt_ringbuf_linear_read_length

```
#define mqtt_ringbuf_linear_read_length( rb)   LWIP_MIN(mqtt_ringbuf_len(rb), (MQTT_OUTPUT_
- (rb)->get))
```

Return number of bytes possible to read without wrapping around

4.21.6 Enumeration Type Documentation

4.21.6.1 anonymous enum

```
anonymous enum
```

MQTT client connection states

4.21.6.2 mqtt_connect_flag

```
enum mqtt_connect_flag
```

MQTT connect flags, only used in CONNECT message

netif	network interface to start timeouts on

4.21.6.3 mqtt_message_type

```
enum mqtt_message_type
```

MQTT control message types

4.22 src/apps/netbiosns/netbiosns.c File Reference

```
#include "lwip/apps/netbiosns.h"#include "lwip/def.h"#include "lwip/udp.h"#include "lwip/ip ↩
    .h"#include "lwip/netif.h"#include "lwip/prot/iana.h"#include <string.h>#include "arch/ ↩
    bpstruct.h"#include "arch/epstruct.h"
```

4.22.1 Data Structures

- struct netbios_hdr
- struct netbios_question_hdr
- struct netbios_name_hdr
- struct netbios_resp
- struct netbios_answer

4.22.2 Macros

- #define NETBIOS_NAME_LEN 16
- #define NETBIOS_NAME_TTL 300000u
- #define NETB_HFLAG_RESPONSE 0x8000U
- #define NETB_NFLAG_UNIQUE 0x8000U

4.22.3 Functions

- void netbiosns_init (void)
- void netbiosns_stop (void)

4.22.4 Detailed Description

NetBIOS name service responder

4.22.5 Macro Definition Documentation

4.22.5.1 NETB_HFLAG_RESPONSE

```
#define NETB_HFLAG_RESPONSE   0x8000U
```

NetBIOS header flags

4.22.5.2 NETB_NFLAG_UNIQUE

```
#define NETB_NFLAG_UNIQUE   0x8000U
```

NetBIOS name flags

4.22.5.3 NETBIOS_NAME_LEN

```
#define NETBIOS_NAME_LEN   16
```

size of a NetBIOS name

4.22.5.4 NETBIOS_NAME_TTL

```
#define NETBIOS_NAME_TTL   300000u
```

The Time-To-Live for NetBIOS name responds (in seconds) Default is 300000 seconds (3 days, 11 hours, 20 minutes)

4.23 src/apps/smtp/smtp.c File Reference

```
#include "lwip/apps/smtp.h"#include "lwip/sys.h"#include "lwip/sockets.h"#include "lwip/ ↩
    altcp.h"#include "lwip/dns.h"#include "lwip/mem.h"#include "lwip/altcp_tcp.h"#include " ↩
    lwip/altcp_tls.h"#include <string.h>#include <stdlib.h>
```

4.23.1 Data Structures

- struct smtp_session

4.23.2 Macros

- #define SMTP_POLL_INTERVAL 4
- #define SMTP_TIMEOUT_DATABLOCK (3 * 60 * SMTP_POLL_INTERVAL / 2)
- #define SMTP_TIMEOUT_DATATERM (10 * 60 * SMTP_POLL_INTERVAL / 2)
- #define SMTP_TIMEOUT (2 * 60 * SMTP_POLL_INTERVAL / 2)

4.23.3 Enumerations

- enum smtp_session_state

4.23.4 Functions

- err_t smtp_set_server_addr (const char *server)
- void smtp_set_server_port (u16_t port)
- void smtp_set_tls_config (struct altcp_tls_config *tls_config)
- err_t smtp_set_auth (const char *username, const char *pass)
- err_t smtp_send_mail (const char *from, const char *to, const char *subject, const char *body, smtp_result_fn callback_fn, void *callback_arg)
- err_t smtp_send_mail_static (const char *from, const char *to, const char *subject, const char *body, smtp_result_fn callback_fn, void *callback_arg)
- void smtp_send_mail_int (void *arg)

4.23.5 Detailed Description

SMTP client module

Author: Simon Goldschmidt

4.23.6 Macro Definition Documentation

4.23.6.1 SMTP_POLL_INTERVAL

```
#define SMTP_POLL_INTERVAL   4
```

TCP poll interval. Unit is 0.5 sec.

4.23.6.2 SMTP_TIMEOUT

```
#define SMTP_TIMEOUT   ( 2 * 60 * SMTP_POLL_INTERVAL / 2)
```

TCP poll timeout while not sending the body. This is somewhat lower than the RFC states (5 minutes for initial, MAIL and RCPT) but still OK for us here. 2 minutes

4.23.6.3 SMTP_TIMEOUT_DATABLOCK

```
#define SMTP_TIMEOUT_DATABLOCK   ( 3 * 60 * SMTP_POLL_INTERVAL / 2)
```

TCP poll timeout while sending message body, reset after every successful write. 3 minutes

4.23.6.4 SMTP_TIMEOUT_DATATERM

```
#define SMTP_TIMEOUT_DATATERM   (10 * 60 * SMTP_POLL_INTERVAL / 2)
```

TCP poll timeout while waiting for confirmation after sending the body. 10 minutes

4.23.7 Enumeration Type Documentation

4.23.7.1 smtp_session_state

```
enum smtp_session_state
```

State for SMTP client state machine

4.24 src/apps/snmp/snmp_asn1.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include "snmp_asn1.h"
```

4.24.1 Functions

- err_t snmp_ans1_enc_tlv (struct snmp_pbuf_stream *pbuf_stream, struct snmp_asn1_tlv *tlv)
- err_t snmp_asn1_enc_raw (struct snmp_pbuf_stream *pbuf_stream, const u8_t *raw, u16_t raw_len)
- err_t snmp_asn1_enc_u32t (struct snmp_pbuf_stream *pbuf_stream, u16_t octets_needed, u32_t value)
- err_t snmp_asn1_enc_s32t (struct snmp_pbuf_stream *pbuf_stream, u16_t octets_needed, s32_t value)

- err_t snmp_asn1_enc_oid (struct snmp_pbuf_stream *pbuf_stream, const u32_t *oid, u16_t oid_len)
- void snmp_asn1_enc_length_cnt (u16_t length, u8_t *octets_needed)
- void snmp_asn1_enc_u32t_cnt (u32_t value, u16_t *octets_needed)
- void snmp_asn1_enc_s32t_cnt (s32_t value, u16_t *octets_needed)
- void snmp_asn1_enc_oid_cnt (const u32_t *oid, u16_t oid_len, u16_t *octets_needed)
- err_t snmp_asn1_dec_tlv (struct snmp_pbuf_stream *pbuf_stream, struct snmp_asn1_tlv *tlv)
- err_t snmp_asn1_dec_u32t (struct snmp_pbuf_stream *pbuf_stream, u16_t len, u32_t *value)
- err_t snmp_asn1_dec_s32t (struct snmp_pbuf_stream *pbuf_stream, u16_t len, s32_t *value)
- err_t snmp_asn1_dec_oid (struct snmp_pbuf_stream *pbuf_stream, u16_t len, u32_t *oid, u8_t *oid_len, u8_t oid_max_len)
- err_t snmp_asn1_dec_raw (struct snmp_pbuf_stream *pbuf_stream, u16_t len, u8_t *buf, u16_t *buf_len, u16_t buf_max_len)

4.24.2 Detailed Description

Abstract Syntax Notation One (ISO 8824, 8825) encoding

4.24.3 Function Documentation

4.24.3.1 snmp_ans1_enc_tlv()

```
err_t snmp_ans1_enc_tlv (struct snmp_pbuf_stream * pbuf_stream, struct snmp_asn1_tlv *
tlv)
```

Encodes a TLV into a pbuf stream.

Parameters

pbuf_stream	points to a pbuf stream
tlv	TLV to encode

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) encode

4.24.3.2 snmp_asn1_dec_oid()

```
err_t snmp_asn1_dec_oid (struct snmp_pbuf_stream * pbuf_stream, u16_t len, u32_t * oid,
u8_t * oid_len, u8_t oid_max_len)
```

Decodes object identifier from incoming message into array of u32_t.

Parameters

pbuf_stream	points to a pbuf stream
len	length of the coded object identifier
oid	return decoded object identifier
oid_len	return decoded object identifier length
oid_max_len	size of oid buffer

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) decode

4.24.3.3 snmp_asn1_dec_raw()

```
err_t snmp_asn1_dec_raw (struct snmp_pbuf_stream * pbuf_stream, u16_t len, u8_t * buf,
u16_t * buf_len, u16_t buf_max_len)
```

Decodes (copies) raw data (ip-addresses, octet strings, opaque encoding) from incoming message into array.

Parameters

pbuf_stream	points to a pbuf stream
len	length of the coded raw data (zero is valid, e.g. empty string!)
buf	return raw bytes
buf_len	returns length of the raw return value
buf_max_len	buffer size

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) decode

4.24.3.4 snmp_asn1_dec_s32t()

```
err_t snmp_asn1_dec_s32t (struct snmp_pbuf_stream * pbuf_stream, u16_t len, s32_t * value)
```

Decodes integer into s32_t.

Parameters

pbuf_stream	points to a pbuf stream
len	length of the coded integer field
value	return host order integer

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) decode

Note
ASN coded integers are *always* signed!

4.24.3.5 snmp_asn1_dec_tlv()

```
err_t snmp_asn1_dec_tlv (struct snmp_pbuf_stream * pbuf_stream, struct snmp_asn1_tlv *
tlv)
```

Decodes a TLV from a pbuf stream.

Parameters

pbuf_stream	points to a pbuf stream
tlv	returns decoded TLV

Returns ERR_OK if successful, ERR_VAL if we can't decode

pbuf_stream	points to a pbuf stream
len	length of the coded integer field
value	return host order integer

4.24.3.6 snmp_asn1_dec_u32t()

```
err_t snmp_asn1_dec_u32t (struct snmp_pbuf_stream * pbuf_stream, u16_t len, u32_t * value)
```

Decodes positive integer (counter, gauge, timeticks) into u32_t.

Parameters

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) decode

Note
ASN coded integers are *always* signed. E.g. +0xFFFF is coded as 0x00,0xFF,0xFF. Note the leading sign octet. A positive value of 0xFFFFFFFF is preceded with 0x00 and the length is 5 octets!!

4.24.3.7 snmp_asn1_enc_length_cnt()

```
void snmp_asn1_enc_length_cnt (u16_t length, u8_t * octets_needed)
```

Returns octet count for length.

Parameters

length	parameter length
octets_needed	points to the return value

4.24.3.8 snmp_asn1_enc_oid()

```
err_t snmp_asn1_enc_oid (struct snmp_pbuf_stream * pbuf_stream, const u32_t * oid, u16_t
oid_len)
```

Encodes object identifier into a pbuf chained ASN1 msg.

Parameters

pbuf_stream	points to a pbuf stream
oid	points to object identifier array
oid_len	object identifier array length

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) encode

4.24.3.9 snmp_asn1_enc_oid_cnt()

```
void snmp_asn1_enc_oid_cnt (const u32_t * oid, u16_t oid_len, u16_t * octets_needed)
```

Returns octet count for an object identifier.

Parameters

oid	points to object identifier array
oid_len	object identifier array length
octets_needed	points to the return value

4.24.3.10 snmp_asn1_enc_raw()

```
err_t snmp_asn1_enc_raw (struct snmp_pbuf_stream * pbuf_stream, const u8_t * raw, u16_t
raw_len)
```

Encodes raw data (octet string, opaque) into a pbuf chained ASN1 msg.

Parameters

pbuf_stream	points to a pbuf stream
raw_len	raw data length
raw	points raw data

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) encode

4.24.3.11 snmp_asn1_enc_s32t()

```
err_t snmp_asn1_enc_s32t (struct snmp_pbuf_stream * pbuf_stream, u16_t octets_needed, s32_
value)
```

Encodes s32_t integer into a pbuf chained ASN1 msg.

Parameters

pbuf_stream	points to a pbuf stream
octets_needed	encoding length (from snmp_asn1_enc_s32t_cnt())
value	is the host order s32_t value to be encoded

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) encode

See also snmp_asn1_enc_s32t_cnt()

4.24.3.12 snmp_asn1_enc_s32t_cnt()

```
void snmp_asn1_enc_s32t_cnt (s32_t value, u16_t * octets_needed)
```

Returns octet count for an s32_t.

Parameters

Note
ASN coded integers are *always* signed.

value	value to be encoded
octets_needed	points to the return value

pbuf_stream	points to a pbuf stream
octets_needed	encoding length (from snmp_asn1_enc_u32t_cnt())
value	is the host order u32_t value to be encoded

4.24.3.13 snmp_asn1_enc_u32t()

```
err_t snmp_asn1_enc_u32t (struct snmp_pbuf_stream * pbuf_stream, u16_t octets_needed, u32_
value)
```

Encodes u32_t (counter, gauge, timeticks) into a pbuf chained ASN1 msg.

Parameters

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) encode

See also snmp_asn1_enc_u32t_cnt()

4.24.3.14 snmp_asn1_enc_u32t_cnt()

```
void snmp_asn1_enc_u32t_cnt (u32_t value, u16_t * octets_needed)
```

Returns octet count for an u32_t.

Parameters

value	value to be encoded
octets_needed	points to the return value

Note
ASN coded integers are *always* signed. E.g. +0xFFFF is coded as 0x00,0xFF,0xFF. Note the leading sign octet. A positive value of 0xFFFFFFFF is preceded with 0x00 and the length is 5 octets!!

4.25 src/apps/snmp/snmp_asn1.h File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/err.h"#include "lwip/apps/snmp_core.h"# ↩
    include "snmp_pbuf_stream.h"
```

4.25.1 Functions

- err_t snmp_asn1_dec_tlv (struct snmp_pbuf_stream *pbuf_stream, struct snmp_asn1_tlv *tlv)
- err_t snmp_asn1_dec_u32t (struct snmp_pbuf_stream *pbuf_stream, u16_t len, u32_t *value)
- err_t snmp_asn1_dec_s32t (struct snmp_pbuf_stream *pbuf_stream, u16_t len, s32_t *value)
- err_t snmp_asn1_dec_oid (struct snmp_pbuf_stream *pbuf_stream, u16_t len, u32_t *oid, u8_t *oid_len, u8_t oid_max_len)
- err_t snmp_asn1_dec_raw (struct snmp_pbuf_stream *pbuf_stream, u16_t len, u8_t *buf, u16_t *buf_len, u16_t buf_max_len)

- err_t snmp_ans1_enc_tlv (struct snmp_pbuf_stream *pbuf_stream, struct snmp_asn1_tlv *tlv)
- void snmp_asn1_enc_length_cnt (u16_t length, u8_t *octets_needed)
- void snmp_asn1_enc_u32t_cnt (u32_t value, u16_t *octets_needed)
- void snmp_asn1_enc_s32t_cnt (s32_t value, u16_t *octets_needed)
- void snmp_asn1_enc_oid_cnt (const u32_t *oid, u16_t oid_len, u16_t *octets_needed)
- err_t snmp_asn1_enc_oid (struct snmp_pbuf_stream *pbuf_stream, const u32_t *oid, u16_t oid_len)
- err_t snmp_asn1_enc_s32t (struct snmp_pbuf_stream *pbuf_stream, u16_t octets_needed, s32_t value)
- err_t snmp_asn1_enc_u32t (struct snmp_pbuf_stream *pbuf_stream, u16_t octets_needed, u32_t value)
- err_t snmp_asn1_enc_raw (struct snmp_pbuf_stream *pbuf_stream, const u8_t *raw, u16_t raw_len)

4.25.2 Detailed Description

Abstract Syntax Notation One (ISO 8824, 8825) codec.

4.25.3 Function Documentation

4.25.3.1 snmp_ans1_enc_tlv()

`err_t snmp_ans1_enc_tlv (struct snmp_pbuf_stream * pbuf_stream, struct snmp_asn1_tlv * tlv)`

Encodes a TLV into a pbuf stream.

Parameters

pbuf_stream	points to a pbuf stream
tlv	TLV to encode

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) encode

4.25.3.2 snmp_asn1_dec_oid()

`err_t snmp_asn1_dec_oid (struct snmp_pbuf_stream * pbuf_stream, u16_t len, u32_t * oid, u8_t * oid_len, u8_t oid_max_len)`

Decodes object identifier from incoming message into array of u32_t.

Parameters

pbuf_stream	points to a pbuf stream
len	length of the coded object identifier
oid	return decoded object identifier
oid_len	return decoded object identifier length
oid_max_len	size of oid buffer

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) decode

4.25.3.3 snmp_asn1_dec_raw()

```
err_t snmp_asn1_dec_raw (struct snmp_pbuf_stream * pbuf_stream, u16_t len, u8_t * buf,
u16_t * buf_len, u16_t buf_max_len)
```

Decodes (copies) raw data (ip-addresses, octet strings, opaque encoding) from incoming message into array.

Parameters

pbuf_stream	points to a pbuf stream
len	length of the coded raw data (zero is valid, e.g. empty string!)
buf	return raw bytes
buf_len	returns length of the raw return value
buf_max_len	buffer size

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) decode

4.25.3.4 snmp_asn1_dec_s32t()

```
err_t snmp_asn1_dec_s32t (struct snmp_pbuf_stream * pbuf_stream, u16_t len, s32_t * value)
```

Decodes integer into s32_t.

Parameters

pbuf_stream	points to a pbuf stream
len	length of the coded integer field
value	return host order integer

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) decode

Note
ASN coded integers are *always* signed!

4.25.3.5 snmp_asn1_dec_tlv()

```
err_t snmp_asn1_dec_tlv (struct snmp_pbuf_stream * pbuf_stream, struct snmp_asn1_tlv *
tlv)
```

Decodes a TLV from a pbuf stream.

Parameters

pbuf_stream	points to a pbuf stream
tlv	returns decoded TLV

Returns ERR_OK if successful, ERR_VAL if we can't decode

pbuf_stream	points to a pbuf stream
len	length of the coded integer field
value	return host order integer

4.25.3.6 snmp_asn1_dec_u32t()

```
err_t snmp_asn1_dec_u32t (struct snmp_pbuf_stream * pbuf_stream, u16_t len, u32_t * value)
```

Decodes positive integer (counter, gauge, timeticks) into u32_t.

Parameters

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) decode

Note
ASN coded integers are *always* signed. E.g. +0xFFFF is coded as 0x00,0xFF,0xFF. Note the leading sign octet. A positive value of 0xFFFFFFFF is preceded with 0x00 and the length is 5 octets!!

4.25.3.7 snmp_asn1_enc_length_cnt()

```
void snmp_asn1_enc_length_cnt (u16_t length, u8_t * octets_needed)
```

Returns octet count for length.

Parameters

length	parameter length
octets_needed	points to the return value

4.25.3.8 snmp_asn1_enc_oid()

```
err_t snmp_asn1_enc_oid (struct snmp_pbuf_stream * pbuf_stream, const u32_t * oid, u16_t
oid_len)
```

Encodes object identifier into a pbuf chained ASN1 msg.

Parameters

pbuf_stream	points to a pbuf stream
oid	points to object identifier array
oid_len	object identifier array length

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) encode

4.25.3.9 snmp_asn1_enc_oid_cnt()

```
void snmp_asn1_enc_oid_cnt (const u32_t * oid, u16_t oid_len, u16_t * octets_needed)
```

Returns octet count for an object identifier.

Parameters

oid	points to object identifier array
oid_len	object identifier array length
octets_needed	points to the return value

4.25.3.10 snmp_asn1_enc_raw()

```
err_t snmp_asn1_enc_raw (struct snmp_pbuf_stream * pbuf_stream, const u8_t * raw, u16_t
raw_len)
```

Encodes raw data (octet string, opaque) into a pbuf chained ASN1 msg.

Parameters

pbuf_stream	points to a pbuf stream
raw_len	raw data length
raw	points raw data

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) encode

4.25.3.11 snmp_asn1_enc_s32t()

```
err_t snmp_asn1_enc_s32t (struct snmp_pbuf_stream * pbuf_stream, u16_t octets_needed, s32_
value)
```

Encodes s32_t integer into a pbuf chained ASN1 msg.

Parameters

pbuf_stream	points to a pbuf stream
octets_needed	encoding length (from snmp_asn1_enc_s32t_cnt())
value	is the host order s32_t value to be encoded

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) encode

See also snmp_asn1_enc_s32t_cnt()

4.25.3.12 snmp_asn1_enc_s32t_cnt()

```
void snmp_asn1_enc_s32t_cnt (s32_t value, u16_t * octets_needed)
```

Returns octet count for an s32_t.

Parameters

Note
ASN coded integers are *always* signed.

value	value to be encoded
octets_needed	points to the return value

pbuf_stream	points to a pbuf stream
octets_needed	encoding length (from snmp_asn1_enc_u32t_cnt())
value	is the host order u32_t value to be encoded

4.25.3.13 snmp_asn1_enc_u32t()

```
err_t snmp_asn1_enc_u32t (struct snmp_pbuf_stream * pbuf_stream, u16_t octets_needed, u32_
value)
```

Encodes u32_t (counter, gauge, timeticks) into a pbuf chained ASN1 msg.

Parameters

Returns ERR_OK if successful, ERR_ARG if we can't (or won't) encode

See also snmp_asn1_enc_u32t_cnt()

4.25.3.14 snmp_asn1_enc_u32t_cnt()

```
void snmp_asn1_enc_u32t_cnt (u32_t value, u16_t * octets_needed)
```

Returns octet count for an u32_t.

Parameters

value	value to be encoded
octets_needed	points to the return value

Note
ASN coded integers are *always* signed. E.g. +0xFFFF is coded as 0x00,0xFF,0xFF. Note the leading sign octet. A positive value of 0xFFFFFFFF is preceded with 0x00 and the length is 5 octets!!

4.26 src/apps/snmp/snmp_core.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/apps/snmp.h"#include "lwip/apps/snmp_core.h ↩
    "#include "snmp_core_priv.h"#include "lwip/netif.h"#include <string.h>#include "lwip/ ↩
    apps/snmp_mib2.h"
```

4.26.1 Functions

- void snmp_set_mibs (const struct snmp_mib **mibs, u8_t num_mibs)
- void snmp_set_device_enterprise_oid (const struct snmp_obj_id *device_enterprise_oid)
- const struct snmp_obj_id * snmp_get_device_enterprise_oid (void)
- u8_t snmp_oid_to_ip4 (const u32_t *oid, ip4_addr_t *ip)
- void snmp_ip4_to_oid (const ip4_addr_t *ip, u32_t *oid)

- u8_t snmp_oid_to_ip6 (const u32_t *oid, ip6_addr_t *ip)
- void snmp_ip6_to_oid (const ip6_addr_t *ip, u32_t *oid)
- u8_t snmp_ip_port_to_oid (const ip_addr_t *ip, u16_t port, u32_t *oid)
- u8_t snmp_ip_to_oid (const ip_addr_t *ip, u32_t *oid)
- u8_t snmp_oid_to_ip (const u32_t *oid, u8_t oid_len, ip_addr_t *ip)
- u8_t snmp_oid_to_ip_port (const u32_t *oid, u8_t oid_len, ip_addr_t *ip, u16_t *port)
- void snmp_oid_assign (struct snmp_obj_id *target, const u32_t *oid, u8_t oid_len)
- void snmp_oid_prefix (struct snmp_obj_id *target, const u32_t *oid, u8_t oid_len)
- void snmp_oid_combine (struct snmp_obj_id *target, const u32_t *oid1, u8_t oid1_len, const u32_t *oid2, u8_t oid2_len)
- void snmp_oid_append (struct snmp_obj_id *target, const u32_t *oid, u8_t oid_len)
- s8_t snmp_oid_compare (const u32_t *oid1, u8_t oid1_len, const u32_t *oid2, u8_t oid2_len)
- u8_t snmp_oid_equal (const u32_t *oid1, u8_t oid1_len, const u32_t *oid2, u8_t oid2_len)
- u8_t netif_to_num (const struct netif *netif)
- const struct snmp_node * snmp_mib_tree_resolve_exact (const struct snmp_mib *mib, const u32_t *oid, u8_t oid_len, u8_t *oid_instance_len)
- void snmp_next_oid_init (struct snmp_next_oid_state *state, const u32_t *start_oid, u8_t start_oid_len, u32_t *next_oid_buf, u8_t next_oid_max_len)
- u8_t snmp_next_oid_precheck (struct snmp_next_oid_state *state, const u32_t *oid, u8_t oid_len)
- u8_t snmp_next_oid_check (struct snmp_next_oid_state *state, const u32_t *oid, u8_t oid_len, void *reference)
- u8_t snmp_oid_in_range (const u32_t *oid_in, u8_t oid_len, const struct snmp_oid_range *oid_ranges, u8_t oid_ranges_len)
- err_t snmp_decode_bits (const u8_t *buf, u32_t buf_len, u32_t *bit_value)
- u8_t snmp_encode_bits (u8_t *buf, u32_t buf_len, u32_t bit_value, u8_t bit_count)

4.26.2 Detailed Description

MIB tree access/construction functions.

4.26.3 Function Documentation

4.26.3.1 netif_to_num()

```
u8_t netif_to_num (const struct netif * netif)
```

Convert netif to interface index

Parameters

netif	netif

Returns index

4.26.3.2 snmp_decode_bits()

```
err_t snmp_decode_bits (const u8_t * buf, u32_t buf_len, u32_t * bit_value)
```

Decodes BITS pseudotype value from ASN.1 OctetString.

Note
Because BITS pseudo type is encoded as OCTET STRING, it cannot directly be encoded/decoded by the agent. Instead call this function as required from get/test/set methods.

Parameters

buf	points to a buffer holding the ASN1 octet string
buf_len	length of octet string
bit_value	decoded Bit value with Bit0 == LSB

Returns ERR_OK if successful, ERR_ARG if bit value contains more than 32 bit

4.26.3.3 snmp_encode_bits()

```
u8_t snmp_encode_bits (u8_t * buf, u32_t buf_len, u32_t bit_value, u8_t bit_count)
```

Encodes BITS pseudotype value into ASN.1 OctetString.

Note
Because BITS pseudo type is encoded as OCTET STRING, it cannot directly be encoded/decoded by the agent. Instead call this function as required from get/test/set methods.

Parameters

buf	points to a buffer where the resulting ASN1 octet string is stored to
buf_len	max length of the buffer
bit_value	Bit value to encode with Bit0 == LSB
bit_count	Number of possible bits for the bit value (according to rfc we have to send all bits independent from their truth value)

Returns number of bytes used from buffer to store the resulting OctetString

4.26.3.4 snmp_ip4_to_oid()

```
void snmp_ip4_to_oid (const ip4_addr_t * ip, u32_t * oid)
```

Convert ip4_addr to InetAddressIPv4 (no InetAddressType)

Parameters

ip	points to input struct
oid	points to u32_t ident[4] output

4.26.3.5 snmp_ip6_to_oid()

```
void snmp_ip6_to_oid (const ip6_addr_t * ip, u32_t * oid)
```

Convert ip6_addr to InetAddressIPv6 (no InetAddressType)

Parameters

ip	points to input struct
oid	points to u32_t ident[16] output

4.26.3.6 snmp_ip_port_to_oid()

```
u8_t snmp_ip_port_to_oid (const ip_addr_t * ip, u16_t port, u32_t * oid)
```

Convert to InetAddressType+InetAddress+InetPortNumber

Parameters

ip	IP address
port	Port
oid	OID

Returns OID length

4.26.3.7 snmp_ip_to_oid()

```
u8_t snmp_ip_to_oid (const ip_addr_t * ip, u32_t * oid)
```

Convert to InetAddressType+InetAddress

Parameters

ip	IP address
oid	OID

Returns OID length

4.26.3.8 snmp_mib_tree_resolve_exact()

```
const struct snmp_node * snmp_mib_tree_resolve_exact (const struct snmp_mib * mib, const
u32_t * oid, u8_t oid_len, u8_t * oid_instance_len)
```

Searches tree for the supplied object identifier.

4.26.3.9 snmp_next_oid_check()

```
u8_t snmp_next_oid_check (struct snmp_next_oid_state * state, const u32_t * oid, u8_t oid_
void * reference)
```

checks the passed OID if it is a candidate to be the next one (get_next); returns !=0 if passed oid is currently closest, otherwise 0

4.26.3.10 snmp_next_oid_init()

`void snmp_next_oid_init (struct snmp_next_oid_state * state, const u32_t * start_oid, u8_t start_oid_len, u32_t * next_oid_buf, u8_t next_oid_max_len)`

initialize struct next_oid_state using this function before passing it to next_oid_check

4.26.3.11 snmp_next_oid_precheck()

`u8_t snmp_next_oid_precheck (struct snmp_next_oid_state * state, const u32_t * oid, u8_t oid_len)`

checks if the passed incomplete OID may be a possible candidate for snmp_next_oid_check(); this method is intended if the complete OID is not yet known but it is very expensive to build it up, so it is possible to test the starting part before building up the complete oid and pass it to snmp_next_oid_check()

4.26.3.12 snmp_oid_append()

`void snmp_oid_append (struct snmp_obj_id * target, const u32_t * oid, u8_t oid_len)`

Append OIDs to struct snmp_obj_id **Parameters**

target	Assignment target to append to
oid	OID
oid_len	OID length

4.26.3.13 snmp_oid_assign()

`void snmp_oid_assign (struct snmp_obj_id * target, const u32_t * oid, u8_t oid_len)`

Assign an OID to struct snmp_obj_id **Parameters**

target	Assignment target
oid	OID
oid_len	OID length

4.26.3.14 snmp_oid_combine()

`void snmp_oid_combine (struct snmp_obj_id * target, const u32_t * oid1, u8_t oid1_len, const u32_t * oid2, u8_t oid2_len)`

Combine two OIDs into struct snmp_obj_id **Parameters**

target	Assignment target
oid1	OID 1
oid1_len	OID 1 length
oid2	OID 2
oid2_len	OID 2 length

4.26.3.15 snmp_oid_compare()

```
s8_t snmp_oid_compare (const u32_t * oid1, u8_t oid1_len, const u32_t * oid2, u8_t oid2_le
```

Compare two OIDs

Parameters

oid1	OID 1
oid1_len	OID 1 length
oid2	OID 2
oid2_len	OID 2 length

Returns -1: OID1<OID2 1: OID1 >OID2 0: equal

4.26.3.16 snmp_oid_equal()

```
u8_t snmp_oid_equal (const u32_t * oid1, u8_t oid1_len, const u32_t * oid2, u8_t oid2_len)
```

Check of two OIDs are equal

Parameters

oid1	OID 1
oid1_len	OID 1 length
oid2	OID 2
oid2_len	OID 2 length

Returns 1: equal 0: non-equal

4.26.3.17 snmp_oid_in_range()

```
u8_t snmp_oid_in_range (const u32_t * oid_in, u8_t oid_len, const struct snmp_oid_range
* oid_ranges, u8_t oid_ranges_len)
```

checks if incoming OID length and values are in allowed ranges

4.26.3.18 snmp_oid_prefix()

```
void snmp_oid_prefix (struct snmp_obj_id * target, const u32_t * oid, u8_t oid_len)
```

Prefix an OID to OID in struct snmp_obj_id **Parameters**

target	Assignment target to prefix
oid	OID
oid_len	OID length

4.26.3.19 snmp_oid_to_ip()

```
u8_t snmp_oid_to_ip (const u32_t * oid, u8_t oid_len, ip_addr_t * ip)
```

Convert from InetAddressType+InetAddress to ip_addr_t

Parameters

Returns Parsed OID length

oid	OID
oid_len	OID length
ip	IP address

4.26.3.20 snmp_oid_to_ip4()

`u8_t snmp_oid_to_ip4 (const u32_t * oid, ip4_addr_t * ip)`

Conversion from InetAddressIPv4 oid to lwIP ip4_addr **Parameters**

oid	points to u32_t ident[4] input
ip	points to output struct

4.26.3.21 snmp_oid_to_ip6()

`u8_t snmp_oid_to_ip6 (const u32_t * oid, ip6_addr_t * ip)`

Conversion from InetAddressIPv6 oid to lwIP ip6_addr **Parameters**

oid	points to u32_t oid[16] input
ip	points to output struct

4.26.3.22 snmp_oid_to_ip_port()

`u8_t snmp_oid_to_ip_port (const u32_t * oid, u8_t oid_len, ip_addr_t * ip, u16_t * port)`

Convert from InetAddressType+InetAddress+InetPortNumber to ip_addr_t and u16_t

Parameters

Returns Parsed OID length

4.27 src/apps/snmp/snmp_mib2.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/snmp.h"#include "lwip/apps/snmp.h"#include " ↩
    lwip/apps/snmp_core.h"#include "lwip/apps/snmp_mib2.h"#include "lwip/apps/snmp_scalar.h ↩
    "#include "lwip/tcpip.h"#include "lwip/priv/tcpip_priv.h"
```

4.27.1 Detailed Description

Management Information Base II (RFC1213) objects and functions.

4.28 src/apps/snmp/snmp_mib2_icmp.c File Reference

```
#include "lwip/snmp.h"#include "lwip/apps/snmp.h"#include "lwip/apps/snmp_core.h"#include " ↩
    lwip/apps/snmp_mib2.h"#include "lwip/apps/snmp_table.h"#include "lwip/apps/snmp_scalar.h ↩
    "#include "lwip/icmp.h"#include "lwip/stats.h"
```

oid	OID
oid_len	OID length
ip	IP address
port	Port

4.28.1 Detailed Description

Management Information Base II (RFC1213) ICMP objects and functions.

4.29 src/apps/snmp/snmp_mib2_interfaces.c File Reference

```
#include "lwip/snmp.h"#include "lwip/apps/snmp.h"#include "lwip/apps/snmp_core.h"#include " ↩
    lwip/apps/snmp_mib2.h"#include "lwip/apps/snmp_table.h"#include "lwip/apps/snmp_scalar.h ↩
    "#include "lwip/netif.h"#include "lwip/stats.h"#include <string.h>
```

4.29.1 Detailed Description

Management Information Base II (RFC1213) INTERFACES objects and functions.

4.30 src/apps/snmp/snmp_mib2_ip.c File Reference

```
#include "lwip/snmp.h"#include "lwip/apps/snmp.h"#include "lwip/apps/snmp_core.h"#include " ↩
    lwip/apps/snmp_mib2.h"#include "lwip/apps/snmp_table.h"#include "lwip/apps/snmp_scalar.h ↩
    "#include "lwip/stats.h"#include "lwip/netif.h"#include "lwip/ip.h"#include "lwip/etharp ↩
    .h"
```

4.30.1 Detailed Description

Management Information Base II (RFC1213) IP objects and functions.

4.31 src/apps/snmp/snmp_mib2_snmp.c File Reference

```
#include "lwip/snmp.h"#include "lwip/apps/snmp.h"#include "lwip/apps/snmp_core.h"#include " ↩
    lwip/apps/snmp_mib2.h"#include "lwip/apps/snmp_scalar.h"
```

4.31.1 Detailed Description

Management Information Base II (RFC1213) SNMP objects and functions.

4.32 src/apps/snmp/snmp_mib2_system.c File Reference

```
#include "lwip/snmp.h"#include "lwip/apps/snmp.h"#include "lwip/apps/snmp_core.h"#include " ↩
    lwip/apps/snmp_mib2.h"#include "lwip/apps/snmp_table.h"#include "lwip/apps/snmp_scalar.h ↩
    "#include "lwip/sys.h"#include <string.h>
```

4.32.1 Functions

- void snmp_mib2_set_sysdescr (const u8_t *str, const u16_t *len)
- void snmp_mib2_set_syscontact (u8_t *ocstr, u16_t *ocstrlen, u16_t bufsize)
- void snmp_mib2_set_syscontact_readonly (const u8_t *ocstr, const u16_t *ocstrlen)
- void snmp_mib2_set_sysname (u8_t *ocstr, u16_t *ocstrlen, u16_t bufsize)
- void snmp_mib2_set_sysname_readonly (const u8_t *ocstr, const u16_t *ocstrlen)
- void snmp_mib2_set_syslocation (u8_t *ocstr, u16_t *ocstrlen, u16_t bufsize)
- void snmp_mib2_set_syslocation_readonly (const u8_t *ocstr, const u16_t *ocstrlen)

4.32.2 Detailed Description

Management Information Base II (RFC1213) SYSTEM objects and functions.

4.33 src/apps/snmp/snmp_mib2_tcp.c File Reference

```
#include "lwip/snmp.h"#include "lwip/apps/snmp.h"#include "lwip/apps/snmp_core.h"#include " ↩
    lwip/apps/snmp_mib2.h"#include "lwip/apps/snmp_table.h"#include "lwip/apps/snmp_scalar.h ↩
    "#include "lwip/tcp.h"#include "lwip/priv/tcp_priv.h"#include "lwip/stats.h"#include < ↩
    string.h>
```

4.33.1 Detailed Description

Management Information Base II (RFC1213) TCP objects and functions.

4.34 src/apps/snmp/snmp_mib2_udp.c File Reference

```
#include "lwip/snmp.h"#include "lwip/apps/snmp.h"#include "lwip/apps/snmp_core.h"#include " ↩
    lwip/apps/snmp_mib2.h"#include "lwip/apps/snmp_table.h"#include "lwip/apps/snmp_scalar.h ↩
    "#include "lwip/udp.h"#include "lwip/stats.h"#include <string.h>
```

4.34.1 Detailed Description

Management Information Base II (RFC1213) UDP objects and functions.

4.35 src/apps/snmp/snmp_msg.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include "snmp_msg.h"#include "snmp_asn1.h"#include " ↩
    snmp_core_priv.h"#include "lwip/ip_addr.h"#include "lwip/stats.h"#include <string.h>
```

4.35.1 Functions

- const char * snmp_get_community (void)
- void snmp_set_community (const char *const community)
- const char * snmp_get_community_write (void)
- const char * snmp_get_community_trap (void)
- void snmp_set_community_write (const char *const community)
- void snmp_set_community_trap (const char *const community)
- void snmp_set_write_callback (snmp_write_callback_fct write_callback, void *callback_arg)
- void snmp_set_inform_callback (snmp_inform_callback_fct inform_callback, void *callback_arg)
- err_t snmp_varbind_length (struct snmp_varbind *varbind, struct snmp_varbind_len *len)

4.35.2 Variables

- const char * snmp_community = "public"
- const char * snmp_community_write = "private"
- const char * snmp_community_trap = "public"

4.35.3 Detailed Description

SNMP message processing (RFC1157).

4.35.4 Function Documentation

4.35.4.1 snmp_varbind_length()

```
err_t snmp_varbind_length (struct snmp_varbind * varbind, struct snmp_varbind_len * len)
```

Calculate the length of a varbind list

4.35.5 Variable Documentation

4.35.5.1 snmp_community

```
const char* snmp_community = "public"
```

SNMP community string

4.35.5.2 snmp_community_trap

```
const char* snmp_community_trap = "public"
```

SNMP community string for sending traps

4.35.5.3 snmp_community_write

```
const char* snmp_community_write = "private"
```

SNMP community string for write access

4.36 src/apps/snmp/snmp_msg.h File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/apps/snmp.h"#include "lwip/apps/snmp_core.h ↩
    "#include "snmp_pbuf_stream.h"#include "lwip/ip_addr.h"#include "lwip/err.h"
```

4.36.1 Data Structures

- struct snmp_varbind_len

4.36.2 Functions

- err_t snmp_varbind_length (struct snmp_varbind *varbind, struct snmp_varbind_len *len)

4.36.3 Variables

- const char * snmp_community
- const char * snmp_community_write
- void * snmp_traps_handle

4.36.4 Detailed Description

SNMP Agent message handling structures (internal API, do not use in client code).

4.36.5 Function Documentation

4.36.5.1 snmp_varbind_length()

`err_t snmp_varbind_length (struct snmp_varbind * varbind, struct snmp_varbind_len * len)`

Calculate the length of a varbind list

4.36.6 Variable Documentation

4.36.6.1 snmp_community

`const char* snmp_community[extern]`

Agent community string

SNMP community string

4.36.6.2 snmp_community_write

`const char* snmp_community_write[extern]`

Agent community string for write access

SNMP community string for write access

4.36.6.3 snmp_traps_handle

```
void* snmp_traps_handle[extern]
```

handle for sending traps

4.37 src/apps/snmp/snmp_netconn.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include <string.h>#include "lwip/api.h"#include "lwip/ip.h ↩
    "#include "lwip/udp.h"#include "snmp_msg.h"#include "lwip/sys.h"#include "lwip/prot/iana ↩
    .h"
```

4.37.1 Functions

- void snmp_init (void)

4.37.2 Detailed Description

SNMP netconn frontend.

4.37.3 Function Documentation

4.37.3.1 snmp_init()

```
void snmp_init (void )
```

Starts SNMP Agent.

4.38 src/apps/snmp/snmp_pbuf_stream.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include "snmp_pbuf_stream.h"#include "lwip/def.h"#include  ↩
    <string.h>
```

4.38.1 Detailed Description

SNMP pbuf stream wrapper implementation (internal API, do not use in client code).

4.39 src/apps/snmp/snmp_pbuf_stream.h File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/err.h"#include "lwip/pbuf.h"
```

4.39.1 Detailed Description

SNMP pbuf stream wrapper (internal API, do not use in client code).

4.40 src/apps/snmp/snmp_raw.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/ip_addr.h"#include "lwip/udp.h"#include " ↩
    lwip/ip.h"#include "lwip/prot/iana.h"#include "snmp_msg.h"
```

4.40.1 Functions

- void snmp_init (void)

4.40.2 Detailed Description

SNMP RAW API frontend.

4.41 src/apps/snmp/snmp_scalar.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/apps/snmp_scalar.h"#include "lwip/apps/ ↩
    snmp_core.h"
```

4.41.1 Detailed Description

SNMP scalar node support implementation.

4.42 src/apps/snmp/snmp_table.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/apps/snmp_core.h"#include "lwip/apps/ ↩
    snmp_table.h"#include <string.h>
```

4.42.1 Detailed Description

SNMP table support implementation.

4.43 src/apps/snmp/snmp_threadsync.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/apps/snmp_threadsync.h"#include "lwip/apps/ ↩
    snmp_core.h"#include "lwip/sys.h"#include <string.h>
```

4.43.1 Functions

- void snmp_threadsync_init (struct snmp_threadsync_instance *instance, snmp_threadsync_synchronizer_fn sync_fn)

4.43.2 Detailed Description

SNMP thread synchronization implementation.

4.43.3 Function Documentation

4.43.3.1 snmp_threadsync_init()

```
void snmp_threadsync_init (struct snmp_threadsync_instance * instance, snmp_threadsync_syn
sync_fn)
```

Initializes thread synchronization instance

4.44 src/apps/snmp/snmp_traps.c File Reference

```
#include "lwip/apps/snmp_opts.h"#include <string.h>#include "lwip/snmp.h"#include "lwip/sys ↩
    .h"#include "lwip/apps/snmp.h"#include "lwip/apps/snmp_core.h"#include "lwip/prot/iana.h ↩
    "#include "snmp_msg.h"#include "snmp_asn1.h"#include "snmp_core_priv.h"
```

4.44.1 Functions

- void snmp_trap_dst_enable (u8_t dst_idx, u8_t enable)
- void snmp_trap_dst_ip_set (u8_t dst_idx, const ip_addr_t *dst)
- void snmp_set_auth_traps_enabled (u8_t enable)
- u8_t snmp_get_auth_traps_enabled (void)
- void snmp_set_default_trap_version (u8_t snmp_version)
- u8_t snmp_get_default_trap_version (void)
- err_t snmp_send_trap (const struct snmp_obj_id *oid, s32_t generic_trap, s32_t specific_trap, struct snmp_varbind *varbinds)
- err_t snmp_send_trap_generic (s32_t generic_trap)
- err_t snmp_send_trap_specific (s32_t specific_trap, struct snmp_varbind *varbinds)
- void snmp_coldstart_trap (void)
- void snmp_authfail_trap (void)
- err_t snmp_send_inform_specific (s32_t specific_trap, struct snmp_varbind *varbinds, s32_t *ptr_request_id)
- err_t snmp_send_inform_generic (s32_t generic_trap, struct snmp_varbind *varbinds, s32_t *ptr_request_id)
- err_t snmp_send_inform (const struct snmp_obj_id *oid, s32_t generic_trap, s32_t specific_trap, struct snmp_varbind *varbinds, s32_t *ptr_request_id)

4.44.2 Variables

- const char * snmp_community_trap
- void * snmp_traps_handle

4.44.3 Detailed Description

SNMPv1 and SNMPv2 traps implementation.

4.44.4 Variable Documentation

4.44.4.1 snmp_community_trap

`const char* snmp_community_trap[extern]`

Agent community string for sending traps

SNMP community string for sending traps

4.44.4.2 snmp_traps_handle

`void* snmp_traps_handle`

handle for sending traps

4.45 src/apps/snmp/snmpv3.c File Reference

```
#include "snmpv3_priv.h"#include "lwip/apps/snmpv3.h"#include "lwip/sys.h"#include <string. ↩
    h>
```

4.45.1 Detailed Description

Additional SNMPv3 functionality RFC3414 and RFC3826.

4.46 src/apps/snmp/snmpv3_mbedtls.c File Reference

```
#include "lwip/apps/snmpv3.h"#include "snmpv3_priv.h"#include "lwip/arch.h"#include " ↩
    snmp_msg.h"#include "lwip/sys.h"#include <string.h>
```

4.46.1 Detailed Description

SNMPv3 crypto/auth functions implemented for ARM mbedtls.

4.47 src/apps/snmp/snmpv3_priv.h File Reference

```
#include "lwip/apps/snmp_opts.h"
```

4.47.1 Detailed Description

Additional SNMPv3 functionality RFC3414 and RFC3826 (internal API, do not use in client code).

4.48 src/apps/sntp/sntp.c File Reference

```
#include "lwip/apps/sntp.h"#include "lwip/opt.h"#include "lwip/timeouts.h"#include "lwip/ ↩
    udp.h"#include "lwip/dns.h"#include "lwip/ip_addr.h"#include "lwip/pbuf.h"#include "lwip ↩
    /dhcp.h"#include <string.h>#include <time.h>#include "arch/bpstruct.h"#include "arch/ ↩
    epstruct.h"
```

4.48.1 Data Structures

- struct sntp_time
- struct sntp_timestamps
- struct sntp_msg
- struct sntp_server

4.48.2 Macros

- #define SNTP_FRAC_TO_US(f)   ((u32_t)(f) / 4295)

4.48.3 Functions

- void sntp_init (void)
- void sntp_stop (void)
- u8_t sntp_enabled (void)
- void sntp_setoperatingmode (u8_t operating_mode)
- u8_t sntp_getoperatingmode (void)
- u8_t sntp_getreachability (u8_t idx)
- void sntp_setserver (u8_t idx, const ip_addr_t *server)
- const ip_addr_t * sntp_getserver (u8_t idx)
- u8_t sntp_getkodreceived (u8_t idx)

4.48.4 Detailed Description

SNTP client module

4.48.5 Macro Definition Documentation

4.48.5.1 SNTP_FRAC_TO_US

```
#define SNTP_FRAC_TO_US( f)    ((u32_t)(f) / 4295)
```

Convert NTP timestamp fraction to microseconds.

4.49 src/apps/tftp/tftp.c File Reference

Trivial File Transfer Protocol (RFC 1350)

```
#include "lwip/apps/tftp_client.h"#include "lwip/apps/tftp_server.h"#include "lwip/udp.h"# ↩
    include "lwip/timeouts.h"#include "lwip/debug.h"#include <string.h>
```

4.49.1 Functions

- err_t tftp_init_common (u8_t mode, const struct tftp_context *ctx)
- err_t tftp_init_server (const struct tftp_context *ctx)
- err_t tftp_init_client (const struct tftp_context *ctx)
- void tftp_cleanup (void)

4.49.2 Detailed Description

Trivial File Transfer Protocol (RFC 1350)

Author Logan Gunthorpe logang@deltatee.com Dirk Ziegelmeier dziegel@gmx.de

Copyright (c) Deltatee Enterprises Ltd. 2013 All rights reserved.

4.49.3 Function Documentation

4.49.3.1 tftp_init_common()

```
err_t tftp_init_common (u8_t mode, const struct tftp_context * ctx)
```

Initialize TFTP client/server.

Parameters

mode	TFTP mode (client/server)
ctx	TFTP callback struct

4.50 src/core/altcp.c File Reference

```
#include "lwip/opt.h"#include "lwip/altcp.h"#include "lwip/priv/altcp_priv.h"#include "lwip ↩
    /altcp_tcp.h"#include "lwip/tcp.h"#include "lwip/mem.h"#include <string.h>
```

4.50.1 Functions

- struct altcp_pcb * altcp_alloc (void)
- void altcp_free (struct altcp_pcb *conn)
- struct altcp_pcb * altcp_new_ip6 (altcp_allocator_t *allocator)
- struct altcp_pcb * altcp_new (altcp_allocator_t *allocator)
- struct altcp_pcb * altcp_new_ip_type (altcp_allocator_t *allocator, u8_t ip_type)
- void altcp_arg (struct altcp_pcb *conn, void *arg)
- void altcp_accept (struct altcp_pcb *conn, altcp_accept_fn accept)
- void altcp_recv (struct altcp_pcb *conn, altcp_recv_fn recv)
- void altcp_sent (struct altcp_pcb *conn, altcp_sent_fn sent)

- void altcp_poll (struct altcp_pcb *conn, altcp_poll_fn poll, u8_t interval)
- void altcp_err (struct altcp_pcb *conn, altcp_err_fn err)
- void altcp_recved (struct altcp_pcb *conn, u16_t len)
- err_t altcp_bind (struct altcp_pcb *conn, const ip_addr_t *ipaddr, u16_t port)
- err_t altcp_connect (struct altcp_pcb *conn, const ip_addr_t *ipaddr, u16_t port, altcp_connected_fn connected)
- struct altcp_pcb * altcp_listen_with_backlog_and_err (struct altcp_pcb *conn, u8_t backlog, err_t *err)
- void altcp_abort (struct altcp_pcb *conn)
- err_t altcp_close (struct altcp_pcb *conn)
- err_t altcp_shutdown (struct altcp_pcb *conn, int shut_rx, int shut_tx)
- err_t altcp_write (struct altcp_pcb *conn, const void *dataptr, u16_t len, u8_t apiflags)
- err_t altcp_output (struct altcp_pcb *conn)
- u16_t altcp_mss (struct altcp_pcb *conn)
- u16_t altcp_sndbuf (struct altcp_pcb *conn)
- u16_t altcp_sndqueuelen (struct altcp_pcb *conn)
- void altcp_setprio (struct altcp_pcb *conn, u8_t prio)

4.50.2 Function Documentation

4.50.2.1 altcp_alloc()

```
struct altcp_pcb * altcp_alloc (void )
```

For altcp layer implementations only: allocate a new struct altcp_pcb from the pool and zero the memory

4.50.2.2 altcp_free()

```
void altcp_free (struct altcp_pcb * conn)
```

For altcp layer implementations only: return a struct altcp_pcb to the pool

4.51 src/core/altcp_alloc.c File Reference

```
#include "lwip/opt.h"#include "lwip/altcp.h"#include "lwip/altcp_tcp.h"#include "lwip/ ↩
    altcp_tls.h"#include "lwip/priv/altcp_priv.h"#include "lwip/mem.h"#include <string.h>
```

4.51.1 Functions

- struct altcp_pcb * altcp_tls_new (struct altcp_tls_config *config, u8_t ip_type)
- struct altcp_pcb * altcp_tls_alloc (void *arg, u8_t ip_type)

4.51.2 Detailed Description

Application layered TCP connection API (to be used from TCPIP thread) This interface mimics the tcp callback API to the application while preventing direct linking (much like virtual functions). This way, an application can make use of other application layer protocols on top of TCP without knowing the details (e.g. TLS, proxy connection).

This file contains allocation implementation that combine several layers.

4.52 src/core/altcp_tcp.c File Reference

```
#include "lwip/opt.h"#include "lwip/altcp.h"#include "lwip/altcp_tcp.h"#include "lwip/priv/ ↩
    altcp_priv.h"#include "lwip/tcp.h"#include "lwip/priv/tcp_priv.h"#include "lwip/mem.h"# ↩
    include <string.h>
```

4.52.1 Functions

- struct altcp_pcb * altcp_tcp_alloc (void *arg, u8_t ip_type)

4.52.2 Detailed Description

Application layered TCP connection API (to be used from TCPIP thread)

This interface mimics the tcp callback API to the application while preventing direct linking (much like virtual functions). This way, an application can make use of other application layer protocols on top of TCP without knowing the details (e.g. TLS, proxy connection).

This file contains the base implementation calling into tcp.

4.52.3 Function Documentation

4.52.3.1 altcp_tcp_alloc()

```
struct altcp_pcb * altcp_tcp_alloc (void * arg, u8_t ip_type)
```

altcp_tcp allocator function fitting to altcp_allocator_t / altcp_new.

arg pointer is not used for TCP.

4.53 src/core/def.c File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include <string.h>
```

4.53.1 Functions

- u16_t lwip_htons (u16_t n)
- u32_t lwip_htonl (u32_t n)
- char * lwip_strnstr (const char *buffer, const char *token, size_t n)
- char * lwip_strnistr (const char *buffer, const char *token, size_t n)
- int lwip_stricmp (const char *str1, const char *str2)

- int lwip_strnicmp (const char *str1, const char *str2, size_t len)
- void lwip_itoa (char *result, size_t bufsize, int number)
- int lwip_memcmp_consttime (const void *s1, const void *s2, size_t len)

4.53.2 Detailed Description

Common functions used throughout the stack.

These are reference implementations of the byte swapping functions. Again with the aim of being simple, correct and fully portable. Byte swapping is the second thing you would want to optimize. You will need to port it to your architecture and in your cc.h:

#define lwip_htons(x) your_htons #define lwip_htonl(x) your_htonl

Note lwip_ntohs() and lwip_ntohl() are merely references to the htonx counterparts.

If you #define them to htons() and htonl(), you should #define LWIP_DONT_PROVIDE_BYTEORDER_FUNCTIONS to prevent lwIP from defining htonx/ntohx compatibility macros.

4.53.3 Function Documentation

4.53.3.1 lwip_htonl()

```
u32_t lwip_htonl (u32_t n)
```

Convert an u32_t from host- to network byte order.

Parameters

n	u32_t in host byte order

Returns n in network byte order

4.53.3.2 lwip_htons()

```
u16_t lwip_htons (u16_t n)
```

Convert an u16_t from host- to network byte order.

Parameters

n	u16_t in host byte order

Returns n in network byte order

4.54 src/core/dns.c File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/udp.h"#include "lwip/mem.h"# ↩
    include "lwip/memp.h"#include "lwip/dns.h"#include "lwip/prot/dns.h"#include <string.h>
```

4.54.1 Data Structures

- struct dns_query
- struct dns_answer
- struct dns_table_entry
- struct dns_req_entry

4.54.2 Macros

- #define DNS_RAND_TXID LWIP_RAND
- #define DNS_PORT_ALLOWED(port) ((port) >= 1024)
- #define DNS_MAX_TTL 604800

4.54.3 Functions

- void dns_init (void)
- void dns_setserver (u8_t numdns, const ip_addr_t *dnsserver)
- const ip_addr_t * dns_getserver (u8_t numdns)
- void dns_tmr (void)
- err_t dns_gethostbyname (const char *hostname, ip_addr_t *addr, dns_found_callback found, void *callback_arg)
- err_t dns_gethostbyname_addrtype (const char *hostname, ip_addr_t *addr, dns_found_callback found, void *callback_arg, u8_t dns_addrtype)

4.54.4 Detailed Description

DNS - host name to IP address resolver.

4.54.5 Macro Definition Documentation

4.54.5.1 DNS_MAX_TTL

```
#define DNS_MAX_TTL   604800
```

DNS resource record max. TTL (one week as default)

4.54.5.2 DNS_PORT_ALLOWED

```
#define DNS_PORT_ALLOWED( port)   ((port) >= 1024)
```

Limits the source port to be >= 1024 by default

4.54.5.3 DNS_RAND_TXID

```
#define DNS_RAND_TXID   LWIP_RAND
```

Random generator function to create random TXIDs and source ports for queries

4.54.6 Function Documentation

4.54.6.1 dns_init()

```
void dns_init (void )
```

Initialize the resolver: set up the UDP pcb and configure the default server (if DNS_SERVER_ADDRESS is set).

4.54.6.2 dns_tmr()

```
void dns_tmr (void )
```

The DNS resolver client timer - handle retries and timeouts and should be called every DNS_TMR_INTERVAL milliseconds (every second by default).

4.55 src/core/inet_chksum.c File Reference

```
#include "lwip/opt.h"#include "lwip/inet_chksum.h"#include "lwip/def.h"#include "lwip/ ↩
    ip_addr.h"#include <string.h>
```

4.55.1 Functions

- u16_t ip6_chksum_pseudo (struct pbuf *p, u8_t proto, u16_t proto_len, const ip6_addr_t *src, const ip6_addr_t *dest)
- u16_t ip6_chksum_pseudo_partial (struct pbuf *p, u8_t proto, u16_t proto_len, u16_t chksum_len, const ip6_addr_t *src, const ip6_addr_t *dest)
- u16_t inet_chksum_pbuf (struct pbuf *p)

4.55.2 Detailed Description

Internet checksum functions.

These are some reference implementations of the checksum algorithm, with the aim of being simple, correct and fully portable. Checksumming is the first thing you would want to optimize for your platform. If you create your own version, link it in and in your cc.h put:

#define LWIP_CHKSUM your_checksum_routine

Or you can select from the implementations below by defining LWIP_CHKSUM_ALGORITHM to 1, 2 or 3.

4.55.3 Function Documentation

4.55.3.1 inet_chksum_pbuf()

```
u16_t inet_chksum_pbuf (struct pbuf * p)
```

Calculate a checksum over a chain of pbufs (without pseudo-header, much like inet_chksum only pbufs are used).

Parameters

p	pbuf chain over that the checksum should be calculated

Returns checksum (as u16_t) to be saved directly in the protocol header

4.55.3.2 ip6_chksum_pseudo()

`u16_t ip6_chksum_pseudo (struct pbuf * p, u8_t proto, u16_t proto_len, const ip6_addr_t * src, const ip6_addr_t * dest)`

Calculates the checksum with IPv6 pseudo header used by TCP and UDP for a pbuf chain. IPv6 addresses are expected to be in network byte order.

Parameters

p	chain of pbufs over that a checksum should be calculated (ip data part)
proto	ipv6 protocol/next header (used for checksum of pseudo header)
proto_len	length of the ipv6 payload (used for checksum of pseudo header)
src	source ipv6 address (used for checksum of pseudo header)
dest	destination ipv6 address (used for checksum of pseudo header)

Returns checksum (as u16_t) to be saved directly in the protocol header

4.55.3.3 ip6_chksum_pseudo_partial()

`u16_t ip6_chksum_pseudo_partial (struct pbuf * p, u8_t proto, u16_t proto_len, u16_t chksu`
`const ip6_addr_t * src, const ip6_addr_t * dest)`

Calculates the checksum with IPv6 pseudo header used by TCP and UDP for a pbuf chain. IPv6 addresses are expected to be in network byte order. Will only compute for a portion of the payload.

Parameters

p	chain of pbufs over that a checksum should be calculated (ip data part)
proto	ipv6 protocol/next header (used for checksum of pseudo header)
proto_len	length of the ipv6 payload (used for checksum of pseudo header)
chksum_len	number of payload bytes used to compute chksum
src	source ipv6 address (used for checksum of pseudo header)
dest	destination ipv6 address (used for checksum of pseudo header)

Returns checksum (as u16_t) to be saved directly in the protocol header

4.56 src/core/init.c File Reference

```
#include "lwip/opt.h"#include "lwip/init.h"#include "lwip/stats.h"#include "lwip/sys.h"# ↩
    include "lwip/mem.h"#include "lwip/memp.h"#include "lwip/pbuf.h"#include "lwip/netif.h"# ↩
    include "lwip/sockets.h"#include "lwip/ip.h"#include "lwip/raw.h"#include "lwip/udp.h"# ↩
    include "lwip/priv/tcp_priv.h"#include "lwip/igmp.h"#include "lwip/dns.h"#include "lwip/ ↩
    timeouts.h"#include "lwip/etharp.h"#include "lwip/ip6.h"#include "lwip/nd6.h"#include " ↩
    lwip/mld6.h"#include "lwip/api.h"#include "netif/ppp/ppp_opts.h"#include "netif/ppp/ ↩
    ppp_impl.h"#include "arch/bpstruct.h"#include "arch/epstruct.h"
```

4.56.1 Functions

- void lwip_init (void)

4.56.2 Detailed Description

Modules initialization

4.57 src/core/ip.c File Reference

```
#include "lwip/opt.h"#include "lwip/ip_addr.h"#include "lwip/ip.h"
```

4.57.1 Functions

- char * ipaddr_ntoa (const ip_addr_t *addr)
- char * ipaddr_ntoa_r (const ip_addr_t *addr, char *buf, int buflen)
- int ipaddr_aton (const char *cp, ip_addr_t *addr)
- err_t ip_input (struct pbuf *p, struct netif *inp)

4.57.2 Variables

- struct ip_globals ip_data

4.57.3 Detailed Description

Common IPv4 and IPv6 code

4.57.4 Variable Documentation

4.57.4.1 ip_data

`struct ip_globals ip_data`

Global data for both IPv4 and IPv6

4.58 src/core/ipv4/acd.c File Reference

```
#include "lwip/opt.h"#include <string.h>#include "lwip/acd.h"#include "lwip/prot/acd.h"
```

4.58.1 Functions

- err_t acd_add (struct netif *netif, struct acd *acd, acd_conflict_callback_t acd_conflict_callback)
- void acd_remove (struct netif *netif, struct acd *acd)
- err_t acd_start (struct netif *netif, struct acd *acd, ip4_addr_t ipaddr)
- err_t acd_stop (struct acd *acd)
- void acd_network_changed_link_down (struct netif *netif)
- void acd_tmr (void)
- void acd_arp_reply (struct netif *netif, struct etharp_hdr *hdr)
- void acd_netif_ip_addr_changed (struct netif *netif, const ip_addr_t *old_addr, const ip_addr_t *new_addr)

4.58.2 Detailed Description

ACD IPv4 Address Conflict Detection

This is an IPv4 address conflict detection implementation for the lwIP TCP/IP stack. It aims to be conform to RFC5227.

4.58.3 Function Documentation

4.58.3.1 acd_arp_reply()

```
void acd_arp_reply (struct netif * netif, struct etharp_hdr * hdr)
```

Handles every incoming ARP Packet, called by etharp_input().

Parameters

netif	network interface to use for acd processing
hdr	Incoming ARP packet

4.58.3.2 acd_tmr()

```
void acd_tmr (void )
```

Has to be called in loop every ACD_TMR_INTERVAL milliseconds

4.59 src/core/ipv4/autoip.c File Reference

```
#include "lwip/opt.h"#include "lwip/mem.h"#include "lwip/ip_addr.h"#include "lwip/netif.h"# ↩
    include "lwip/autoip.h"#include "lwip/acd.h"#include "lwip/etharp.h"#include "lwip/prot/ ↩
    autoip.h"#include <string.h>
```

4.59.1 Macros

- #define LWIP_AUTOIP_CREATE_SEED_ADDR(netif)

4.59.2 Functions

- void autoip_set_struct (struct netif *netif, struct autoip *autoip)
- void autoip_remove_struct (struct netif *netif)
- err_t autoip_start (struct netif *netif)
- void autoip_network_changed_link_up (struct netif *netif)
- void autoip_network_changed_link_down (struct netif *netif)
- err_t autoip_stop (struct netif *netif)
- u8_t autoip_supplied_address (struct netif *netif)

4.59.3 Detailed Description

AutoIP Automatic LinkLocal IP Configuration

This is a AutoIP implementation for the lwIP TCP/IP stack. It aims to conform with RFC 3927. It uses IPv4 address conflict detection to evaluate the chosen address. The ACD module aims to be conform to RFC 5227. RFC 5227 is extracted out of RFC 3927 so the acd module fits nicely in autoip.

4.59.4 Macro Definition Documentation

4.59.4.1 LWIP_AUTOIP_CREATE_SEED_ADDR

`#define LWIP_AUTOIP_CREATE_SEED_ADDR( netif)` **Value:**

```
lwip_htonl(AUTOIP_RANGE_START + ((u32_t)(((u8_t)(netif->hwaddr[4])) | \
                 ((u32_t)((u8_t)(netif->hwaddr[5]))) << 8)))
```

Macro that generates the initial IP address to be tried by AUTOIP. If you want to override this, define it to something else in lwipopts.h.

4.59.5 Function Documentation

4.59.5.1 autoip_network_changed_link_down()

`void autoip_network_changed_link_down (struct netif * netif)`

Handle a possible change in the network configuration: link down

If there is an AutoIP address configured and AutoIP is in cooperation with DHCP, then stop the autoip module. When the link goes up, we do not want the autoip module to start again. DHCP will initiate autoip when needed.

4.59.5.2 autoip_network_changed_link_up()

`void autoip_network_changed_link_up (struct netif * netif)`

Handle a possible change in the network configuration: link up

If there is an AutoIP address configured and AutoIP is not in cooperation with DHCP, start probing for previous address.

4.59.5.3 autoip_supplied_address()

`u8_t autoip_supplied_address (struct netif * netif)`

check if AutoIP supplied netif->ip_addr

Parameters

netif	the netif to check

Returns 1 if AutoIP supplied netif->ip_addr (state BOUND), 0 otherwise

4.60 src/core/ipv4/dhcp.c File Reference

```
#include "lwip/opt.h"#include "lwip/stats.h"#include "lwip/mem.h"#include "lwip/udp.h"# ↩
    include "lwip/ip_addr.h"#include "lwip/netif.h"#include "lwip/def.h"#include "lwip/dhcp. ↩
    h"#include "lwip/autoip.h"#include "lwip/acd.h"#include "lwip/dns.h"#include "lwip/ ↩
    etharp.h"#include "lwip/prot/dhcp.h"#include "lwip/prot/iana.h"#include <string.h># ↩
    include "path/to/my/lwip_hooks.h"
```

4.60.1 Macros

- #define DHCP_ADD_EXTRA_REQUEST_OPTIONS
- #define SET_TIMEOUT_FROM_OFFERED(result, offered, min, max)
- #define DHCP_CREATE_RAND_XID 1
- #define DHCP_MAX_MSG_LEN(netif) (netif->mtu)
- #define DHCP_MIN_REPLY_LEN 44

4.60.2 Enumerations

- enum dhcp_option_idx

4.60.3 Functions

- void dhcp_coarse_tmr (void)
- void dhcp_fine_tmr (void)
- void dhcp_set_struct (struct netif *netif, struct dhcp *dhcp)
- void dhcp_cleanup (struct netif *netif)
- err_t dhcp_start (struct netif *netif)
- void dhcp_inform (struct netif *netif)
- void dhcp_network_changed_link_up (struct netif *netif)
- err_t dhcp_renew (struct netif *netif)
- void dhcp_release_and_stop (struct netif *netif)
- err_t dhcp_release (struct netif *netif)
- void dhcp_stop (struct netif *netif)
- u8_t dhcp_supplied_address (const struct netif *netif)

4.60.4 Detailed Description

Dynamic Host Configuration Protocol client

4.60.5 Macro Definition Documentation

4.60.5.1 DHCP_ADD_EXTRA_REQUEST_OPTIONS

```
#define DHCP_ADD_EXTRA_REQUEST_OPTIONS
```

DHCP_ADD_EXTRA_REQUEST_OPTIONS: Additional options added to the list of options that the client requests from the servers (opt 55: DHCP_OPTION_PARAMETER_REQUEST_LIST) If additional options are requested, define this macro as a comma separated list, with leading comma. This macro is useful for example when requested vendor specific ids (VCI/VSI options), here is an example of requesting the VSI option (option 43) (yes, the notation is a bit strange, but it works :) (NOTE: the space between # and define is required because of doxygen...) **define DHCP_ADD_EXTRA_REQUEST_OPTIONS ,43**

4.60.5.2 DHCP_CREATE_RAND_XID

```
#define DHCP_CREATE_RAND_XID   1
```

DHCP_CREATE_RAND_XID: if this is set to 1, the xid is created using LWIP_RAND() (this overrides DHCP_GLOBAL_XID)

4.60.5.3 DHCP_MAX_MSG_LEN

```
#define DHCP_MAX_MSG_LEN( netif)   (netif->mtu)
```

Default for DHCP_GLOBAL_XID is 0xABCD0000 This can be changed by defining DHCP_GLOBAL_XID and DHCP_GLOBAL_XI
e.g. #define DHCP_GLOBAL_XID_HEADER "stdlib.h" #define DHCP_GLOBAL_XID rand() DHCP_OPTION_MAX_MSG_SIZE is set to the MTU MTU is checked to be big enough in dhcp_start

4.60.5.4 DHCP_MIN_REPLY_LEN

```
#define DHCP_MIN_REPLY_LEN   44
```

Minimum length for reply before packet is parsed

4.60.5.5 SET_TIMEOUT_FROM_OFFERED

`#define SET_TIMEOUT_FROM_OFFERED( result, offered, min, max)` **Value:**

```
  do { \
  u32_t timeout = (offered + DHCP_COARSE_TIMER_SECS / 2) / DHCP_COARSE_TIMER_SECS;   \
  if (timeout > max) {     \
    timeout = max;         \
  }                        \
  if (timeout == min) {    \
    timeout = 1;           \
  } \
  result = (dhcp_timeout_t)timeout; \
} while(0)
```

DHCP_DEFINE_CUSTOM_TIMEOUTS: if this is defined then you can customize various DHCP timeouts using these macros:

- DHCP_SET_TIMEOUT_FROM_OFFERED_T0_LEASE() to adjust the t0 lease timeout from the offered value
- DHCP_SET_TIMEOUT_FROM_OFFERED_T1_RENEW() same for t1 renew
- DHCP_SET_TIMEOUT_FROM_OFFERED_T2_REBIND() same for t2 rebind
- DHCP_NEXT_TIMEOUT_THRESHOLD to adjust the period of the next timeout
- DHCP_REQUEST_BACKOFF_SEQUENCE to adjust back-off times based on DHCP request attempts

4.60.6 Enumeration Type Documentation

4.60.6.1 dhcp_option_idx

```
enum dhcp_option_idx
```

Option handling: options are parsed in dhcp_parse_reply and saved in an array where other functions can load them from. This might be moved into the struct dhcp (not necessarily since lwIP is single-threaded and the array is only used while in recv callback).

4.60.7 Function Documentation

4.60.7.1 dhcp_coarse_tmr()

```
void dhcp_coarse_tmr (void )
```

The DHCP timer that checks for lease renewal/rebind timeouts. Must be called once a minute (see DHCP_COARSE_TIMER_SECS).

4.60.7.2 dhcp_fine_tmr()

```
void dhcp_fine_tmr (void )
```

DHCP transaction timeout handling (this function must be called every 500ms, see DHCP_FINE_TIMER_MSECS).

A DHCP server is expected to respond within a short period of time. This timer checks whether an outstanding DHCP request is timed out.

4.60.7.3 dhcp_network_changed_link_up()

```
void dhcp_network_changed_link_up (struct netif * netif)
```

Handle a possible change in the network configuration.

This enters the REBOOTING state to verify that the currently bound address is still valid.

4.60.7.4 dhcp_supplied_address()

```
u8_t dhcp_supplied_address (const struct netif * netif)
```

check if DHCP supplied netif->ip_addr

Parameters

netif	the netif to check

Returns 1 if DHCP supplied netif->ip_addr (states BOUND or RENEWING), 0 otherwise

4.61 src/core/ipv4/etharp.c File Reference

```
#include "lwip/opt.h"#include "lwip/etharp.h"#include "lwip/stats.h"#include "lwip/snmp.h"# ←
    include "lwip/dhcp.h"#include "lwip/autoip.h"#include "lwip/acd.h"#include "lwip/prot/ ←
    iana.h"#include "netif/ethernet.h"#include <string.h>#include "path/to/my/lwip_hooks.h"
```

4.61.1 Macros

- #define ARP_AGE_REREQUEST_USED_UNICAST (ARP_MAXAGE - 30)
- #define ARP_MAXPENDING 5
- #define ETHARP_FLAG_TRY_HARD 1

4.61.2 Enumerations

- enum etharp_state

4.61.3 Functions

- void etharp_tmr (void)
- void etharp_cleanup_netif (struct netif *netif)
- ssize_t etharp_find_addr (struct netif *netif, const ip4_addr_t *ipaddr, struct eth_addr **eth_ret, const ip4_addr_t **ip_ret)
- int etharp_get_entry (size_t i, ip4_addr_t **ipaddr, struct netif **netif, struct eth_addr **eth_ret)
- void etharp_input (struct pbuf *p, struct netif *netif)
- err_t etharp_output (struct netif *netif, struct pbuf *q, const ip4_addr_t *ipaddr)
- err_t etharp_query (struct netif *netif, const ip4_addr_t *ipaddr, struct pbuf *q)
- err_t etharp_request (struct netif *netif, const ip4_addr_t *ipaddr)
- err_t etharp_acd_probe (struct netif *netif, const ip4_addr_t *ipaddr)
- err_t etharp_acd_announce (struct netif *netif, const ip4_addr_t *ipaddr)

4.61.4 Detailed Description

Address Resolution Protocol module for IP over Ethernet

Functionally, ARP is divided into two parts. The first maps an IP address to a physical address when sending a packet, and the second part answers requests from other machines for our physical address.

This implementation complies with RFC 826 (Ethernet ARP). It supports Gratuitous ARP from RFC3220 (IP Mobility Support for IPv4) section 4.6 if an interface calls etharp_gratuitous(our_netif) upon address change.

4.61.5 Macro Definition Documentation

4.61.5.1 ARP_AGE_REREQUEST_USED_UNICAST

```
#define ARP_AGE_REREQUEST_USED_UNICAST   (ARP_MAXAGE - 30)
```

Re-request a used ARP entry 1 minute before it would expire to prevent breaking a steadily used connection because the ARP entry timed out.

4.61.5.2 ARP_MAXPENDING

```
#define ARP_MAXPENDING   5
```

the time an ARP entry stays pending after first request, for ARP_TMR_INTERVAL = 1000, this is 10 seconds.

4.61.5.3 ETHARP_FLAG_TRY_HARD

```
#define ETHARP_FLAG_TRY_HARD   1
```

Try hard to create a new entry - we want the IP address to appear in the cache (even if this means removing an active entry or so).

4.61.6 Enumeration Type Documentation

4.61.6.1 etharp_state

```
enum etharp_state
```

ARP states

4.61.7 Function Documentation

4.61.7.1 etharp_acd_announce()

```
err_t etharp_acd_announce (struct netif * netif, const ip4_addr_t * ipaddr)
```

Send an ARP request packet announcing an ipaddr. Used to send announce messages for address conflict detection.

Parameters

netif	the lwip network interface on which to send the request
ipaddr	the IP address to announce

Returns ERR_OK if the request has been sent ERR_MEM if the ARP packet couldn't be allocated any other err_t on failure

4.61.7.2 etharp_acd_probe()

```
err_t etharp_acd_probe (struct netif * netif, const ip4_addr_t * ipaddr)
```

Send an ARP request packet probing for an ipaddr. Used to send probe messages for address conflict detection.

Parameters

netif	the lwip network interface on which to send the request
ipaddr	the IP address to probe

Returns ERR_OK if the request has been sent ERR_MEM if the ARP packet couldn't be allocated any other err_t on failure

4.61.7.3 etharp_cleanup_netif()

```
void etharp_cleanup_netif (struct netif * netif)
```

Remove all ARP table entries of the specified netif.

Parameters

netif	points to a network interface

4.61.7.4 etharp_find_addr()

```
ssize_t etharp_find_addr (struct netif * netif, const ip4_addr_t * ipaddr, struct eth_addr
** eth_ret, const ip4_addr_t ** ip_ret)
```

Finds (stable) ethernet/IP address pair from ARP table using interface and IP address index.

Note
the addresses in the ARP table are in network order!

Parameters

Returns table index if found, -1 otherwise

netif	points to interface index
ipaddr	points to the (network order) IP address index
eth_ret	points to return pointer
ip_ret	points to return pointer

4.61.7.5 etharp_get_entry()

```
int etharp_get_entry (size_t i, ip4_addr_t ** ipaddr, struct netif ** netif, struct eth_ad
** eth_ret)
```

Possibility to iterate over stable ARP table entries

Parameters

i	entry number, 0 to ARP_TABLE_SIZE
ipaddr	return value: IP address
netif	return value: points to interface
eth_ret	return value: ETH address

Returns 1 on valid index, 0 otherwise

4.61.7.6 etharp_input()

```
void etharp_input (struct pbuf * p, struct netif * netif)
```

Responds to ARP requests to us. Upon ARP replies to us, add entry to cache send out queued IP packets. Updates cache with snooped address pairs.

Should be called for incoming ARP packets. The pbuf in the argument is freed by this function.

Parameters

p	The ARP packet that arrived on netif. Is freed by this function.
netif	The lwIP network interface on which the ARP packet pbuf arrived.

See also pbuf_free()

4.61.7.7 etharp_output()

```
err_t etharp_output (struct netif * netif, struct pbuf * q, const ip4_addr_t * ipaddr)
```

Resolve and fill-in Ethernet address header for outgoing IP packet.

For IP multicast and broadcast, corresponding Ethernet addresses are selected and the packet is transmitted on the link.

For unicast addresses, the packet is submitted to etharp_query(). In case the IP address is outside the local network, the IP address of the gateway is used.

Parameters

Returns

- ERR_RTE No route to destination (no gateway to external networks), or the return type of either etharp_query() or ethernet_output().

netif	The lwIP network interface which the IP packet will be sent on.
q	The pbuf(s) containing the IP packet to be sent.
ipaddr	The IP address of the packet destination.

4.61.7.8 etharp_query()

```
err_t etharp_query (struct netif * netif, const ip4_addr_t * ipaddr, struct pbuf * q)
```

Send an ARP request for the given IP address and/or queue a packet.

If the IP address was not yet in the cache, a pending ARP cache entry is added and an ARP request is sent for the given address. The packet is queued on this entry.

If the IP address was already pending in the cache, a new ARP request is sent for the given address. The packet is queued on this entry.

If the IP address was already stable in the cache, and a packet is given, it is directly sent and no ARP request is sent out.

If the IP address was already stable in the cache, and no packet is given, an ARP request is sent out.

Parameters

netif	The lwIP network interface on which ipaddr must be queried for.
ipaddr	The IP address to be resolved.
q	If non-NULL, a pbuf that must be delivered to the IP address. q is not freed by this function.

Note
q must only be ONE packet, not a packet queue!

Returns

- ERR_BUF Could not make room for Ethernet header.
- ERR_MEM Hardware address unknown, and no more ARP entries available to query for address or queue the packet.
- ERR_MEM Could not queue packet due to memory shortage.
- ERR_RTE No route to destination (no gateway to external networks).
- ERR_ARG Non-unicast address given, those will not appear in ARP cache.

4.61.7.9 etharp_request()

```
err_t etharp_request (struct netif * netif, const ip4_addr_t * ipaddr)
```

Send an ARP request packet asking for ipaddr.

Parameters

netif	the lwip network interface on which to send the request
ipaddr	the IP address for which to ask

Returns ERR_OK if the request has been sent ERR_MEM if the ARP packet couldn't be allocated any other err_t on failure

4.61.7.10 etharp_tmr()

```
void etharp_tmr (void )
```

Clears expired entries in the ARP table.

This function should be called every ARP_TMR_INTERVAL milliseconds (1 second), in order to expire entries in the ARP table.

4.62 src/core/ipv4/icmp.c File Reference

```
#include "lwip/opt.h"#include "lwip/icmp.h"#include "lwip/inet_chksum.h"#include "lwip/ip.h ↩
    "#include "lwip/def.h"#include "lwip/stats.h"#include <string.h>#include "path/to/my/ ↩
    lwip_hooks.h"
```

4.62.1 Macros

- #define LWIP_ICMP_ECHO_CHECK_INPUT_PBUF_LEN 1

4.62.2 Functions

- void icmp_input (struct pbuf *p, struct netif *inp)
- void icmp_dest_unreach (struct pbuf *p, enum icmp_dur_type t)
- void icmp_time_exceeded (struct pbuf *p, enum icmp_te_type t)

4.62.3 Detailed Description

ICMP - Internet Control Message Protocol

4.62.4 Macro Definition Documentation

4.62.4.1 LWIP_ICMP_ECHO_CHECK_INPUT_PBUF_LEN

```
#define LWIP_ICMP_ECHO_CHECK_INPUT_PBUF_LEN   1
```

Small optimization: set to 0 if incoming PBUF_POOL pbuf always can be used to modify and send a response packet (and to 1 if this is not the case, e.g. when link header is stripped off when receiving)

4.62.5 Function Documentation

4.62.5.1 icmp_dest_unreach()

```
void icmp_dest_unreach (struct pbuf * p, enum icmp_dur_type t)
```

Send an icmp 'destination unreachable' packet, called from ip_input() if the transport layer protocol is unknown and from udp_input() if the local port is not bound.

Parameters

p	the input packet for which the 'unreachable' should be sent, p->payload pointing to the IP header
t	type of the 'unreachable' packet

p	the icmp echo request packet, p->payload pointing to the icmp header
inp	the netif on which this packet was received

4.62.5.2 icmp_input()

`void icmp_input (struct pbuf * p, struct netif * inp)`

Processes ICMP input packets, called from ip_input().

Currently only processes icmp echo requests and sends out the echo response.

Parameters

4.62.5.3 icmp_time_exceeded()

`void icmp_time_exceeded (struct pbuf * p, enum icmp_te_type t)`

Send a 'time exceeded' packet, called from ip_forward() if TTL is 0.

Parameters

p	the input packet for which the 'time exceeded' should be sent, p->payload pointing to the IP header
t	type of the 'time exceeded' packet

4.63 src/core/ipv4/igmp.c File Reference

```
#include "lwip/opt.h"#include "lwip/igmp.h"#include "lwip/debug.h"#include "lwip/def.h"# ↩
    include "lwip/mem.h"#include "lwip/ip.h"#include "lwip/inet_chksum.h"#include "lwip/ ↩
    netif.h"#include "lwip/stats.h"#include "lwip/prot/igmp.h"#include <string.h>
```

4.63.1 Functions

- void igmp_init (void)
- err_t igmp_start (struct netif *netif)
- err_t igmp_stop (struct netif *netif)
- void igmp_report_groups (struct netif *netif)
- struct igmp_group * igmp_lookfor_group (struct netif *ifp, const ip4_addr_t *addr)
- void igmp_input (struct pbuf *p, struct netif *inp, const ip4_addr_t *dest)
- err_t igmp_joingroup (const ip4_addr_t *ifaddr, const ip4_addr_t *groupaddr)
- err_t igmp_joingroup_netif (struct netif *netif, const ip4_addr_t *groupaddr)
- err_t igmp_leavegroup (const ip4_addr_t *ifaddr, const ip4_addr_t *groupaddr)
- err_t igmp_leavegroup_netif (struct netif *netif, const ip4_addr_t *groupaddr)
- void igmp_tmr (void)

4.63.2 Detailed Description

IGMP - Internet Group Management Protocol

4.63.3 Function Documentation

4.63.3.1 igmp_init()

`void igmp_init (void )`

Initialize the IGMP module

4.63.3.2 igmp_input()

`void igmp_input (struct pbuf * p, struct netif * inp, const ip4_addr_t * dest)`

Called from ip_input() if a new IGMP packet is received.

Parameters

p	received igmp packet, p->payload pointing to the igmp header
inp	network interface on which the packet was received
dest	destination ip address of the igmp packet

4.63.3.3 igmp_lookfor_group()

`struct igmp_group * igmp_lookfor_group (struct netif * ifp, const ip4_addr_t * addr)`

Search for a group in the netif's igmp group list

Parameters

ifp	the network interface for which to look
addr	the group ip address to search for

Returns a struct igmp_group* if the group has been found, NULL if the group wasn't found.

4.63.3.4 igmp_report_groups()

`void igmp_report_groups (struct netif * netif)`

Report IGMP memberships for this interface

Parameters

netif	network interface on which report IGMP memberships

4.63.3.5 igmp_start()

`err_t igmp_start (struct netif * netif)`

Start IGMP processing on interface

Parameters

netif	network interface on which start IGMP processing

4.63.3.6 igmp_stop()

```
err_t igmp_stop (struct netif * netif)
```

Stop IGMP processing on interface

Parameters

netif	network interface on which stop IGMP processing

4.63.3.7 igmp_tmr()

```
void igmp_tmr (void )
```

The igmp timer function (both for NO_SYS=1 and =0) Should be called every IGMP_TMR_INTERVAL milliseconds (100 ms is default).

4.64 src/core/ipv4/ip4.c File Reference

```
#include "lwip/opt.h"#include "lwip/ip.h"#include "lwip/def.h"#include "lwip/mem.h"#include ↩
     "lwip/ip4_frag.h"#include "lwip/inet_chksum.h"#include "lwip/netif.h"#include "lwip/ ↩
    icmp.h"#include "lwip/igmp.h"#include "lwip/priv/raw_priv.h"#include "lwip/udp.h"# ↩
    include "lwip/priv/tcp_priv.h"#include "lwip/autoip.h"#include "lwip/stats.h"#include " ↩
    lwip/prot/iana.h"#include <string.h>#include "path/to/my/lwip_hooks.h"
```

4.64.1 Macros

- #define LWIP_INLINE_IP_CHKSUM 1
- #define IP_ACCEPT_LINK_LAYER_ADDRESSED_PORT(port) ((port) == PP_NTOHS(LWIP_IANA_PORT_DHCP_CLIENT))

4.64.2 Functions

- void ip4_set_default_multicast_netif (struct netif *default_multicast_netif)
- struct netif * ip4_route_src (const ip4_addr_t *src, const ip4_addr_t *dest)
- struct netif * ip4_route (const ip4_addr_t *dest)
- err_t ip4_input (struct pbuf *p, struct netif *inp)
- err_t ip4_output_if (struct pbuf *p, const ip4_addr_t *src, const ip4_addr_t *dest, u8_t ttl, u8_t tos, u8_t proto, struct netif *netif)
- err_t ip4_output_if_opt (struct pbuf *p, const ip4_addr_t *src, const ip4_addr_t *dest, u8_t ttl, u8_t tos, u8_t proto, struct netif *netif, void *ip_options, u16_t optlen)
- err_t ip4_output_if_src (struct pbuf *p, const ip4_addr_t *src, const ip4_addr_t *dest, u8_t ttl, u8_t tos, u8_t proto, struct netif *netif)
- err_t ip4_output_if_opt_src (struct pbuf *p, const ip4_addr_t *src, const ip4_addr_t *dest, u8_t ttl, u8_t tos, u8_t proto, struct netif *netif, void *ip_options, u16_t optlen)
- err_t ip4_output (struct pbuf *p, const ip4_addr_t *src, const ip4_addr_t *dest, u8_t ttl, u8_t tos, u8_t proto)

4.64.3 Detailed Description

This is the IPv4 layer implementation for incoming and outgoing IP traffic.

See also ip_frag.c

4.64.4 Macro Definition Documentation

4.64.4.1 IP_ACCEPT_LINK_LAYER_ADDRESSED_PORT

```
#define IP_ACCEPT_LINK_LAYER_ADDRESSED_PORT( port)   ((port) == PP_NTOHS(LWIP_IANA_PORT_DH
```

Some defines for DHCP to let link-layer-addressed packets through while the netif is down. To use this in your own application/protocol, define LWIP_IP_ACCEPT_UDP_PORT(port) to return 1 if the port is accepted and 0 if the port is not accepted.

4.64.4.2 LWIP_INLINE_IP_CHKSUM

```
#define LWIP_INLINE_IP_CHKSUM   1
```

Set this to 0 in the rare case of wanting to call an extra function to generate the IP checksum (in contrast to calculating it on-the-fly).

4.64.5 Function Documentation

4.64.5.1 ip4_input()

```
err_t ip4_input (struct pbuf * p, struct netif * inp)
```

This function is called by the network interface device driver when an IP packet is received. The function does the basic checks of the IP header such as packet size being at least larger than the header size etc. If the packet was not destined for us, the packet is forwarded (using ip_forward). The IP checksum is always checked.

Finally, the packet is sent to the upper layer protocol input function.

Parameters

p	the received IP packet (p->payload points to IP header)
inp	the netif on which this packet was received

Returns ERR_OK if the packet was processed (could return ERR_* if it wasn't processed, but currently always returns ERR_OK)

4.64.5.2 ip4_output()

```
err_t ip4_output (struct pbuf * p, const ip4_addr_t * src, const ip4_addr_t * dest, u8_t
ttl, u8_t tos, u8_t proto)
```

Simple interface to ip_output_if. It finds the outgoing network interface and calls upon ip_output_if to do the actual work.

Parameters

Returns ERR_RTE if no route is found see ip_output_if() for more return values

p	the packet to send (p->payload points to the data, e.g. next protocol header; if dest == LWIP_IP_HDRINCL, p already includes an IP header and p->payload points to that IP header)
src	the source IP address to send from (if src == IP4_ADDR_ANY, the IP address of the netif used to send is used as source address)
dest	the destination IP address to send the packet to
ttl	the TTL value to be set in the IP header
tos	the TOS value to be set in the IP header
proto	the PROTOCOL to be set in the IP header

4.64.5.3 ip4_output_if()

```
err_t ip4_output_if (struct pbuf * p, const ip4_addr_t * src, const ip4_addr_t * dest,
u8_t ttl, u8_t tos, u8_t proto, struct netif * netif)
```

Sends an IP packet on a network interface. This function constructs the IP header and calculates the IP header checksum. If the source IP address is NULL, the IP address of the outgoing network interface is filled in as source address. If the destination IP address is LWIP_IP_HDRINCL, p is assumed to already include an IP header and p->payload points to it instead of the data.

Parameters

p	the packet to send (p->payload points to the data, e.g. next protocol header; if dest == LWIP_IP_HDRINCL, p already includes an IP header and p->payload points to that IP header)
src	the source IP address to send from (if src == IP4_ADDR_ANY, the IP address of the netif used to send is used as source address)
dest	the destination IP address to send the packet to
ttl	the TTL value to be set in the IP header
tos	the TOS value to be set in the IP header
proto	the PROTOCOL to be set in the IP header
netif	the netif on which to send this packet

Returns ERR_OK if the packet was sent OK ERR_BUF if p doesn't have enough space for IP/LINK headers returns errors returned by netif->output

Note
ip_id: RFC791 "some host may be able to simply use unique identifiers independent of destination"

4.64.5.4 ip4_output_if_opt()

```
err_t ip4_output_if_opt (struct pbuf * p, const ip4_addr_t * src, const ip4_addr_t * dest,
u8_t ttl, u8_t tos, u8_t proto, struct netif * netif, void * ip_options, u16_t optlen)
```

Same as ip_output_if() but with the possibility to include IP options:

@ param ip_options pointer to the IP options, copied into the IP header @ param optlen length of ip_options

4.64.5.5 ip4_output_if_opt_src()

```
err_t ip4_output_if_opt_src (struct pbuf * p, const ip4_addr_t * src, const ip4_addr_t
* dest, u8_t ttl, u8_t tos, u8_t proto, struct netif * netif, void * ip_options, u16_t
optlen)
```

Same as ip_output_if_opt() but 'src' address is not replaced by netif address when it is 'any'.

4.64.5.6 ip4_output_if_src()

```
err_t ip4_output_if_src (struct pbuf * p, const ip4_addr_t * src, const ip4_addr_t * dest,
u8_t ttl, u8_t tos, u8_t proto, struct netif * netif)
```

Same as ip_output_if() but 'src' address is not replaced by netif address when it is 'any'.

4.64.5.7 ip4_route()

```
struct netif * ip4_route (const ip4_addr_t * dest)
```

Finds the appropriate network interface for a given IP address. It searches the list of network interfaces linearly. A match is found if the masked IP address of the network interface equals the masked IP address given to the function.

Parameters

dest	the destination IP address for which to find the route

Returns the netif on which to send to reach dest

4.64.5.8 ip4_route_src()

```
struct netif * ip4_route_src (const ip4_addr_t * src, const ip4_addr_t * dest)
```

Source based IPv4 routing must be fully implemented in LWIP_HOOK_IP4_ROUTE_SRC(). This function only provides the parameters.

4.65 src/core/ipv4/ip4_addr.c File Reference

```
#include "lwip/opt.h"#include "lwip/ip_addr.h"#include "lwip/netif.h"
```

4.65.1 Functions

- u8_t ip4_addr_isbroadcast_u32 (u32_t addr, const struct netif *netif)
- u8_t ip4_addr_netmask_valid (u32_t netmask)
- u32_t ipaddr_addr (const char *cp)
- int ip4addr_aton (const char *cp, ip4_addr_t *addr)
- char * ip4addr_ntoa (const ip4_addr_t *addr)
- char * ip4addr_ntoa_r (const ip4_addr_t *addr, char *buf, int buflen)

4.65.2 Detailed Description

This is the IPv4 address tools implementation.

addr	address to be checked
netif	the network interface against which the address is checked

4.65.3 Function Documentation

4.65.3.1 ip4_addr_isbroadcast_u32()

`u8_t ip4_addr_isbroadcast_u32 (u32_t addr, const struct netif * netif)`

Determine if an address is a broadcast address on a network interface

Parameters

Returns returns non-zero if the address is a broadcast address

4.65.3.2 ip4_addr_netmask_valid()

`u8_t ip4_addr_netmask_valid (u32_t netmask)`

Checks if a netmask is valid (starting with ones, then only zeros)

Parameters

netmask	the IPv4 netmask to check (in network byte order!)

Returns 1 if the netmask is valid, 0 if it is not

4.65.3.3 ip4addr_aton()

`int ip4addr_aton (const char * cp, ip4_addr_t * addr)`

Check whether "cp" is a valid ascii representation of an Internet address and convert to a binary address. Returns 1 if the address is valid, 0 if not. This replaces inet_addr, the return value from which cannot distinguish between failure and a local broadcast address.

Parameters

cp	IP address in ascii representation (e.g. "127.0.0.1")
addr	pointer to which to save the ip address in network order

Returns 1 if cp could be converted to addr, 0 on failure

4.65.3.4 ip4addr_ntoa()

`char * ip4addr_ntoa (const ip4_addr_t * addr)`

Convert numeric IP address into decimal dotted ASCII representation. returns ptr to static buffer; not reentrant!

Parameters

Returns pointer to a global static (!) buffer that holds the ASCII representation of addr

addr	ip address in network order to convert

addr	ip address in network order to convert
buf	target buffer where the string is stored
buflen	length of buf

4.65.3.5 ip4addr_ntoa_r()

```
char * ip4addr_ntoa_r (const ip4_addr_t * addr, char * buf, int buflen)
```

Same as ip4addr_ntoa, but reentrant since a user-supplied buffer is used.

Parameters

Returns either pointer to buf which now holds the ASCII representation of addr or NULL if buf was too small

4.65.3.6 ipaddr_addr()

```
u32_t ipaddr_addr (const char * cp)
```

Ascii internet address interpretation routine. The value returned is in network order.

Parameters

cp	IP address in ascii representation (e.g. "127.0.0.1")

Returns ip address in network order

4.66 src/core/ipv4/ip4_frag.c File Reference

```
#include "lwip/opt.h"#include "lwip/ip4_frag.h"#include "lwip/def.h"#include "lwip/ ↩
    inet_chksum.h"#include "lwip/netif.h"#include "lwip/stats.h"#include "lwip/icmp.h"# ↩
    include <string.h>#include "arch/bpstruct.h"#include "arch/epstruct.h"
```

4.66.1 Data Structures

- struct ip_reass_helper

4.66.2 Macros

- #define IP_REASS_CHECK_OVERLAP 1
- #define IP_REASS_FREE_OLDEST 1

4.66.3 Functions

- void ip_reass_tmr (void)
- struct pbuf * ip4_reass (struct pbuf *p)
- err_t ip4_frag (struct pbuf *p, struct netif *netif, const ip4_addr_t *dest)

4.66.4 Detailed Description

This is the IPv4 packet segmentation and reassembly implementation.

4.66.5 Macro Definition Documentation

4.66.5.1 IP_REASS_CHECK_OVERLAP

```
#define IP_REASS_CHECK_OVERLAP   1
```

The IP reassembly code currently has the following limitations:

- IP header options are not supported
- fragments must not overlap (e.g. due to different routes), currently, overlapping or duplicate fragments are thrown away if IP_REASS_CHECK_OVERLAP=1 (the default)!

  Setting this to 0, you can turn off checking the fragments for overlapping regions. The code gets a little smaller. Only use this if you know that overlapping won't occur on your network!

4.66.5.2 IP_REASS_FREE_OLDEST

```
#define IP_REASS_FREE_OLDEST   1
```

Set to 0 to prevent freeing the oldest datagram when the reassembly buffer is full (IP_REASS_MAX_PBUFS pbufs are enqueued). The code gets a little smaller. Datagrams will be freed by timeout only. Especially useful when MEMP_NUM_REASSDATA is set to 1, so one datagram can be reassembled at a time, only.

4.66.6 Function Documentation

4.66.6.1 ip4_frag()

```
err_t ip4_frag (struct pbuf * p, struct netif * netif, const ip4_addr_t * dest)
```

Fragment an IP datagram if too large for the netif.

Chop the datagram in MTU sized chunks and send them in order by pointing PBUF_REFs into p.

Parameters

p	ip packet to send
netif	the netif on which to send
dest	destination ip address to which to send

Returns ERR_OK if sent successfully, err_t otherwise

4.66.6.2 ip4_reass()

```
struct pbuf * ip4_reass (struct pbuf * p)
```

Reassembles incoming IP fragments into an IP datagram.

Parameters

Returns NULL if reassembly is incomplete, ? otherwise

p	points to a pbuf chain of the fragment

4.66.6.3 ip_reass_tmr()

```
void ip_reass_tmr (void )
```

Reassembly timer base function for both NO_SYS == 0 and 1 (!).

Should be called every 1000 msec (defined by IP_TMR_INTERVAL).

4.67 src/core/ipv6/dhcp6.c File Reference

```
#include "lwip/opt.h"#include "lwip/dhcp6.h"#include "lwip/prot/dhcp6.h"#include "lwip/def. ↩
    h"#include "lwip/udp.h"#include "lwip/dns.h"#include <string.h>#include "path/to/my/ ↩
    lwip_hooks.h"
```

4.67.1 Enumerations

- enum dhcp6_option_idx

4.67.2 Functions

- void dhcp6_set_struct (struct netif *netif, struct dhcp6 *dhcp6)
- void dhcp6_cleanup (struct netif *netif)
- err_t dhcp6_enable_stateful (struct netif *netif)
- err_t dhcp6_enable_stateless (struct netif *netif)
- void dhcp6_disable (struct netif *netif)
- void dhcp6_nd6_ra_trigger (struct netif *netif, u8_t managed_addr_config, u8_t other_config)
- void dhcp6_tmr (void)

4.67.3 Variables

- struct dhcp6_option_info dhcp6_rx_options [DHCP6_OPTION_IDX_MAX]

4.67.4 Enumeration Type Documentation

4.67.4.1 dhcp6_option_idx

```
enum dhcp6_option_idx
```

Option handling: options are parsed in dhcp6_parse_reply and saved in an array where other functions can load them from. This might be moved into the struct dhcp6 (not necessarily since lwIP is single-threaded and the array is only used while in recv callback).

4.67.5 Function Documentation

4.67.5.1 dhcp6_nd6_ra_trigger()

```
void dhcp6_nd6_ra_trigger (struct netif * netif, u8_t managed_addr_config, u8_t other_conf
```

This function is called from nd6 module when an RA message is received It triggers DHCPv6 requests (if enabled).

4.67.5.2 dhcp6_tmr()

```
void dhcp6_tmr (void )
```

DHCPv6 timeout handling (this function must be called every 500ms, see DHCP6_TIMER_MSECS).

A DHCPv6 server is expected to respond within a short period of time. This timer checks whether an outstanding DHCPv6 request is timed out.

4.67.6 Variable Documentation

4.67.6.1 dhcp6_rx_options

```
struct dhcp6_option_info dhcp6_rx_options[DHCP6_OPTION_IDX_MAX]
```

Holds the decoded option info, only valid while in dhcp6_recv.

4.68 src/core/ipv6/ethip6.c File Reference

```
#include "lwip/opt.h"#include "lwip/ethip6.h"#include "lwip/nd6.h"#include "lwip/pbuf.h"# ↩
    include "lwip/ip6.h"#include "lwip/ip6_addr.h"#include "lwip/inet_chksum.h"#include " ↩
    lwip/netif.h"#include "lwip/icmp6.h"#include "lwip/prot/ethernet.h"#include "netif/ ↩
    ethernet.h"#include <string.h>
```

4.68.1 Functions

- err_t ethip6_output (struct netif *netif, struct pbuf *q, const ip6_addr_t *ip6addr)

4.68.2 Detailed Description

Ethernet output for IPv6. Uses ND tables for link-layer addressing.

4.68.3 Function Documentation

4.68.3.1 ethip6_output()

```
err_t ethip6_output (struct netif * netif, struct pbuf * q, const ip6_addr_t * ip6addr)
```

Resolve and fill-in Ethernet address header for outgoing IPv6 packet.

For IPv6 multicast, corresponding Ethernet addresses are selected and the packet is transmitted on the link.

For unicast addresses, ask the ND6 module what to do. It will either let us send the the packet right away, or queue the packet for later itself, unless an error occurs.

Parameters

Returns

- ERR_OK or the return value of nd6_get_next_hop_addr_or_queue.

netif	The lwIP network interface which the IP packet will be sent on.
q	The pbuf(s) containing the IP packet to be sent.
ip6addr	The IP address of the packet destination.

4.69 src/core/ipv6/icmp6.c File Reference

```
#include "lwip/opt.h"#include "lwip/icmp6.h"#include "lwip/prot/icmp6.h"#include "lwip/ip6. ↩
    h"#include "lwip/ip6_addr.h"#include "lwip/inet_chksum.h"#include "lwip/pbuf.h"#include  ↩
    "lwip/netif.h"#include "lwip/nd6.h"#include "lwip/mld6.h"#include "lwip/ip.h"#include " ↩
    lwip/stats.h"#include <string.h>
```

4.69.1 Functions

- void icmp6_input (struct pbuf *p, struct netif *inp)
- void icmp6_dest_unreach (struct pbuf *p, enum icmp6_dur_code c)
- void icmp6_packet_too_big (struct pbuf *p, u32_t mtu)
- void icmp6_time_exceeded (struct pbuf *p, enum icmp6_te_code c)
- void icmp6_time_exceeded_with_addrs (struct pbuf *p, enum icmp6_te_code c, const ip6_addr_t *src_addr, const ip6_addr_t *dest_addr)
- void icmp6_param_problem (struct pbuf *p, enum icmp6_pp_code c, const void *pointer)

4.69.2 Detailed Description

IPv6 version of ICMP, as per RFC 4443.

4.69.3 Function Documentation

4.69.3.1 icmp6_dest_unreach()

```
void icmp6_dest_unreach (struct pbuf * p, enum icmp6_dur_code c)
```

Send an icmpv6 'destination unreachable' packet.

This function must be used only in direct response to a packet that is being received right now. Otherwise, address zones would be lost.

Parameters

p	the input packet for which the 'unreachable' should be sent, p->payload pointing to the IPv6 header
c	ICMPv6 code for the unreachable type

4.69.3.2 icmp6_input()

```
void icmp6_input (struct pbuf * p, struct netif * inp)
```

Process an input ICMPv6 message. Called by ip6_input.

Will generate a reply for echo requests. Other messages are forwarded to nd6_input, or mld6_input.

Parameters

p	the mld packet, p->payload pointing to the icmpv6 header
inp	the netif on which this packet was received

4.69.3.3 icmp6_packet_too_big()

```
void icmp6_packet_too_big (struct pbuf * p, u32_t mtu)
```

Send an icmpv6 'packet too big' packet.

This function must be used only in direct response to a packet that is being received right now. Otherwise, address zones would be lost.

Parameters

p	the input packet for which the 'packet too big' should be sent, p->payload pointing to the IPv6 header
mtu	the maximum mtu that we can accept

4.69.3.4 icmp6_param_problem()

```
void icmp6_param_problem (struct pbuf * p, enum icmp6_pp_code c, const void * pointer)
```

Send an icmpv6 'parameter problem' packet.

This function must be used only in direct response to a packet that is being received right now. Otherwise, address zones would be lost and the calculated offset would be wrong (calculated against ip6_current_header()).

Parameters

p	the input packet for which the 'param problem' should be sent, p->payload pointing to the IP header
c	ICMPv6 code for the param problem type
pointer	the pointer to the byte where the parameter is found

4.69.3.5 icmp6_time_exceeded()

```
void icmp6_time_exceeded (struct pbuf * p, enum icmp6_te_code c)
```

Send an icmpv6 'time exceeded' packet.

This function must be used only in direct response to a packet that is being received right now. Otherwise, address zones would be lost.

Parameters

4.69.3.6 icmp6_time_exceeded_with_addrs()

```
void icmp6_time_exceeded_with_addrs (struct pbuf * p, enum icmp6_te_code c, const ip6_addr
* src_addr, const ip6_addr_t * dest_addr)
```

Send an icmpv6 'time exceeded' packet, with explicit source and destination addresses.

This function may be used to send a response sometime after receiving the packet for which this response is meant. The provided source and destination addresses are used primarily to retain their zone information.

Parameters

p	the input packet for which the 'time exceeded' should be sent, p->payload pointing to the IPv6 header
c	ICMPv6 code for the time exceeded type

p	the input packet for which the 'time exceeded' should be sent, p->payload pointing to the IPv6 header
c	ICMPv6 code for the time exceeded type
src_addr	source address of the original packet, with zone information
dest_addr	destination address of the original packet, with zone information

4.70 src/core/ipv6/inet6.c File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/inet.h"
```

4.70.1 Variables

- const struct in6_addr in6addr_any = {{{0,0,0,0}}}

4.70.2 Detailed Description

INET v6 addresses.

4.70.3 Variable Documentation

4.70.3.1 in6addr_any

```
const struct in6_addr in6addr_any = {{{0,0,0,0}}}
```

This variable is initialized by the system to contain the wildcard IPv6 address.

4.71 src/core/ipv6/ip6.c File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/mem.h"#include "lwip/netif.h"# ←
    include "lwip/ip.h"#include "lwip/ip6.h"#include "lwip/ip6_addr.h"#include "lwip/ ←
    ip6_frag.h"#include "lwip/icmp6.h"#include "lwip/priv/raw_priv.h"#include "lwip/udp.h"# ←
    include "lwip/priv/tcp_priv.h"#include "lwip/dhcp6.h"#include "lwip/nd6.h"#include "lwip ←
    /mld6.h"#include "lwip/debug.h"#include "lwip/stats.h"#include "path/to/my/lwip_hooks.h"
```

4.71.1 Functions

- struct netif * ip6_route (const ip6_addr_t *src, const ip6_addr_t *dest)
- const ip_addr_t * ip6_select_source_address (struct netif *netif, const ip6_addr_t *dest)
- err_t ip6_input (struct pbuf *p, struct netif *inp)
- err_t ip6_output_if (struct pbuf *p, const ip6_addr_t *src, const ip6_addr_t *dest, u8_t hl, u8_t tc, u8_t nexth, struct netif *netif)

- err_t ip6_output_if_src (struct pbuf *p, const ip6_addr_t *src, const ip6_addr_t *dest, u8_t hl, u8_t tc, u8_t nexth, struct netif *netif)
- err_t ip6_output (struct pbuf *p, const ip6_addr_t *src, const ip6_addr_t *dest, u8_t hl, u8_t tc, u8_t nexth)
- err_t ip6_options_add_hbh_ra (struct pbuf *p, u8_t nexth, u8_t value)

4.71.2 Detailed Description

IPv6 layer.

4.71.3 Function Documentation

4.71.3.1 ip6_input()

```
err_t ip6_input (struct pbuf * p, struct netif * inp)
```

This function is called by the network interface device driver when an IPv6 packet is received. The function does the basic checks of the IP header such as packet size being at least larger than the header size etc. If the packet was not destined for us, the packet is forwarded (using ip6_forward).

Finally, the packet is sent to the upper layer protocol input function.

Parameters

p	the received IPv6 packet (p->payload points to IPv6 header)
inp	the netif on which this packet was received

Returns ERR_OK if the packet was processed (could return ERR_* if it wasn't processed, but currently always returns ERR_OK)

4.71.3.2 ip6_options_add_hbh_ra()

```
err_t ip6_options_add_hbh_ra (struct pbuf * p, u8_t nexth, u8_t value)
```

Add a hop-by-hop options header with a router alert option and padding.

Used by MLD when sending a Multicast listener report/done message.

Parameters

p	the packet to which we will prepend the options header
nexth	the next header protocol number (e.g. IP6_NEXTH_ICMP6)
value	the value of the router alert option data (e.g. IP6_ROUTER_ALERT_VALUE_MLD)

Returns ERR_OK if hop-by-hop header was added, ERR_* otherwise

4.71.3.3 ip6_output()

```
err_t ip6_output (struct pbuf * p, const ip6_addr_t * src, const ip6_addr_t * dest, u8_t
hl, u8_t tc, u8_t nexth)
```

Simple interface to ip6_output_if. It finds the outgoing network interface and calls upon ip6_output_if to do the actual work.

Parameters

Returns ERR_RTE if no route is found see ip_output_if() for more return values

p	the packet to send (p->payload points to the data, e.g. next protocol header; if dest == LWIP_IP_HDRINCL, p already includes an IPv6 header and p->payload points to that IPv6 header)
src	the source IPv6 address to send from (if src == IP6_ADDR_ANY, an IP address of the netif is selected and used as source address. if src == NULL, IP6_ADDR_ANY is used as source)
dest	the destination IPv6 address to send the packet to
hl	the Hop Limit value to be set in the IPv6 header
tc	the Traffic Class value to be set in the IPv6 header
nexth	the Next Header to be set in the IPv6 header

4.71.3.4 ip6_output_if()

```
err_t ip6_output_if (struct pbuf * p, const ip6_addr_t * src, const ip6_addr_t * dest,
u8_t hl, u8_t tc, u8_t nexth, struct netif * netif)
```

Sends an IPv6 packet on a network interface. This function constructs the IPv6 header. If the source IPv6 address is NULL, the IPv6 "ANY" address is used as source (usually during network startup). If the source IPv6 address it IP6_ADDR_ANY, the most appropriate IPv6 address of the outgoing network interface is filled in as source address. If the destination IPv6 address is LWIP_IP_HDRINCL, p is assumed to already include an IPv6 header and p->payload points to it instead of the data.

Parameters

p	the packet to send (p->payload points to the data, e.g. next protocol header; if dest == LWIP_IP_HDRINCL, p already includes an IPv6 header and p->payload points to that IPv6 header)
src	the source IPv6 address to send from (if src == IP6_ADDR_ANY, an IP address of the netif is selected and used as source address. if src == NULL, IP6_ADDR_ANY is used as source) (src is possibly not properly zoned)
dest	the destination IPv6 address to send the packet to (possibly not properly zoned)
hl	the Hop Limit value to be set in the IPv6 header
tc	the Traffic Class value to be set in the IPv6 header
nexth	the Next Header to be set in the IPv6 header
netif	the netif on which to send this packet

Returns ERR_OK if the packet was sent OK ERR_BUF if p doesn't have enough space for IPv6/LINK headers returns errors returned by netif->output_ip6

4.71.3.5 ip6_output_if_src()

```
err_t ip6_output_if_src (struct pbuf * p, const ip6_addr_t * src, const ip6_addr_t * dest,
u8_t hl, u8_t tc, u8_t nexth, struct netif * netif)
```

Same as ip6_output_if() but 'src' address is not replaced by netif address when it is 'any'.

4.71.3.6 ip6_route()

```
struct netif * ip6_route (const ip6_addr_t * src, const ip6_addr_t * dest)
```

Finds the appropriate network interface for a given IPv6 address. It tries to select a netif following a sequence of heuristics: 1) if there is only 1 netif, return it 2) if the destination is a zoned address, match its zone to a netif 3) if the either the source or destination address is a scoped address, match the source address's zone (if set) or address (if not) to a netif 4) tries to match the destination subnet to a configured address 5) tries to find a router-announced route 6) tries to match the (unscoped) source address to the netif 7) returns the default netif, if configured

Note that each of the two given addresses may or may not be properly zoned.

src	the source IPv6 address, if known
dest	the destination IPv6 address for which to find the route

Parameters

Returns the netif on which to send to reach dest

4.72 src/core/ipv6/ip6_addr.c File Reference

```
#include "lwip/opt.h"#include "lwip/ip_addr.h"#include "lwip/def.h"#include "lwip/netif.h"# ↩
    include <string.h>#include "lwip/ip4_addr.h"
```

4.72.1 Functions

- int ip6addr_aton (const char *cp, ip6_addr_t *addr)
- char * ip6addr_ntoa (const ip6_addr_t *addr)
- char * ip6addr_ntoa_r (const ip6_addr_t *addr, char *buf, int buflen)

4.72.2 Detailed Description

IPv6 addresses.

4.72.3 Function Documentation

4.72.3.1 ip6addr_aton()

```
int ip6addr_aton (const char * cp, ip6_addr_t * addr)
```

Check whether "cp" is a valid ascii representation of an IPv6 address and convert to a binary address. Returns 1 if the address is valid, 0 if not.

Parameters

cp	IPv6 address in ascii representation (e.g. "FF01::1")
addr	pointer to which to save the ip address in network order

Returns 1 if cp could be converted to addr, 0 on failure

4.72.3.2 ip6addr_ntoa()

```
char * ip6addr_ntoa (const ip6_addr_t * addr)
```

Convert numeric IPv6 address into ASCII representation. returns ptr to static buffer; not reentrant!

Parameters

addr	ip6 address in network order to convert

Returns pointer to a global static (!) buffer that holds the ASCII representation of addr

4.72.3.3 ip6addr_ntoa_r()

```
char * ip6addr_ntoa_r (const ip6_addr_t * addr, char * buf, int buflen)
```

Same as ipaddr_ntoa, but reentrant since a user-supplied buffer is used.

Parameters

addr	ip6 address in network order to convert
buf	target buffer where the string is stored
buflen	length of buf

Returns either pointer to buf which now holds the ASCII representation of addr or NULL if buf was too small

4.73 src/core/ipv6/ip6_frag.c File Reference

```
#include "lwip/opt.h"#include "lwip/ip6_frag.h"#include "lwip/ip6.h"#include "lwip/icmp6.h ←
    "#include "lwip/nd6.h"#include "lwip/ip.h"#include "lwip/pbuf.h"#include "lwip/memp.h"# ←
    include "lwip/stats.h"#include <string.h>#include "arch/bpstruct.h"#include "arch/ ←
    epstruct.h"
```

4.73.1 Data Structures

- struct ip6_reass_helper

4.73.2 Macros

- #define IP_REASS_CHECK_OVERLAP 1
- #define IP_REASS_FREE_OLDEST 1

4.73.3 Functions

- struct pbuf * ip6_reass (struct pbuf *p)
- err_t ip6_frag (struct pbuf *p, struct netif *netif, const ip6_addr_t *dest)

4.73.4 Detailed Description

IPv6 fragmentation and reassembly.

4.73.5 Macro Definition Documentation

4.73.5.1 IP_REASS_CHECK_OVERLAP

```
#define IP_REASS_CHECK_OVERLAP 1
```

Setting this to 0, you can turn off checking the fragments for overlapping regions. The code gets a little smaller. Only use this if you know that overlapping won't occur on your network!

4.73.5.2 IP_REASS_FREE_OLDEST

```
#define IP_REASS_FREE_OLDEST   1
```

Set to 0 to prevent freeing the oldest datagram when the reassembly buffer is full (IP_REASS_MAX_PBUFS pbufs are enqueued). The code gets a little smaller. Datagrams will be freed by timeout only. Especially useful when MEMP_NUM_REASSDATA is set to 1, so one datagram can be reassembled at a time, only.

4.73.6 Function Documentation

4.73.6.1 ip6_frag()

```
err_t ip6_frag (struct pbuf * p, struct netif * netif, const ip6_addr_t * dest)
```

Fragment an IPv6 datagram if too large for the netif or path MTU.

Chop the datagram in MTU sized chunks and send them in order by pointing PBUF_REFs into p

Parameters

p	ipv6 packet to send
netif	the netif on which to send
dest	destination ipv6 address to which to send

Returns ERR_OK if sent successfully, err_t otherwise

4.73.6.2 ip6_reass()

```
struct pbuf * ip6_reass (struct pbuf * p)
```

Reassembles incoming IPv6 fragments into an IPv6 datagram.

Parameters

p	points to the IPv6 Fragment Header

Returns NULL if reassembly is incomplete, pbuf pointing to IPv6 Header if reassembly is complete

4.74 src/core/ipv6/mld6.c File Reference

```
#include "lwip/opt.h"#include "lwip/mld6.h"#include "lwip/prot/mld6.h"#include "lwip/icmp6. ↩
    h"#include "lwip/ip6.h"#include "lwip/ip6_addr.h"#include "lwip/ip.h"#include "lwip/ ↩
    inet_chksum.h"#include "lwip/pbuf.h"#include "lwip/netif.h"#include "lwip/memp.h"# ↩
    include "lwip/stats.h"#include <string.h>
```

4.74.1 Functions

- err_t mld6_stop (struct netif *netif)
- void mld6_report_groups (struct netif *netif)
- struct mld_group * mld6_lookfor_group (struct netif *ifp, const ip6_addr_t *addr)
- void mld6_input (struct pbuf *p, struct netif *inp)

- err_t mld6_joingroup (const ip6_addr_t *srcaddr, const ip6_addr_t *groupaddr)
- err_t mld6_joingroup_netif (struct netif *netif, const ip6_addr_t *groupaddr)
- err_t mld6_leavegroup (const ip6_addr_t *srcaddr, const ip6_addr_t *groupaddr)
- err_t mld6_leavegroup_netif (struct netif *netif, const ip6_addr_t *groupaddr)
- void mld6_tmr (void)

4.74.2 Detailed Description

Multicast listener discovery

4.74.3 Function Documentation

4.74.3.1 mld6_input()

```
void mld6_input (struct pbuf * p, struct netif * inp)
```

Process an input MLD message. Called by icmp6_input.

Parameters

p	the mld packet, p->payload pointing to the icmpv6 header
inp	the netif on which this packet was received

4.74.3.2 mld6_lookfor_group()

```
struct mld_group * mld6_lookfor_group (struct netif * ifp, const ip6_addr_t * addr)
```

Search for a group that is joined on a netif

Parameters

ifp	the network interface for which to look
addr	the group ipv6 address to search for

Returns a struct mld_group* if the group has been found, NULL if the group wasn't found.

4.74.3.3 mld6_report_groups()

```
void mld6_report_groups (struct netif * netif)
```

Report MLD memberships for this interface

Parameters

netif	network interface on which report MLD memberships

netif	network interface on which stop MLD processing

4.74.3.4 mld6_stop()

```
err_t mld6_stop (struct netif * netif)
```

Stop MLD processing on interface

Parameters

4.74.3.5 mld6_tmr()

```
void mld6_tmr (void )
```

Periodic timer for mld processing. Must be called every MLD6_TMR_INTERVAL milliseconds (100).

When a delaying member expires, a membership report is sent.

4.75 src/core/ipv6/nd6.c File Reference

```
#include "lwip/opt.h"#include "lwip/nd6.h"#include "lwip/priv/nd6_priv.h"#include "lwip/ ↩
    prot/nd6.h"#include "lwip/prot/icmp6.h"#include "lwip/pbuf.h"#include "lwip/mem.h"# ↩
    include "lwip/memp.h"#include "lwip/ip6.h"#include "lwip/ip6_addr.h"#include "lwip/ ↩
    inet_chksum.h"#include "lwip/netif.h"#include "lwip/icmp6.h"#include "lwip/mld6.h"# ↩
    include "lwip/dhcp6.h"#include "lwip/ip.h"#include "lwip/stats.h"#include "lwip/dns.h"# ↩
    include <string.h>#include "path/to/my/lwip_hooks.h"
```

4.75.1 Functions

- void nd6_input (struct pbuf *p, struct netif *inp)
- void nd6_tmr (void)
- void nd6_clear_destination_cache (void)
- struct netif * nd6_find_route (const ip6_addr_t *ip6addr)
- err_t nd6_get_next_hop_addr_or_queue (struct netif *netif, struct pbuf *q, const ip6_addr_t *ip6addr, const u8_t **hwaddrp)
- u16_t nd6_get_destination_mtu (const ip6_addr_t *ip6addr, struct netif *netif)
- void nd6_reachability_hint (const ip6_addr_t *ip6addr)
- void nd6_cleanup_netif (struct netif *netif)
- void nd6_adjust_mld_membership (struct netif *netif, s8_t addr_idx, u8_t new_state)
- void nd6_restart_netif (struct netif *netif)

4.75.2 Detailed Description

Neighbor discovery and stateless address autoconfiguration for IPv6. Aims to be compliant with RFC 4861 (Neighbor discovery) and RFC 4862 (Address autoconfiguration).

4.75.3 Function Documentation

4.75.3.1 nd6_adjust_mld_membership()

```
void nd6_adjust_mld_membership (struct netif * netif, s8_t addr_idx, u8_t new_state)
```

The state of a local IPv6 address entry is about to change. If needed, join or leave the solicited-node multicast group for the address.

Parameters

netif	The netif that owns the address.
addr_idx	The index of the address.
new_state	The new (IP6_ADDR_) state for the address.

4.75.3.2 nd6_cleanup_netif()

```
void nd6_cleanup_netif (struct netif * netif)
```

Remove all prefix, neighbor_cache and router entries of the specified netif.

Parameters

netif	points to a network interface

4.75.3.3 nd6_clear_destination_cache()

```
void nd6_clear_destination_cache (void )
```

Clear the destination cache.

This operation may be necessary for consistency in the light of changing local addresses and/or use of the gateway hook.

4.75.3.4 nd6_find_route()

```
struct netif * nd6_find_route (const ip6_addr_t * ip6addr)
```

Find a router-announced route to the given destination. This route may be based on an on-link prefix or a default router.

If a suitable route is found, the returned netif is guaranteed to be in a suitable state (up, link up) to be used for packet transmission.

Parameters

ip6addr	the destination IPv6 address

Returns the netif to use for the destination, or NULL if none found

4.75.3.5 nd6_get_destination_mtu()

```
u16_t nd6_get_destination_mtu (const ip6_addr_t * ip6addr, struct netif * netif)
```

Get the Path MTU for a destination.

Parameters

Returns the Path MTU, if known, or the netif default MTU

ip6addr	the destination address
netif	the netif on which the packet will be sent

4.75.3.6 nd6_get_next_hop_addr_or_queue()

```
err_t nd6_get_next_hop_addr_or_queue (struct netif * netif, struct pbuf * q, const ip6_add
* ip6addr, const u8_t ** hwaddrp)
```

A packet is to be transmitted to a specific IPv6 destination on a specific interface. Check if we can find the hardware address of the next hop to use for the packet. If so, give the hardware address to the caller, which should use it to send the packet right away. Otherwise, enqueue the packet for later transmission while looking up the hardware address, if possible.

As such, this function returns one of three different possible results:

- ERR_OK with a non-NULL 'hwaddrp': the caller should send the packet now.
- ERR_OK with a NULL 'hwaddrp': the packet has been enqueued for later.
- not ERR_OK: something went wrong; forward the error upward in the stack.

Parameters

netif	The lwIP network interface on which the IP packet will be sent.
q	The pbuf(s) containing the IP packet to be sent.
ip6addr	The destination IPv6 address of the packet.
hwaddrp	On success, filled with a pointer to a HW address or NULL (meaning the packet has been queued).

Returns

- ERR_OK on success, ERR_RTE if no route was found for the packet, or ERR_MEM if low memory conditions prohibit sending the packet at all.

4.75.3.7 nd6_input()

```
void nd6_input (struct pbuf * p, struct netif * inp)
```

Process an incoming neighbor discovery message

Parameters

p	the nd packet, p->payload pointing to the icmpv6 header
inp	the netif on which this packet was received

4.75.3.8 nd6_reachability_hint()

```
void nd6_reachability_hint (const ip6_addr_t * ip6addr)
```

Provide the Neighbor discovery process with a hint that a destination is reachable. Called by tcp_receive when ACKs are received or sent (as per RFC). This is useful to avoid sending NS messages every 30 seconds.

Parameters

ip6addr	the destination address which is know to be reachable by an upper layer protocol (TCP)

4.75.3.9 nd6_restart_netif()

```
void nd6_restart_netif (struct netif * netif)
```

Netif was added, set up, or reconnected (link up)

4.75.3.10 nd6_tmr()

```
void nd6_tmr (void )
```

Periodic timer for Neighbor discovery functions:

- Update neighbor reachability states
- Update destination cache entries age
- Update invalidation timers of default routers and on-link prefixes
- Update lifetimes of our addresses
- Perform duplicate address detection (DAD) for our addresses
- Send router solicitations

4.76 src/core/mem.c File Reference

```
#include "lwip/opt.h"#include "lwip/mem.h"#include "lwip/def.h"#include "lwip/sys.h"# ↩
    include "lwip/stats.h"#include "lwip/err.h"#include <stdio.h>#include <string.h>
```

4.76.1 Data Structures

- struct mem

4.76.2 Macros

- #define MIN_SIZE 12

4.76.3 Functions

- void mem_init (void)
- void mem_free (void *rmem)
- void * mem_trim (void *rmem, mem_size_t new_size)
- void * mem_malloc (mem_size_t size_in)
- void * mem_calloc (mem_size_t count, mem_size_t size)

4.76.4 Variables

- u8_t ram_heap [((((((1600)+1 - 1U) &~(1 -1U))+(2U *(((sizeof(struct mem))+1 - 1U) &~(1 -1U))))+1 - 1U))]

4.76.5 Detailed Description

Dynamic memory manager

This is a lightweight replacement for the standard C library malloc().

If you want to use the standard C library malloc() instead, define MEM_LIBC_MALLOC to 1 in your lwipopts.h

To let mem_malloc() use pools (prevents fragmentation and is much faster than a heap but might waste some memory), define MEM_USE_POOLS to 1, define MEMP_USE_CUSTOM_POOLS to 1 and create a file "lwippools.h" that includes a list of pools like this (more pools can be added between _START and _END):

Define three pools with sizes 256, 512, and 1512 bytes LWIP_MALLOC_MEMPOOL_START LWIP_MALLOC_MEMPOOL(20, 256) LWIP_MALLOC_MEMPOOL(10, 512) LWIP_MALLOC_MEMPOOL(5, 1512) LWIP_MALLOC_MEMPOOL_END

4.76.6 Macro Definition Documentation

4.76.6.1 MIN_SIZE

```
#define MIN_SIZE   12
```

All allocated blocks will be MIN_SIZE bytes big, at least! MIN_SIZE can be overridden to suit your needs. Smaller values save space, larger values could prevent too small blocks to fragment the RAM too much.

4.76.7 Function Documentation

4.76.7.1 mem_calloc()

```
void * mem_calloc (mem_size_t count, mem_size_t size)
```

Contiguously allocates enough space for count objects that are size bytes of memory each and returns a pointer to the allocated memory.

The allocated memory is filled with bytes of value zero.

Parameters

count	number of objects to allocate
size	size of the objects to allocate

Returns pointer to allocated memory / NULL pointer if there is an error

4.76.7.2 mem_free()

```
void mem_free (void * rmem)
```

Put a struct mem back on the heap

Parameters

rmem	is the data portion of a struct mem as returned by a previous call to mem_malloc()

4.76.7.3 mem_init()

```
void mem_init (void )
```

Zero the heap and initialize start, end and lowest-free

4.76.7.4 mem_malloc()

```
void * mem_malloc (mem_size_t size_in)
```

Allocate a block of memory with a minimum of 'size' bytes.

Parameters

size_in	is the minimum size of the requested block in bytes.

Returns pointer to allocated memory or NULL if no free memory was found.

Note that the returned value will always be aligned (as defined by MEM_ALIGNMENT).

4.76.7.5 mem_trim()

```
void * mem_trim (void * rmem, mem_size_t new_size)
```

Shrink memory returned by mem_malloc().

Parameters

rmem	pointer to memory allocated by mem_malloc the is to be shrunk
new_size	required size after shrinking (needs to be smaller than or equal to the previous size)

Returns for compatibility reasons: is always == rmem, at the moment or NULL if newsize is > old size, in which case rmem is NOT touched or freed!

4.76.8 Variable Documentation

4.76.8.1 ram_heap

```
u8_t ram_heap[((((((( 1600)+ 1 - 1U) &~(1 -1U))+(2U *(((sizeof(struct mem))+ 1 - 1U) &~(1
-1U))))+ 1 - 1U))]
```

If you want to relocate the heap to external memory, simply define LWIP_RAM_HEAP_POINTER as a void-pointer to that location. If so, make sure the memory at that location is big enough (see below on how that space is calculated). the heap. we need one struct mem at the end and some room for alignment

4.77 src/core/memp.c File Reference

```
#include "lwip/opt.h"#include "lwip/memp.h"#include "lwip/sys.h"#include "lwip/stats.h"# ↩
    include <string.h>#include "lwip/pbuf.h"#include "lwip/raw.h"#include "lwip/udp.h"# ↩
    include "lwip/tcp.h"#include "lwip/priv/tcp_priv.h"#include "lwip/altcp.h"#include "lwip ↩
    /ip4_frag.h"#include "lwip/netbuf.h"#include "lwip/api.h"#include "lwip/priv/tcpip_priv. ↩
    h"#include "lwip/priv/api_msg.h"#include "lwip/priv/sockets_priv.h"#include "lwip/etharp ↩
    .h"#include "lwip/igmp.h"#include "lwip/timeouts.h"#include "netif/ppp/ppp_opts.h"# ↩
    include "lwip/netdb.h"#include "lwip/dns.h"#include "lwip/priv/nd6_priv.h"#include "lwip ↩
    /ip6_frag.h"#include "lwip/mld6.h"#include "lwip/priv/memp_std.h"#include "path/to/my/ ↩
    lwip_hooks.h"
```

4.77.1 Functions

- void memp_init_pool (const struct memp_desc *desc)
- void memp_init (void)
- void * memp_malloc_pool (const struct memp_desc *desc)
- void * memp_malloc (memp_t type)
- void memp_free_pool (const struct memp_desc *desc, void *mem)
- void memp_free (memp_t type, void *mem)

4.77.2 Detailed Description

Dynamic pool memory manager

lwIP has dedicated pools for many structures (netconn, protocol control blocks, packet buffers, ...). All these pools are managed here.

4.77.3 Function Documentation

4.77.3.1 memp_free()

```
void memp_free (memp_t type, void * mem)
```

Put an element back into its pool.

Parameters

type	the pool where to put mem
mem	the memp element to free

4.77.3.2 memp_free_pool()

```
void memp_free_pool (const struct memp_desc * desc, void * mem)
```

Put a custom pool element back into its pool.

Parameters

desc	the pool where to put mem
mem	the memp element to free

4.77.3.3 memp_init()

```
void memp_init (void )
```

Initializes lwIP built-in pools. Related functions: memp_malloc, memp_free

Carves out memp_memory into linked lists for each pool-type.

desc	pool to initialize

4.77.3.4 memp_init_pool()

`void memp_init_pool (const struct memp_desc * desc)`

Initialize custom memory pool. Related functions: memp_malloc_pool, memp_free_pool

Parameters

4.77.3.5 memp_malloc()

`void * memp_malloc (memp_t type)`

Get an element from a specific pool.

Parameters

type	the pool to get an element from

Returns a pointer to the allocated memory or a NULL pointer on error

4.77.3.6 memp_malloc_pool()

`void * memp_malloc_pool (const struct memp_desc * desc)`

Get an element from a custom pool.

Parameters

desc	the pool to get an element from

Returns a pointer to the allocated memory or a NULL pointer on error

4.78 src/core/netif.c File Reference

```
#include "lwip/opt.h"#include <string.h>#include <stdlib.h>#include "lwip/def.h"#include " ↩
    lwip/ip_addr.h"#include "lwip/ip6_addr.h"#include "lwip/netif.h"#include "lwip/priv/ ↩
    tcp_priv.h"#include "lwip/udp.h"#include "lwip/priv/raw_priv.h"#include "lwip/snmp.h"# ↩
    include "lwip/igmp.h"#include "lwip/etharp.h"#include "lwip/stats.h"#include "lwip/sys.h ↩
    "#include "lwip/ip.h"#include "lwip/tcpip.h"#include "netif/ethernet.h"#include "lwip/ ↩
    autoip.h"#include "lwip/dhcp.h"#include "lwip/acd.h"#include "lwip/dhcp6.h"#include " ↩
    lwip/mld6.h"#include "lwip/nd6.h"
```

4.78.1 Functions

- err_t netif_input (struct pbuf *p, struct netif *inp)
- struct netif * netif_add_noaddr (struct netif *netif, void *state, netif_init_fn init, netif_input_fn input)
- struct netif * netif_add (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw, void *state, netif_init_fn init, netif_input_fn input)

- void netif_set_ipaddr (struct netif *netif, const ip4_addr_t *ipaddr)
- void netif_set_netmask (struct netif *netif, const ip4_addr_t *netmask)
- void netif_set_gw (struct netif *netif, const ip4_addr_t *gw)
- void netif_set_addr (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw)
- void netif_remove (struct netif *netif)
- void netif_set_default (struct netif *netif)
- void netif_set_up (struct netif *netif)
- void netif_set_down (struct netif *netif)
- void netif_set_status_callback (struct netif *netif, netif_status_callback_fn status_callback)
- void netif_set_remove_callback (struct netif *netif, netif_status_callback_fn remove_callback)
- void netif_set_link_up (struct netif *netif)
- void netif_set_link_down (struct netif *netif)
- void netif_set_link_callback (struct netif *netif, netif_status_callback_fn link_callback)
- err_t netif_loop_output (struct netif *netif, struct pbuf *p)
- void netif_poll (struct netif *netif)
- u8_t netif_alloc_client_data_id (void)
- void netif_ip6_addr_set (struct netif *netif, s8_t addr_idx, const ip6_addr_t *addr6)
- void netif_ip6_addr_set_state (struct netif *netif, s8_t addr_idx, u8_t state)
- s8_t netif_get_ip6_addr_match (struct netif *netif, const ip6_addr_t *ip6addr)
- void netif_create_ip6_linklocal_address (struct netif *netif, u8_t from_mac_48bit)
- err_t netif_add_ip6_address (struct netif *netif, const ip6_addr_t *ip6addr, s8_t *chosen_idx)
- u8_t netif_name_to_index (const char *name)
- char * netif_index_to_name (u8_t idx, char *name)
- struct netif * netif_get_by_index (u8_t idx)
- struct netif * netif_find (const char *name)
- void netif_add_ext_callback (netif_ext_callback_t *callback, netif_ext_callback_fn fn)
- void netif_remove_ext_callback (netif_ext_callback_t *callback)
- void netif_invoke_ext_callback (struct netif *netif, netif_nsc_reason_t reason, const netif_ext_callback_args_t *args)

4.78.2 Variables

- struct netif * netif_list
- struct netif * netif_default

4.78.3 Detailed Description

lwIP network interface abstraction

4.78.4 Function Documentation

4.78.4.1 netif_get_ip6_addr_match()

```
s8_t netif_get_ip6_addr_match (struct netif * netif, const ip6_addr_t * ip6addr)
```

Checks if a specific local address is present on the netif and returns its index. Depending on its state, it may or may not be assigned to the interface (as per RFC terminology).

The given address may or may not be zoned (i.e., have a zone index other than IP6_NO_ZONE). If the address is zoned, it must have the correct zone for the given netif, or no match will be found.

Parameters

netif	the netif to check
ip6addr	the IPv6 address to find

Returns >= 0: address found, this is its index -1: address not found on this netif

4.78.4.2 netif_invoke_ext_callback()

```
void netif_invoke_ext_callback (struct netif * netif, netif_nsc_reason_t reason, const
netif_ext_callback_args_t * args)
```

Invoke extended netif status event

Parameters

netif	netif that is affected by change
reason	change reason
args	depends on reason, see reason description

4.78.4.3 netif_poll()

```
void netif_poll (struct netif * netif)
```

Call netif_poll() in the main loop of your application. This is to prevent reentering non-reentrant functions like tcp_input(). Packets passed to netif_loop_output() are put on a list that is passed to netif->input() by netif_poll().

4.78.5 Variable Documentation

4.78.5.1 netif_default

```
struct netif* netif_default
```

The default network interface.

4.78.5.2 netif_list

```
struct netif* netif_list
```

The list of network interfaces.

4.79 src/core/pbuf.c File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/stats.h"#include "lwip/def.h"# ↩
    include "lwip/mem.h"#include "lwip/memp.h"#include "lwip/sys.h"#include "lwip/netif.h"# ↩
    include "lwip/priv/tcp_priv.h"#include <string.h>#include "lwip/tcpip.h"
```

4.79.1 Functions

- struct pbuf * pbuf_alloc (pbuf_layer layer, u16_t length, pbuf_type type)
- struct pbuf * pbuf_alloc_reference (void *payload, u16_t length, pbuf_type type)
- struct pbuf * pbuf_alloced_custom (pbuf_layer l, u16_t length, pbuf_type type, struct pbuf_custom *p, void *payload_mem, u16_t payload_mem_len)
- void pbuf_realloc (struct pbuf *p, u16_t new_len)
- u8_t pbuf_add_header (struct pbuf *p, size_t header_size_increment)
- u8_t pbuf_add_header_force (struct pbuf *p, size_t header_size_increment)
- u8_t pbuf_remove_header (struct pbuf *p, size_t header_size_decrement)
- u8_t pbuf_header (struct pbuf *p, s16_t header_size_increment)
- u8_t pbuf_header_force (struct pbuf *p, s16_t header_size_increment)
- struct pbuf * pbuf_free_header (struct pbuf *q, u16_t size)
- u8_t pbuf_free (struct pbuf *p)
- u16_t pbuf_clen (const struct pbuf *p)
- void pbuf_ref (struct pbuf *p)
- void pbuf_cat (struct pbuf *h, struct pbuf *t)
- void pbuf_chain (struct pbuf *h, struct pbuf *t)
- struct pbuf * pbuf_dechain (struct pbuf *p)
- err_t pbuf_copy (struct pbuf *p_to, const struct pbuf *p_from)
- err_t pbuf_copy_partial_pbuf (struct pbuf *p_to, const struct pbuf *p_from, u16_t copy_len, u16_t offset)
- u16_t pbuf_copy_partial (const struct pbuf *buf, void *dataptr, u16_t len, u16_t offset)
- void * pbuf_get_contiguous (const struct pbuf *p, void *buffer, size_t bufsize, u16_t len, u16_t offset)
- struct pbuf * pbuf_skip (struct pbuf *in, u16_t in_offset, u16_t *out_offset)
- err_t pbuf_take (struct pbuf *buf, const void *dataptr, u16_t len)
- err_t pbuf_take_at (struct pbuf *buf, const void *dataptr, u16_t len, u16_t offset)
- struct pbuf * pbuf_coalesce (struct pbuf *p, pbuf_layer layer)
- struct pbuf * pbuf_clone (pbuf_layer layer, pbuf_type type, struct pbuf *p)
- u8_t pbuf_get_at (const struct pbuf *p, u16_t offset)
- int pbuf_try_get_at (const struct pbuf *p, u16_t offset)
- void pbuf_put_at (struct pbuf *p, u16_t offset, u8_t data)
- u16_t pbuf_memcmp (const struct pbuf *p, u16_t offset, const void *s2, u16_t n)
- u16_t pbuf_memfind (const struct pbuf *p, const void *mem, u16_t mem_len, u16_t start_offset)
- u16_t pbuf_strstr (const struct pbuf *p, const char *substr)

4.79.2 Detailed Description

Packet buffer management

4.79.3 Function Documentation

4.79.3.1 pbuf_add_header()

`u8_t pbuf_add_header (struct pbuf * p, size_t header_size_increment)`

Adjusts the payload pointer to reveal headers in the payload.

Adjusts the ->payload pointer so that space for a header appears in the pbuf payload.

The ->payload, ->tot_len and ->len fields are adjusted.

Parameters

p	pbuf to change the header size.
header_size_increment	Number of bytes to increment header size which increases the size of the pbuf. New space is on the front. If header_size_increment is 0, this function does nothing and returns successful.

PBUF_ROM and PBUF_REF type buffers cannot have their sizes increased, so the call will fail. A check is made that the increase in header size does not move the payload pointer in front of the start of the buffer.

Returns non-zero on failure, zero on success.

4.79.3.2 pbuf_add_header_force()

`u8_t pbuf_add_header_force (struct pbuf * p, size_t header_size_increment)`

Same as pbuf_add_header but does not check if 'header_size > 0' is allowed. This is used internally only, to allow PBUF_REF for RX.

4.79.3.3 pbuf_clen()

`u16_t pbuf_clen (const struct pbuf * p)`

Count number of pbufs in a chain

Parameters

p	first pbuf of chain

Returns the number of pbufs in a chain

4.79.3.4 pbuf_dechain()

`struct pbuf * pbuf_dechain (struct pbuf * p)`

Dechains the first pbuf from its succeeding pbufs in the chain.

Makes p->tot_len field equal to p->len.

Parameters

Returns remainder of the pbuf chain, or NULL if it was de-allocated.

Note
May not be called on a packet queue.

p	pbuf to dechain

4.79.3.5 pbuf_free_header()

```
struct pbuf * pbuf_free_header (struct pbuf * q, u16_t size)
```

Similar to pbuf_header(-size) but de-refs header pbufs for (size >= p->len)

Parameters

q	pbufs to operate on
size	The number of bytes to remove from the beginning of the pbuf list. While size >= p->len, pbufs are freed. ATTENTION: this is the opposite direction as pbuf_header, but takes an u16_t not s16_t!

Returns the new head pbuf

4.79.3.6 pbuf_header()

```
u8_t pbuf_header (struct pbuf * p, s16_t header_size_increment)
```

Adjusts the payload pointer to hide or reveal headers in the payload.

Adjusts the ->payload pointer so that space for a header (dis)appears in the pbuf payload.

The ->payload, ->tot_len and ->len fields are adjusted.

Parameters

p	pbuf to change the header size.
header_size_increment	Number of bytes to increment header size which increases the size of the pbuf. New space is on the front. (Using a negative value decreases the header size.) If header_size_increment is 0, this function does nothing and returns successful.

PBUF_ROM and PBUF_REF type buffers cannot have their sizes increased, so the call will fail. A check is made that the increase in header size does not move the payload pointer in front of the start of the buffer. **Returns** non-zero on failure, zero on success.

4.79.3.7 pbuf_header_force()

```
u8_t pbuf_header_force (struct pbuf * p, s16_t header_size_increment)
```

Same as pbuf_header but does not check if 'header_size > 0' is allowed. This is used internally only, to allow PBUF_REF for RX.

4.79.3.8 pbuf_remove_header()

```
u8_t pbuf_remove_header (struct pbuf * p, size_t header_size_decrement)
```

Adjusts the payload pointer to hide headers in the payload.

Adjusts the ->payload pointer so that space for a header disappears in the pbuf payload.

The ->payload, ->tot_len and ->len fields are adjusted.

Parameters

Returns non-zero on failure, zero on success.

p	pbuf to change the header size.
header_size_decrement	Number of bytes to decrement header size which decreases the size of the pbuf. If header_size_decrement is 0, this function does nothing and returns successful.

4.79.3.9 pbuf_strstr()

```
u16_t pbuf_strstr (const struct pbuf * p, const char * substr)
```

Find occurrence of substr with length substr_len in pbuf p, start at offset start_offset WARNING: in contrast to strstr(), this one does not stop at the first \0 in the pbuf/source string!

Parameters

p	pbuf to search, maximum length is 0xFFFE since 0xFFFF is used as return value 'not found'
substr	string to search for in p, maximum length is 0xFFFE

Returns 0xFFFF if substr was not found in p or the index where it was found

4.80 src/core/raw.c File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/memp.h"#include "lwip/ip_addr.h"# ↩
    include "lwip/netif.h"#include "lwip/raw.h"#include "lwip/priv/raw_priv.h"#include "lwip ↩
    /stats.h"#include "lwip/ip6.h"#include "lwip/ip6_addr.h"#include "lwip/inet_chksum.h"# ↩
    include <string.h>
```

4.80.1 Functions

- raw_input_state_t raw_input (struct pbuf *p, struct netif *inp)
- err_t raw_bind (struct raw_pcb *pcb, const ip_addr_t *ipaddr)
- void raw_bind_netif (struct raw_pcb *pcb, const struct netif *netif)
- err_t raw_connect (struct raw_pcb *pcb, const ip_addr_t *ipaddr)
- void raw_disconnect (struct raw_pcb *pcb)
- void raw_recv (struct raw_pcb *pcb, raw_recv_fn recv, void *recv_arg)
- err_t raw_sendto (struct raw_pcb *pcb, struct pbuf *p, const ip_addr_t *ipaddr)
- err_t raw_sendto_if_src (struct raw_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, struct netif *netif, const ip_addr_t *src_ip)
- err_t raw_send (struct raw_pcb *pcb, struct pbuf *p)
- void raw_remove (struct raw_pcb *pcb)
- struct raw_pcb * raw_new (u8_t proto)
- struct raw_pcb * raw_new_ip_type (u8_t type, u8_t proto)
- void raw_netif_ip_addr_changed (const ip_addr_t *old_addr, const ip_addr_t *new_addr)

4.80.2 Detailed Description

Implementation of raw protocol PCBs for low-level handling of different types of protocols besides (or overriding) those already available in lwIP. See also RAW

4.80.3 Function Documentation

4.80.3.1 raw_input()

```
raw_input_state_t raw_input (struct pbuf * p, struct netif * inp)
```

Determine if in incoming IP packet is covered by a RAW PCB and if so, pass it to a user-provided receive callback function.

Given an incoming IP datagram (as a chain of pbufs) this function finds a corresponding RAW PCB and calls the corresponding receive callback function.

Parameters

p	pbuf to be demultiplexed to a RAW PCB.
inp	network interface on which the datagram was received.

Returns - 1 if the packet has been eaten by a RAW PCB receive callback function. The caller MAY NOT not reference the packet any longer, and MAY NOT call pbuf_free().

- 0 if packet is not eaten (pbuf is still referenced by the caller).

4.80.3.2 raw_netif_ip_addr_changed()

```
void raw_netif_ip_addr_changed (const ip_addr_t * old_addr, const ip_addr_t * new_addr)
```

This function is called from netif.c when address is changed

Parameters

old_addr	IP address of the netif before change
new_addr	IP address of the netif after change

4.81 src/core/stats.c File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/stats.h"#include "lwip/mem.h"# ↩
    include "lwip/debug.h"#include <string.h>
```

4.81.1 Functions

- void stats_init (void)

4.81.2 Variables

- struct stats_ lwip_stats

4.81.3 Detailed Description

Statistics module

4.81.4 Function Documentation

4.81.4.1 stats_init()

```
void stats_init (void )
```

Init statistics

4.81.5 Variable Documentation

4.81.5.1 lwip_stats

```
struct stats_ lwip_stats
```

Global variable containing lwIP internal statistics. Add this to your debugger's watchlist.

4.82 src/core/sys.c File Reference

```
#include "lwip/opt.h"#include "lwip/sys.h"
```

4.82.1 Functions

- void sys_msleep (u32_t ms)

4.82.2 Detailed Description

lwIP Operating System abstraction

4.83 src/core/tcp.c File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/mem.h"#include "lwip/memp.h"# ↩
    include "lwip/tcp.h"#include "lwip/priv/tcp_priv.h"#include "lwip/debug.h"#include "lwip ↩
    /stats.h"#include "lwip/ip6.h"#include "lwip/ip6_addr.h"#include "lwip/nd6.h"#include < ↩
    string.h>#include "path/to/my/lwip_hooks.h"
```

4.83.1 Functions

- void tcp_init (void)
- void tcp_free (struct tcp_pcb *pcb)
- void tcp_tmr (void)
- void tcp_backlog_delayed (struct tcp_pcb *pcb)
- void tcp_backlog_accepted (struct tcp_pcb *pcb)

- err_t tcp_close (struct tcp_pcb *pcb)
- err_t tcp_shutdown (struct tcp_pcb *pcb, int shut_rx, int shut_tx)
- void tcp_abandon (struct tcp_pcb *pcb, int reset)
- void tcp_abort (struct tcp_pcb *pcb)
- err_t tcp_bind (struct tcp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port)
- void tcp_bind_netif (struct tcp_pcb *pcb, const struct netif *netif)
- struct tcp_pcb * tcp_listen_with_backlog (struct tcp_pcb *pcb, u8_t backlog)
- struct tcp_pcb * tcp_listen_with_backlog_and_err (struct tcp_pcb *pcb, u8_t backlog, err_t *err)
- u32_t tcp_update_rcv_ann_wnd (struct tcp_pcb *pcb)
- void tcp_recved (struct tcp_pcb *pcb, u16_t len)
- err_t tcp_connect (struct tcp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port, tcp_connected_fn connected)
- void tcp_slowtmr (void)
- void tcp_fasttmr (void)
- void tcp_txnow (void)
- err_t tcp_process_refused_data (struct tcp_pcb *pcb)
- void tcp_segs_free (struct tcp_seg *seg)
- void tcp_seg_free (struct tcp_seg *seg)
- void tcp_setprio (struct tcp_pcb *pcb, u8_t prio)
- struct tcp_seg * tcp_seg_copy (struct tcp_seg *seg)
- err_t tcp_recv_null (void *arg, struct tcp_pcb *pcb, struct pbuf *p, err_t err)
- struct tcp_pcb * tcp_alloc (u8_t prio)
- struct tcp_pcb * tcp_new (void)
- struct tcp_pcb * tcp_new_ip_type (u8_t type)
- void tcp_arg (struct tcp_pcb *pcb, void *arg)
- void tcp_recv (struct tcp_pcb *pcb, tcp_recv_fn recv)
- void tcp_sent (struct tcp_pcb *pcb, tcp_sent_fn sent)
- void tcp_err (struct tcp_pcb *pcb, tcp_err_fn err)
- void tcp_accept (struct tcp_pcb *pcb, tcp_accept_fn accept)
- void tcp_poll (struct tcp_pcb *pcb, tcp_poll_fn poll, u8_t interval)
- void tcp_pcb_purge (struct tcp_pcb *pcb)
- void tcp_pcb_remove (struct tcp_pcb **pcblist, struct tcp_pcb *pcb)
- u32_t tcp_next_iss (struct tcp_pcb *pcb)
- u16_t tcp_eff_send_mss_netif (u16_t sendmss, struct netif *outif, const ip_addr_t *dest)
- void tcp_netif_ip_addr_changed (const ip_addr_t *old_addr, const ip_addr_t *new_addr)
- u8_t tcp_ext_arg_alloc_id (void)

- void tcp_ext_arg_set_callbacks (struct tcp_pcb *pcb, u8_t id, const struct tcp_ext_arg_callbacks *const callbacks)
- void tcp_ext_arg_set (struct tcp_pcb *pcb, u8_t id, void *arg)
- void * tcp_ext_arg_get (const struct tcp_pcb *pcb, u8_t id)
- err_t tcp_ext_arg_invoke_callbacks_passive_open (struct tcp_pcb_listen *lpcb, struct tcp_pcb *cpcb)

4.83.2 Variables

- struct tcp_pcb * tcp_bound_pcbs
- union tcp_listen_pcbs_t tcp_listen_pcbs
- struct tcp_pcb * tcp_active_pcbs
- struct tcp_pcb * tcp_tw_pcbs
- struct tcp_pcb **const tcp_pcb_lists []

4.83.3 Detailed Description

Transmission Control Protocol for IP See also TCP

4.83.4 Function Documentation

4.83.4.1 tcp_abandon()

```
void tcp_abandon (struct tcp_pcb * pcb, int reset)
```

Abandons a connection and optionally sends a RST to the remote host. Deletes the local protocol control block. This is done when a connection is killed because of shortage of memory.

Parameters

pcb	the tcp_pcb to abort
reset	boolean to indicate whether a reset should be sent

4.83.4.2 tcp_alloc()

```
struct tcp_pcb * tcp_alloc (u8_t prio)
```

Allocate a new tcp_pcb structure.

Parameters

prio	priority for the new pcb

Returns a new tcp_pcb that initially is in state CLOSED

4.83.4.3 tcp_eff_send_mss_netif()

```
u16_t tcp_eff_send_mss_netif (u16_t sendmss, struct netif * outif, const ip_addr_t * dest)
```

Calculates the effective send mss that can be used for a specific IP address by calculating the minimum of TCP_MSS and the mtu (if set) of the target netif (if not NULL).

4.83.4.4 tcp_ext_arg_invoke_callbacks_passive_open()

```
err_t tcp_ext_arg_invoke_callbacks_passive_open (struct tcp_pcb_listen * lpcb, struct tcp_
* cpcb)
```

This function calls the "passive_open" callback for all ext_args if a connection is in the process of being accepted. This is called just after the SYN is received and before a SYN/ACK is sent, to allow to modify the very first segment sent even on passive open. Naturally, the "accepted" callback of the pcb has not been called yet!

4.83.4.5 tcp_fasttmr()

```
void tcp_fasttmr (void )
```

Is called every TCP_FAST_INTERVAL (250 ms) and process data previously "refused" by upper layer (application) and sends delayed ACKs or pending FINs.

Automatically called from tcp_tmr().

4.83.4.6 tcp_free()

```
void tcp_free (struct tcp_pcb * pcb)
```

Free a tcp pcb

4.83.4.7 tcp_init()

```
void tcp_init (void )
```

Initialize this module.

4.83.4.8 tcp_netif_ip_addr_changed()

```
void tcp_netif_ip_addr_changed (const ip_addr_t * old_addr, const ip_addr_t * new_addr)
```

This function is called from netif.c when address is changed or netif is removed

Parameters

old_addr	IP address of the netif before change
new_addr	IP address of the netif after change or NULL if netif has been removed

4.83.4.9 tcp_next_iss()

```
u32_t tcp_next_iss (struct tcp_pcb * pcb)
```

Calculates a new initial sequence number for new connections.

Returns u32_t pseudo random sequence number

4.83.4.10 tcp_pcb_purge()

```
void tcp_pcb_purge (struct tcp_pcb * pcb)
```

Purges a TCP PCB. Removes any buffered data and frees the buffer memory (pcb->ooseq, pcb->unsent and pcb->unacked are freed).

Parameters

pcb	tcp_pcb to purge. The pcb itself is not deallocated!

4.83.4.11 tcp_pcb_remove()

`void tcp_pcb_remove (struct tcp_pcb ** pcblist, struct tcp_pcb * pcb)`

Purges the PCB and removes it from a PCB list. Any delayed ACKs are sent first.

Parameters

pcblist	PCB list to purge.
pcb	tcp_pcb to purge. The pcb itself is NOT deallocated!

4.83.4.12 tcp_process_refused_data()

`err_t tcp_process_refused_data (struct tcp_pcb * pcb)`

Pass pcb->refused_data to the recv callback

4.83.4.13 tcp_recv_null()

`err_t tcp_recv_null (void * arg, struct tcp_pcb * pcb, struct pbuf * p, err_t err)`

Default receive callback that is called if the user didn't register a recv callback for the pcb.

4.83.4.14 tcp_seg_copy()

`struct tcp_seg * tcp_seg_copy (struct tcp_seg * seg)`

Returns a copy of the given TCP segment. The pbuf and data are not copied, only the pointers

Parameters

seg	the old tcp_seg

Returns a copy of seg

4.83.4.15 tcp_seg_free()

`void tcp_seg_free (struct tcp_seg * seg)`

Frees a TCP segment (tcp_seg structure).

Parameters

4.83.4.16 tcp_segs_free()

`void tcp_segs_free (struct tcp_seg * seg)`

Deallocates a list of TCP segments (tcp_seg structures).

Parameters

seg	single tcp_seg to free

seg	tcp_seg list of TCP segments to free

4.83.4.17 tcp_setprio()

```
void tcp_setprio (struct tcp_pcb * pcb, u8_t prio)
```

Sets the priority of a connection.

Parameters

pcb	the tcp_pcb to manipulate
prio	new priority

4.83.4.18 tcp_slowtmr()

```
void tcp_slowtmr (void )
```

Called every 500 ms and implements the retransmission timer and the timer that removes PCBs that have been in TIME-WAIT for enough time. It also increments various timers such as the inactivity timer in each PCB.

Automatically called from tcp_tmr().

4.83.4.19 tcp_tmr()

```
void tcp_tmr (void )
```

Called periodically to dispatch TCP timers.

4.83.4.20 tcp_txnow()

```
void tcp_txnow (void )
```

Call tcp_output for all active pcbs that have TF_NAGLEMEMERR set

4.83.4.21 tcp_update_rcv_ann_wnd()

```
u32_t tcp_update_rcv_ann_wnd (struct tcp_pcb * pcb)
```

Update the state that tracks the available window space to advertise.

Returns how much extra window would be advertised if we sent an update now.

4.83.5 Variable Documentation

4.83.5.1 tcp_active_pcbs

```
struct tcp_pcb* tcp_active_pcbs
```

List of all TCP PCBs that are in a state in which they accept or send data.

4.83.5.2 tcp_bound_pcbs

`struct tcp_pcb* tcp_bound_pcbs`

List of all TCP PCBs bound but not yet (connected || listening)

4.83.5.3 tcp_listen_pcbs

`union tcp_listen_pcbs_t tcp_listen_pcbs`

List of all TCP PCBs in LISTEN state

4.83.5.4 tcp_pcb_lists

`struct tcp_pcb** const tcp_pcb_lists[]` **Initial value:**

```
= {&tcp_listen_pcbs.pcbs, &tcp_bound_pcbs,
         &tcp_active_pcbs, &tcp_tw_pcbs
}
```

An array with all (non-temporary) PCB lists, mainly used for smaller code size

4.83.5.5 tcp_tw_pcbs

`struct tcp_pcb* tcp_tw_pcbs`

List of all TCP PCBs in TIME-WAIT state

4.84 src/core/tcp_in.c File Reference

```
#include "lwip/opt.h"#include "lwip/priv/tcp_priv.h"#include "lwip/def.h"#include "lwip/ ↩
    ip_addr.h"#include "lwip/netif.h"#include "lwip/mem.h"#include "lwip/memp.h"#include " ↩
    lwip/inet_chksum.h"#include "lwip/stats.h"#include "lwip/ip6.h"#include "lwip/ip6_addr.h ↩
    "#include "lwip/nd6.h"#include <string.h>#include "path/to/my/lwip_hooks.h"
```

4.84.1 Macros

- #define LWIP_TCP_CALC_INITIAL_CWND(mss) ((tcpwnd_size_t)LWIP_MIN((4U * (mss)), LWIP_MAX((2U * (mss)), 4380U)))

4.84.2 Functions

- void tcp_input (struct pbuf *p, struct netif *inp)

4.84.3 Detailed Description

Transmission Control Protocol, incoming traffic

The input processing functions of the TCP layer.

These functions are generally called in the order (ip_input() ->) tcp_input() -> * tcp_process() -> tcp_receive() (-> application).

4.84.4 Macro Definition Documentation

4.84.4.1 LWIP_TCP_CALC_INITIAL_CWND

```
#define LWIP_TCP_CALC_INITIAL_CWND( mss)    ((tcpwnd_size_t)LWIP_MIN((4U * (mss)), LWIP_MAX
* (mss)), 4380U)))
```

Initial CWND calculation as defined RFC 2581

4.84.5 Function Documentation

4.84.5.1 tcp_input()

```
void tcp_input (struct pbuf * p, struct netif * inp)
```

The initial input processing of TCP. It verifies the TCP header, demultiplexes the segment between the PCBs and passes it on to tcp_process(), which implements the TCP finite state machine. This function is called by the IP layer (in ip_input()).

Parameters

p	received TCP segment to process (p->payload pointing to the TCP header)
inp	network interface on which this segment was received

4.85 src/core/tcp_out.c File Reference

```
#include "lwip/opt.h"#include "lwip/priv/tcp_priv.h"#include "lwip/def.h"#include "lwip/mem ↩
    .h"#include "lwip/memp.h"#include "lwip/ip_addr.h"#include "lwip/netif.h"#include "lwip/ ↩
    inet_chksum.h"#include "lwip/stats.h"#include "lwip/ip6.h"#include "lwip/ip6_addr.h"# ↩
    include <string.h>#include "path/to/my/lwip_hooks.h"
```

4.85.1 Macros

- #define TCP_CHECKSUM_ON_COPY_SANITY_CHECK   0
- #define TCP_OVERSIZE_CALC_LENGTH(length)   ((length) + TCP_OVERSIZE)

4.85.2 Functions

- err_t tcp_write (struct tcp_pcb *pcb, const void *arg, u16_t len, u8_t apiflags)
- err_t tcp_split_unsent_seg (struct tcp_pcb *pcb, u16_t split)
- err_t tcp_send_fin (struct tcp_pcb *pcb)
- err_t tcp_enqueue_flags (struct tcp_pcb *pcb, u8_t flags)
- err_t tcp_output (struct tcp_pcb *pcb)
- err_t tcp_rexmit_rto_prepare (struct tcp_pcb *pcb)
- void tcp_rexmit_rto_commit (struct tcp_pcb *pcb)
- void tcp_rexmit_rto (struct tcp_pcb *pcb)
- err_t tcp_rexmit (struct tcp_pcb *pcb)

- void tcp_rexmit_fast (struct tcp_pcb *pcb)
- void tcp_rst (const struct tcp_pcb *pcb, u32_t seqno, u32_t ackno, const ip_addr_t *local_ip, const ip_addr_t *remote_ip, u16_t local_port, u16_t remote_port)
- void tcp_rst_netif (struct netif *netif, u32_t seqno, u32_t ackno, const ip_addr_t *local_ip, const ip_addr_t *remote_ip, u16_t local_port, u16_t remote_port)
- err_t tcp_send_empty_ack (struct tcp_pcb *pcb)
- err_t tcp_keepalive (struct tcp_pcb *pcb)
- err_t tcp_zero_window_probe (struct tcp_pcb *pcb)

4.85.3 Detailed Description

Transmission Control Protocol, outgoing traffic

The output functions of TCP.

There are two distinct ways for TCP segments to get sent:

- queued data: these are segments transferring data or segments containing SYN or FIN (which both count as one sequence number). They are created as struct Packet buffers (PBUF) together with a struct tcp_seg and enqueue to the unsent list of the pcb. They are sent by tcp_output:
  - tcp_write : creates data segments
  - tcp_split_unsent_seg : splits a data segment
  - tcp_enqueue_flags : creates SYN-only or FIN-only segments
  - tcp_output / tcp_output_segment : finalize the tcp header (e.g. sequence numbers, options, checksum) and output to IP
  - the various tcp_rexmit functions shuffle around segments between the unsent an unacked lists to retransmit them
  - tcp_create_segment and tcp_pbuf_prealloc allocate pbuf and segment for these functions
- direct send: these segments don't contain data but control the connection behaviour. They are created as pbuf only and sent directly without enqueueing them:
  - tcp_send_empty_ack sends an ACK-only segment
  - tcp_rst sends a RST segment
  - tcp_keepalive sends a keepalive segment
  - tcp_zero_window_probe sends a window probe segment
  - tcp_output_alloc_header allocates a header-only pbuf for these functions

4.85.4 Macro Definition Documentation

4.85.4.1 TCP_CHECKSUM_ON_COPY_SANITY_CHECK

```
#define TCP_CHECKSUM_ON_COPY_SANITY_CHECK   0
```

Define this to 1 for an extra check that the output checksum is valid (useful when the checksum is generated by the application, not the stack)

4.85.4.2 TCP_OVERSIZE_CALC_LENGTH

```
#define TCP_OVERSIZE_CALC_LENGTH( length)    ((length) + TCP_OVERSIZE)
```

The size of segment pbufs created when TCP_OVERSIZE is enabled

4.85.5 Function Documentation

4.85.5.1 tcp_enqueue_flags()

`err_t tcp_enqueue_flags (struct tcp_pcb * pcb, u8_t flags)`

Enqueue SYN or FIN for transmission.

Called by tcp_connect, tcp_listen_input, and tcp_close (via tcp_send_fin)

Parameters

pcb	Protocol control block for the TCP connection.
flags	TCP header flags to set in the outgoing segment.

4.85.5.2 tcp_keepalive()

`err_t tcp_keepalive (struct tcp_pcb * pcb)`

Send keepalive packets to keep a connection active although no data is sent over it.

Called by tcp_slowtmr()

Parameters

pcb	the tcp_pcb for which to send a keepalive packet

4.85.5.3 tcp_rexmit()

`err_t tcp_rexmit (struct tcp_pcb * pcb)`

Requeue the first unacked segment for retransmission

Called by tcp_receive() for fast retransmit.

Parameters

pcb	the tcp_pcb for which to retransmit the first unacked segment

4.85.5.4 tcp_rexmit_fast()

`void tcp_rexmit_fast (struct tcp_pcb * pcb)`

Handle retransmission after three dupacks received

Parameters

4.85.5.5 tcp_rexmit_rto()

`void tcp_rexmit_rto (struct tcp_pcb * pcb)`

Requeue all unacked segments for retransmission

Called by tcp_process() only, tcp_slowtmr() needs to do some things between "prepare" and "commit".

Parameters

pcb	the tcp_pcb for which to retransmit the first unacked segment

pcb	the tcp_pcb for which to re-enqueue all unacked segments

4.85.5.6 tcp_rexmit_rto_commit()

```
void tcp_rexmit_rto_commit (struct tcp_pcb * pcb)
```

Requeue all unacked segments for retransmission

Called by tcp_slowtmr() for slow retransmission.

Parameters

pcb	the tcp_pcb for which to re-enqueue all unacked segments

4.85.5.7 tcp_rexmit_rto_prepare()

```
err_t tcp_rexmit_rto_prepare (struct tcp_pcb * pcb)
```

Requeue all unacked segments for retransmission

Called by tcp_slowtmr() for slow retransmission.

Parameters

4.85.5.8 tcp_rst()

```
void tcp_rst (const struct tcp_pcb * pcb, u32_t seqno, u32_t ackno, const ip_addr_t * loca
const ip_addr_t * remote_ip, u16_t local_port, u16_t remote_port)
```

Send a TCP RESET packet (empty segment with RST flag set) to abort a connection.

Called by tcp_abort() (to abort a local connection), tcp_closen() (if not all data has been received by the application), tcp_timewait_input(
(if a SYN is received) and tcp_process() (received segment in the wrong state).

Since a RST segment is in most cases not sent for an active connection, tcp_rst() has a number of arguments that are taken from a tcp_pcb for most other segment output functions.

Parameters

4.85.5.9 tcp_rst_netif()

```
void tcp_rst_netif (struct netif * netif, u32_t seqno, u32_t ackno, const ip_addr_t * loca
const ip_addr_t * remote_ip, u16_t local_port, u16_t remote_port)
```

Send a TCP RESET packet (empty segment with RST flag set) to show that there is no matching local connection for a received segment.

Called by tcp_input() (if no matching local pcb was found) and tcp_listen_input() (if incoming segment has ACK flag set).

Since a RST segment is in most cases not sent for an active connection, tcp_rst() has a number of arguments that are taken from a tcp_pcb for most other segment output functions.

Parameters

pcb	the tcp_pcb for which to re-enqueue all unacked segments

pcb	TCP pcb (may be NULL if no pcb is available)
seqno	the sequence number to use for the outgoing segment
ackno	the acknowledge number to use for the outgoing segment
local_ip	the local IP address to send the segment from
remote_ip	the remote IP address to send the segment to
local_port	the local TCP port to send the segment from
remote_port	the remote TCP port to send the segment to

4.85.5.10 tcp_send_empty_ack()

```
err_t tcp_send_empty_ack (struct tcp_pcb * pcb)
```

Send an ACK without data.

Parameters

4.85.5.11 tcp_send_fin()

```
err_t tcp_send_fin (struct tcp_pcb * pcb)
```

Called by tcp_close() to send a segment including FIN flag but not data. This FIN may be added to an existing segment or a new, otherwise empty segment is enqueued.

Parameters

Returns ERR_OK if sent, another err_t otherwise

4.85.5.12 tcp_split_unsent_seg()

```
err_t tcp_split_unsent_seg (struct tcp_pcb * pcb, u16_t split)
```

Split segment on the head of the unsent queue. If return is not ERR_OK, existing head remains intact

The split is accomplished by creating a new TCP segment and pbuf which holds the remainder payload after the split. The original pbuf is trimmed to new length. This allows splitting of read-only pbufs

Parameters

4.85.5.13 tcp_zero_window_probe()

```
err_t tcp_zero_window_probe (struct tcp_pcb * pcb)
```

Send persist timer zero-window probes to keep a connection active when a window update is lost.

Called by tcp_slowtmr()

Parameters

netif	the netif on which to send the RST (since we have no pcb)
seqno	the sequence number to use for the outgoing segment
ackno	the acknowledge number to use for the outgoing segment
local_ip	the local IP address to send the segment from
remote_ip	the remote IP address to send the segment to
local_port	the local TCP port to send the segment from
remote_port	the remote TCP port to send the segment to

pcb	Protocol control block for the TCP connection to send the ACK

pcb	the tcp_pcb over which to send a segment

4.86 src/core/timeouts.c File Reference

```
#include "lwip/opt.h"#include "lwip/timeouts.h"#include "lwip/priv/tcp_priv.h"#include " ↩
    lwip/def.h"#include "lwip/memp.h"#include "lwip/priv/tcpip_priv.h"#include "lwip/ ↩
    ip4_frag.h"#include "lwip/etharp.h"#include "lwip/dhcp.h"#include "lwip/acd.h"#include " ↩
    lwip/igmp.h"#include "lwip/dns.h"#include "lwip/nd6.h"#include "lwip/ip6_frag.h"#include ↩
     "lwip/mld6.h"#include "lwip/dhcp6.h"#include "lwip/sys.h"#include "lwip/pbuf.h"
```

4.86.1 Functions

- void tcp_timer_needed (void)
- void sys_timeouts_init (void)
- void sys_timeout (u32_t msecs, sys_timeout_handler handler, void *arg)
- void sys_untimeout (sys_timeout_handler handler, void *arg)
- void sys_check_timeouts (void)
- void sys_restart_timeouts (void)
- u32_t sys_timeouts_sleeptime (void)

4.86.2 Variables

- const struct lwip_cyclic_timer lwip_cyclic_timers []
- const int lwip_num_cyclic_timers = (sizeof(lwip_cyclic_timers)/sizeof((lwip_cyclic_timers)[0]))

4.86.3 Detailed Description

Stack-internal timers implementation. This file includes timer callbacks for stack-internal timers as well as functions to set up or stop timers and check for expired timers.

4.86.4 Function Documentation

4.86.4.1 sys_restart_timeouts()

```
void sys_restart_timeouts (void )
```

Rebase the timeout times to the current time. This is necessary if sys_check_timeouts() hasn't been called for a long time (e.g. while saving energy) to prevent all timer functions of that period being called.

pcb	the tcp_pcb for which to split the unsent head
split	the amount of payload to remain in the head

pcb	the tcp_pcb for which to send a zero-window probe packet

4.86.4.2 sys_timeout()

`void sys_timeout (u32_t msecs, sys_timeout_handler handler, void * arg)`

Create a one-shot timer (aka timeout). Timeouts are processed in the following cases:

- while waiting for a message using sys_timeouts_mbox_fetch()
- by calling sys_check_timeouts() (NO_SYS==1 only)

Parameters

msecs	time in milliseconds after that the timer should expire
handler	callback function to call when msecs have elapsed
arg	argument to pass to the callback function

4.86.4.3 sys_timeouts_init()

`void sys_timeouts_init (void )`

Initialize this module

4.86.4.4 sys_timeouts_sleeptime()

`u32_t sys_timeouts_sleeptime (void )`

Return the time left before the next timeout is due. If no timeouts are enqueued, returns 0xffffffff

4.86.4.5 sys_untimeout()

`void sys_untimeout (sys_timeout_handler handler, void * arg)`

Go through timeout list (for this task only) and remove the first matching entry (subsequent entries remain untouched), even though the timeout has not triggered yet.

Parameters

handler	callback function that would be called by the timeout
arg	callback argument that would be passed to handler

4.86.4.6 tcp_timer_needed()

`void tcp_timer_needed (void )`

Called from TCP_REG when registering a new PCB: the reason is to have the TCP timer only running when there are active (or time-wait) PCBs.

4.86.5 Variable Documentation

4.86.5.1 lwip_cyclic_timers

```
const struct lwip_cyclic_timer lwip_cyclic_timers[]
```

This array contains all stack-internal cyclic timers. To get the number of timers, use LWIP_ARRAYSIZE()

4.86.5.2 lwip_num_cyclic_timers

```
const int lwip_num_cyclic_timers = (sizeof( lwip_cyclic_timers )/sizeof(( lwip_cyclic_time
)[0]))
```

Array size of lwip_cyclic_timers[]

4.87 src/core/udp.c File Reference

```
#include "lwip/opt.h"#include "lwip/udp.h"#include "lwip/def.h"#include "lwip/memp.h"# ↩
    include "lwip/inet_chksum.h"#include "lwip/ip_addr.h"#include "lwip/ip6.h"#include "lwip ↩
    /ip6_addr.h"#include "lwip/netif.h"#include "lwip/icmp.h"#include "lwip/icmp6.h"#include ↩
     "lwip/stats.h"#include "lwip/snmp.h"#include "lwip/dhcp.h"#include <string.h>
```

4.87.1 Functions

- void udp_init (void)
- void udp_input (struct pbuf *p, struct netif *inp)
- err_t udp_send (struct udp_pcb *pcb, struct pbuf *p)
- err_t udp_sendto (struct udp_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, u16_t dst_port)
- err_t udp_sendto_if (struct udp_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, u16_t dst_port, struct netif *netif)
- err_t udp_sendto_if_src (struct udp_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, u16_t dst_port, struct netif *netif, const ip_addr_t *src_ip)
- err_t udp_bind (struct udp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port)
- void udp_bind_netif (struct udp_pcb *pcb, const struct netif *netif)
- err_t udp_connect (struct udp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port)
- void udp_disconnect (struct udp_pcb *pcb)
- void udp_recv (struct udp_pcb *pcb, udp_recv_fn recv, void *recv_arg)
- void udp_remove (struct udp_pcb *pcb)
- struct udp_pcb * udp_new (void)
- struct udp_pcb * udp_new_ip_type (u8_t type)
- void udp_netif_ip_addr_changed (const ip_addr_t *old_addr, const ip_addr_t *new_addr)

4.87.2 Detailed Description

User Datagram Protocol module The code for the User Datagram Protocol UDP & UDPLite (RFC 3828). See also UDP

4.87.3 Function Documentation

4.87.3.1 udp_init()

```
void udp_init (void )
```

Initialize this module.

4.87.3.2 udp_input()

```
void udp_input (struct pbuf * p, struct netif * inp)
```

Process an incoming UDP datagram.

Given an incoming UDP datagram (as a chain of pbufs) this function finds a corresponding UDP PCB and hands over the pbuf to the pcbs recv function. If no pcb is found or the datagram is incorrect, the pbuf is freed.

Parameters

p	pbuf to be demultiplexed to a UDP PCB (p->payload pointing to the UDP header)
inp	network interface on which the datagram was received.

4.87.3.3 udp_netif_ip_addr_changed()

```
void udp_netif_ip_addr_changed (const ip_addr_t * old_addr, const ip_addr_t * new_addr)
```

This function is called from netif.c when address is changed

Parameters

old_addr	IP address of the netif before change
new_addr	IP address of the netif after change

4.88 src/include/compat/posix/arpa/inet.h File Reference

```
#include "lwip/sockets.h"
```

4.88.1 Detailed Description

This file is a posix wrapper for lwip/sockets.h.

4.89 src/include/lwip/inet.h File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/ip_addr.h"#include "lwip/ip6_addr. ←
   h"
```

4.89.1 Macros

- #define INADDR_NONE IPADDR_NONE
- #define INADDR_LOOPBACK IPADDR_LOOPBACK
- #define INADDR_ANY IPADDR_ANY
- #define INADDR_BROADCAST IPADDR_BROADCAST
- #define IN6ADDR_ANY_INIT {{{0,0,0,0}}}
- #define IN6ADDR_LOOPBACK_INIT {{{0,0,0,PP_HTONL(1)}}}

4.89.2 Variables

- const struct in6_addr in6addr_any

4.89.3 Detailed Description

This file (together with sockets.h) aims to provide structs and functions from

- arpa/inet.h
- netinet/in.h

4.89.4 Macro Definition Documentation

4.89.4.1 IN6ADDR_ANY_INIT

```
#define IN6ADDR_ANY_INIT   {{{0,0,0,0}}}
```

This macro can be used to initialize a variable of type struct in6_addr to the IPv6 wildcard address.

4.89.4.2 IN6ADDR_LOOPBACK_INIT

```
#define IN6ADDR_LOOPBACK_INIT   {{{0,0,0,PP_HTONL(1)}}}
```

This macro can be used to initialize a variable of type struct in6_addr to the IPv6 loopback address.

4.89.4.3 INADDR_ANY

```
#define INADDR_ANY   IPADDR_ANY
```

0.0.0.0

4.89.4.4 INADDR_BROADCAST

```
#define INADDR_BROADCAST   IPADDR_BROADCAST
```

255.255.255.255

4.89.4.5 INADDR_LOOPBACK

```
#define INADDR_LOOPBACK   IPADDR_LOOPBACK
```

127.0.0.1

4.89.4.6 INADDR_NONE

```
#define INADDR_NONE   IPADDR_NONE
```

255.255.255.255

4.89.5 Variable Documentation

4.89.5.1 in6addr_any

```
const struct in6_addr in6addr_any[extern]
```

This variable is initialized by the system to contain the wildcard IPv6 address.

4.90 src/include/compat/posix/net/if.h File Reference

```
#include "lwip/if_api.h"
```

4.90.1 Detailed Description

This file is a posix wrapper for lwip/if_api.h.

4.91 src/include/compat/posix/netdb.h File Reference

```
#include "lwip/netdb.h"
```

4.91.1 Detailed Description

This file is a posix wrapper for lwip/netdb.h.

4.92 src/include/lwip/netdb.h File Reference

```
#include "lwip/opt.h"#include "lwip/arch.h"#include "lwip/inet.h"#include "lwip/sockets.h"
```

4.92.1 Macros

- #define EAI_NONAME 200

4.92.2 Functions

- struct hostent * lwip_gethostbyname (const char *name)
- int lwip_gethostbyname_r (const char *name, struct hostent *ret, char *buf, size_t buflen, struct hostent **result, int *h_errnop)
- void lwip_freeaddrinfo (struct addrinfo *ai)
- int lwip_getaddrinfo (const char *nodename, const char *servname, const struct addrinfo *hints, struct addrinfo **res)

4.92.3 Variables

- int h_errno

4.92.4 Detailed Description

NETDB API (sockets)

4.92.5 Macro Definition Documentation

4.92.5.1 EAI_NONAME

```
#define EAI_NONAME   200
```

Errors used by the DNS API functions, h_errno can be one of them

4.92.6 Function Documentation

4.92.6.1 lwip_freeaddrinfo()

```
void lwip_freeaddrinfo (struct addrinfo * ai)
```

Frees one or more addrinfo structures returned by getaddrinfo(), along with any additional storage associated with those structures. If the ai_next field of the structure is not null, the entire list of structures is freed.

Parameters

ai	struct addrinfo to free

4.92.6.2 lwip_getaddrinfo()

```
int lwip_getaddrinfo (const char * nodename, const char * servname, const struct addrinfo
* hints, struct addrinfo ** res)
```

Translates the name of a service location (for example, a host name) and/or a service name and returns a set of socket addresses and associated information to be used in creating a socket with which to address the specified service. Memory for the result is allocated internally and must be freed by calling lwip_freeaddrinfo()!

Due to a limitation in dns_gethostbyname, only the first address of a host is returned. Also, service names are not supported (only port numbers)!

Parameters

nodename	descriptive name or address string of the host (may be NULL -> local address)
servname	port number as string of NULL
hints	structure containing input values that set socktype and protocol
res	pointer to a pointer where to store the result (set to NULL on failure)

Returns 0 on success, non-zero on failure

name	the hostname to resolve

4.92.6.3 lwip_gethostbyname()

```
struct hostent * lwip_gethostbyname (const char * name)
```

Returns an entry containing addresses of address family AF_INET for the host with name name. Due to dns_gethostbyname limitations, only one address is returned.

Parameters

Returns an entry containing addresses of address family AF_INET for the host with name name

4.92.6.4 lwip_gethostbyname_r()

```
int lwip_gethostbyname_r (const char * name, struct hostent * ret, char * buf, size_t bufl
struct hostent ** result, int * h_errnop)
```

Thread-safe variant of lwip_gethostbyname: instead of using a static buffer, this function takes buffer and errno pointers as arguments and uses these for the result.

Parameters

name	the hostname to resolve
ret	pre-allocated struct where to store the result
buf	pre-allocated buffer where to store additional data
buflen	the size of buf
result	pointer to a hostent pointer that is set to ret on success and set to zero on error
h_errnop	pointer to an int where to store errors (instead of modifying the global h_errno)

Returns 0 on success, non-zero on error, additional error information is stored in *h_errnop instead of h_errno to be thread-safe

4.92.7 Variable Documentation

4.92.7.1 h_errno

```
int h_errno[extern]
```

h_errno is exported in netdb.h for access by applications.

4.93 src/include/compat/posix/sys/socket.h File Reference

```
#include "lwip/sockets.h"
```

4.93.1 Detailed Description

This file is a posix wrapper for lwip/sockets.h.

4.94 src/include/compat/stdc/errno.h File Reference

```
#include "lwip/errno.h"
```

4.94.1 Detailed Description

This file is a posix/stdc wrapper for lwip/errno.h.

4.95 src/include/lwip/errno.h File Reference

```
#include "lwip/opt.h"
```

4.95.1 Detailed Description

Posix Errno defines

4.96 src/include/lwip/acd.h File Reference

```
#include "lwip/opt.h"#include "lwip/netif.h"#include "lwip/etharp.h"#include "lwip/prot/acd ↩
    .h"
```

4.96.1 Data Structures

- struct acd

4.96.2 Macros

- #define ACD_TMR_INTERVAL 100

4.96.3 Typedefs

- typedef void(* acd_conflict_callback_t) (struct netif *netif, acd_callback_enum_t state)

4.96.4 Functions

- err_t acd_add (struct netif *netif, struct acd *acd, acd_conflict_callback_t acd_conflict_callback)
- void acd_remove (struct netif *netif, struct acd *acd)
- err_t acd_start (struct netif *netif, struct acd *acd, ip4_addr_t ipaddr)
- err_t acd_stop (struct acd *acd)
- void acd_arp_reply (struct netif *netif, struct etharp_hdr *hdr)
- void acd_tmr (void)
- void acd_network_changed_link_down (struct netif *netif)
- void acd_netif_ip_addr_changed (struct netif *netif, const ip_addr_t *old_addr, const ip_addr_t *new_addr)

4.96.5 Detailed Description

ACD IPv4 Address Conflict Detection

4.96.6 Macro Definition Documentation

4.96.6.1 ACD_TMR_INTERVAL

```
#define ACD_TMR_INTERVAL   100
```

ACD Timing ACD_TMR_INTERVAL msecs, I recommend a value of 100. The value must divide 1000 with a remainder almost 0. Possible values are 1000, 500, 333, 250, 200, 166, 142, 125, 111, 100 ....

4.96.7 Typedef Documentation

4.96.7.1 acd_conflict_callback_t

```
typedef void(* acd_conflict_callback_t) (struct netif *netif, acd_callback_enum_t state)
```

Callback function: Handle conflict information from ACD module

Parameters

netif	network interface to handle conflict information on
state	acd_callback_enum_t

4.96.8 Function Documentation

4.96.8.1 acd_arp_reply()

```
void acd_arp_reply (struct netif * netif, struct etharp_hdr * hdr)
```

Handles every incoming ARP Packet, called by etharp_input().

Parameters

netif	network interface to use for acd processing
hdr	Incoming ARP packet

4.96.8.2 acd_tmr()

```
void acd_tmr (void )
```

Has to be called in loop every ACD_TMR_INTERVAL milliseconds

4.97 src/include/lwip/prot/acd.h File Reference

4.97.1 Detailed Description

ACD protocol definitions

4.98 src/include/lwip/altcp.h File Reference

```
#include "lwip/opt.h"#include "lwip/tcpbase.h"#include "lwip/err.h"#include "lwip/pbuf.h"# ↩
    include "lwip/ip_addr.h"
```

4.98.1 Data Structures

- struct altcp_allocator_s

4.98.2 Typedefs

- typedef struct altcp_allocator_s altcp_allocator_t

4.98.3 Functions

- struct altcp_pcb * altcp_new (altcp_allocator_t *allocator)
- struct altcp_pcb * altcp_new_ip6 (altcp_allocator_t *allocator)
- struct altcp_pcb * altcp_new_ip_type (altcp_allocator_t *allocator, u8_t ip_type)
- void altcp_arg (struct altcp_pcb *conn, void *arg)
- void altcp_accept (struct altcp_pcb *conn, altcp_accept_fn accept)
- void altcp_recv (struct altcp_pcb *conn, altcp_recv_fn recv)
- void altcp_sent (struct altcp_pcb *conn, altcp_sent_fn sent)
- void altcp_poll (struct altcp_pcb *conn, altcp_poll_fn poll, u8_t interval)
- void altcp_err (struct altcp_pcb *conn, altcp_err_fn err)
- void altcp_recved (struct altcp_pcb *conn, u16_t len)
- err_t altcp_bind (struct altcp_pcb *conn, const ip_addr_t *ipaddr, u16_t port)
- err_t altcp_connect (struct altcp_pcb *conn, const ip_addr_t *ipaddr, u16_t port, altcp_connected_fn connected)
- struct altcp_pcb * altcp_listen_with_backlog_and_err (struct altcp_pcb *conn, u8_t backlog, err_t *err)
- void altcp_abort (struct altcp_pcb *conn)
- err_t altcp_close (struct altcp_pcb *conn)
- err_t altcp_shutdown (struct altcp_pcb *conn, int shut_rx, int shut_tx)
- err_t altcp_write (struct altcp_pcb *conn, const void *dataptr, u16_t len, u8_t apiflags)
- err_t altcp_output (struct altcp_pcb *conn)
- u16_t altcp_mss (struct altcp_pcb *conn)
- u16_t altcp_sndbuf (struct altcp_pcb *conn)
- u16_t altcp_sndqueuelen (struct altcp_pcb *conn)
- void altcp_setprio (struct altcp_pcb *conn, u8_t prio)

4.98.4 Detailed Description

Application layered TCP connection API (to be used from TCPIP thread)

This file contains the generic API. For more details see Application layered TCP Introduction.

4.99 src/include/lwip/altcp_tcp.h File Reference

```
#include "lwip/opt.h"#include "lwip/altcp.h"
```

4.99.1 Functions

- struct altcp_pcb * altcp_tcp_alloc (void *arg, u8_t ip_type)

4.99.2 Detailed Description

Application layered TCP connection API (to be used from TCPIP thread) This interface mimics the tcp callback API to the application while preventing direct linking (much like virtual functions). This way, an application can make use of other application layer protocols on top of TCP without knowing the details (e.g. TLS, proxy connection).

This file contains the base implementation calling into tcp.

4.99.3 Function Documentation

4.99.3.1 altcp_tcp_alloc()

```
struct altcp_pcb * altcp_tcp_alloc (void * arg, u8_t ip_type)
```

altcp_tcp allocator function fitting to altcp_allocator_t / altcp_new.

arg pointer is not used for TCP.

4.100 src/include/lwip/altcp_tls.h File Reference

```
#include "lwip/opt.h"#include "lwip/altcp.h"#include "lwip/apps/altcp_tls_mbedtls_opts.h"
```

4.100.1 Functions

- struct altcp_tls_config * altcp_tls_create_config_server (u8_t cert_count)
- err_t altcp_tls_config_server_add_privkey_cert (struct altcp_tls_config *config, const u8_t *privkey, size_t privkey_len, const u8_t *privkey_pass, size_t privkey_pass_len, const u8_t *cert, size_t cert_len)
- struct altcp_tls_config * altcp_tls_create_config_server_privkey_cert (const u8_t *privkey, size_t privkey_len, const u8_t *privkey_pass, size_t privkey_pass_len, const u8_t *cert, size_t cert_len)
- struct altcp_tls_config * altcp_tls_create_config_client (const u8_t *cert, size_t cert_len)
- struct altcp_tls_config * altcp_tls_create_config_client_2wayauth (const u8_t *ca, size_t ca_len, const u8_t *privkey, size_t privkey_len, const u8_t *privkey_pass, size_t privkey_pass_len, const u8_t *cert, size_t cert_len)
- int altcp_tls_configure_alpn_protocols (struct altcp_tls_config *conf, const char **protos)
- void altcp_tls_free_config (struct altcp_tls_config *conf)
- void altcp_tls_free_entropy (void)
- struct altcp_pcb * altcp_tls_wrap (struct altcp_tls_config *config, struct altcp_pcb *inner_pcb)
- struct altcp_pcb * altcp_tls_new (struct altcp_tls_config *config, u8_t ip_type)

- struct altcp_pcb * altcp_tls_alloc (void *arg, u8_t ip_type)
- void * altcp_tls_context (struct altcp_pcb *conn)
- void altcp_tls_init_session (struct altcp_tls_session *dest)
- err_t altcp_tls_get_session (struct altcp_pcb *conn, struct altcp_tls_session *dest)
- err_t altcp_tls_set_session (struct altcp_pcb *conn, struct altcp_tls_session *from)
- void altcp_tls_free_session (struct altcp_tls_session *dest)

4.100.2 Detailed Description

Application layered TCP/TLS connection API (to be used from TCPIP thread)

4.101 src/include/lwip/api.h File Reference

```
#include "lwip/opt.h"#include "lwip/arch.h"#include "lwip/netbuf.h"#include "lwip/sys.h"# ↩
    include "lwip/ip_addr.h"#include "lwip/err.h"
```

4.101.1 Data Structures

- struct netconn
- struct netvector

4.101.2 Macros

- #define NETCONN_FLAG_MBOXCLOSED 0x01
- #define NETCONN_FLAG_NON_BLOCKING 0x02
- #define NETCONN_FLAG_IN_NONBLOCKING_CONNECT 0x04
- #define NETCONN_FLAG_CHECK_WRITESPACE 0x10
- #define NETCONN_FLAG_IPV6_V6ONLY 0x20
- #define NETCONN_FIN_RX_PENDING 0x80
- #define API_EVENT(c, e, l)
- #define netconn_new(t) netconn_new_with_proto_and_callback(t, 0, NULL)
- #define netconn_type(conn) (conn->type)
- #define netconn_set_nonblocking(conn, val)
- #define netconn_is_nonblocking(conn) (((conn)->flags & NETCONN_FLAG_NON_BLOCKING) != 0)
- #define netconn_set_ipv6only(conn, val)
- #define netconn_get_ipv6only(conn) (((conn)->flags & NETCONN_FLAG_IPV6_V6ONLY) != 0)
- #define netconn_set_sendtimeout(conn, timeout) ((conn)->send_timeout = (timeout))
- #define netconn_get_sendtimeout(conn) ((conn)->send_timeout)
- #define netconn_set_recvbufsize(conn, recvbufsize) ((conn)->recv_bufsize = (recvbufsize))
- #define netconn_get_recvbufsize(conn) ((conn)->recv_bufsize)

4.101.3 Typedefs

- typedef void(* netconn_callback) (struct netconn *, enum netconn_evt, u16_t len)

4.101.4 Enumerations

- enum netconn_type { }
- enum netconn_state
- enum netconn_evt
- enum netconn_igmp

4.101.5 Functions

- struct netconn * netconn_new_with_proto_and_callback (enum netconn_type t, u8_t proto, netconn_callback callback)
- err_t netconn_prepare_delete (struct netconn *conn)
- err_t netconn_delete (struct netconn *conn)
- err_t netconn_getaddr (struct netconn *conn, ip_addr_t *addr, u16_t *port, u8_t local)
- err_t netconn_bind (struct netconn *conn, const ip_addr_t *addr, u16_t port)
- err_t netconn_bind_if (struct netconn *conn, u8_t if_idx)
- err_t netconn_connect (struct netconn *conn, const ip_addr_t *addr, u16_t port)
- err_t netconn_disconnect (struct netconn *conn)
- err_t netconn_listen_with_backlog (struct netconn *conn, u8_t backlog)
- err_t netconn_accept (struct netconn *conn, struct netconn **new_conn)
- err_t netconn_recv (struct netconn *conn, struct netbuf **new_buf)
- err_t netconn_recv_udp_raw_netbuf (struct netconn *conn, struct netbuf **new_buf)
- err_t netconn_recv_udp_raw_netbuf_flags (struct netconn *conn, struct netbuf **new_buf, u8_t apiflags)
- err_t netconn_recv_tcp_pbuf (struct netconn *conn, struct pbuf **new_buf)
- err_t netconn_recv_tcp_pbuf_flags (struct netconn *conn, struct pbuf **new_buf, u8_t apiflags)
- err_t netconn_sendto (struct netconn *conn, struct netbuf *buf, const ip_addr_t *addr, u16_t port)
- err_t netconn_send (struct netconn *conn, struct netbuf *buf)
- err_t netconn_write_partly (struct netconn *conn, const void *dataptr, size_t size, u8_t apiflags, size_t *bytes_written)
- err_t netconn_write_vectors_partly (struct netconn *conn, struct netvector *vectors, u16_t vectorcnt, u8_t apiflags, size_t *bytes_written)
- err_t netconn_close (struct netconn *conn)
- err_t netconn_shutdown (struct netconn *conn, u8_t shut_rx, u8_t shut_tx)
- err_t netconn_join_leave_group (struct netconn *conn, const ip_addr_t *multiaddr, const ip_addr_t *netif_addr, enum netconn_igmp join_or_leave)
- err_t netconn_join_leave_group_netif (struct netconn *conn, const ip_addr_t *multiaddr, u8_t if_idx, enum netconn_igmp join_or_leave)
- err_t netconn_gethostbyname_addrtype (const char *name, ip_addr_t *addr, u8_t dns_addrtype)
- err_t netconn_err (struct netconn *conn)

4.101.6 Detailed Description

netconn API (to be used from non-TCPIP threads)

4.101.7 Macro Definition Documentation

4.101.7.1 API_EVENT

`#define API_EVENT( c, e, l)` **Value:**

```
                        if (c->callback) {          \
                        (*c->callback)(c, e, l); \
                      }
```

Register an Network connection event

4.101.7.2 NETCONN_FIN_RX_PENDING

`#define NETCONN_FIN_RX_PENDING   0x80`

A FIN has been received but not passed to the application yet

4.101.7.3 NETCONN_FLAG_CHECK_WRITESPACE

`#define NETCONN_FLAG_CHECK_WRITESPACE   0x10`

If a nonblocking write has been rejected before, poll_tcp needs to check if the netconn is writable again

4.101.7.4 NETCONN_FLAG_IN_NONBLOCKING_CONNECT

`#define NETCONN_FLAG_IN_NONBLOCKING_CONNECT   0x04`

Was the last connect action a non-blocking one?

4.101.7.5 NETCONN_FLAG_IPV6_V6ONLY

`#define NETCONN_FLAG_IPV6_V6ONLY   0x20`

If this flag is set then only IPv6 communication is allowed on the netconn. As per RFC#3493 this features defaults to OFF allowing dual-stack usage by default.

4.101.7.6 NETCONN_FLAG_MBOXCLOSED

`#define NETCONN_FLAG_MBOXCLOSED   0x01`

This netconn had an error, don’t block on recvmbox/acceptmbox any more

4.101.7.7 NETCONN_FLAG_NON_BLOCKING

`#define NETCONN_FLAG_NON_BLOCKING   0x02`

Should this netconn avoid blocking?

4.101.7.8 netconn_get_recvbufsize

`#define netconn_get_recvbufsize( conn)    ((conn)->recv_bufsize)`

Get the receive buffer in bytes

4.101.7.9 netconn_get_sendtimeout

`#define netconn_get_sendtimeout( conn)    ((conn)->send_timeout)`

Get the send timeout in milliseconds

4.101.7.10 netconn_is_nonblocking

`#define netconn_is_nonblocking( conn)    (((conn)->flags & NETCONN_FLAG_NON_BLOCKING) != 0)`

Get the blocking status of netconn calls (

4.101.7.11 netconn_set_nonblocking

`#define netconn_set_nonblocking( conn, val)` **Value:**

```
  do { if(val) { \
  netconn_set_flags(conn, NETCONN_FLAG_NON_BLOCKING); \
} else { \
  netconn_clear_flags(conn, NETCONN_FLAG_NON_BLOCKING); }} while(0)
```

Set the blocking status of netconn calls (

4.101.7.12 netconn_set_recvbufsize

`#define netconn_set_recvbufsize( conn, recvbufsize)    ((conn)->recv_bufsize = (recvbufsize`

Set the receive buffer in bytes

4.101.7.13 netconn_set_sendtimeout

`#define netconn_set_sendtimeout( conn, timeout)    ((conn)->send_timeout = (timeout))`

Set the send timeout in milliseconds

4.101.7.14 netconn_type

`#define netconn_type( conn)    (conn->type)`

Get the type of a netconn (as enum netconn_type).

4.101.8 Typedef Documentation

4.101.8.1 netconn_callback

`typedef void(* netconn_callback) (struct netconn *, enum netconn_evt, u16_t len)`

A callback prototype to inform about events for a netconn

4.101.9 Enumeration Type Documentation

4.101.9.1 netconn_evt

`enum netconn_evt`

Used to inform the callback function about changes

Event explanation:

In the netconn implementation, there are three ways to block a client:

- accept mbox (sys_arch_mbox_fetch(&conn->acceptmbox, &accept_ptr, 0); in netconn_accept())
- receive mbox (sys_arch_mbox_fetch(&conn->recvmbox, &buf, 0); in netconn_recv_data())
- send queue is full (sys_arch_sem_wait(LWIP_API_MSG_SEM(msg), 0); in lwip_netconn_do_write())

The events have to be seen as events signaling the state of these mboxes/semaphores. For non-blocking connections, you need to know in advance whether a call to a netconn function call would block or not, and these events tell you about that.

RCVPLUS events say: Safe to perform a potentially blocking call call once more. They are counted in sockets - three RCVPLUS events for accept mbox means you are safe to call netconn_accept 3 times without being blocked. Same thing for receive mbox.

RCVMINUS events say: Your call to to a possibly blocking function is "acknowledged". Socket implementation decrements the counter.

For TX, there is no need to count, its merely a flag. SENDPLUS means you may send something. SENDPLUS occurs when enough data was delivered to peer so netconn_send() can be called again. A SENDMINUS event occurs when the next call to a netconn_send() would be blocking.

4.101.9.2 netconn_igmp

`enum netconn_igmp`

Used for netconn_join_leave_group()

4.101.9.3 netconn_state

`enum netconn_state`

Current state of the netconn. Non-TCP netconns are always in state NETCONN_NONE!

4.101.10 Function Documentation

4.101.10.1 netconn_getaddr()

`err_t netconn_getaddr (struct netconn * conn, ip_addr_t * addr, u16_t * port, u8_t local)`

Get the local or remote IP address and port of a netconn. For RAW netconns, this returns the protocol instead of a port!

Parameters

conn	the netconn to query
addr	a pointer to which to save the IP address
port	a pointer to which to save the port (or protocol for RAW)
local	1 to get the local IP address, 0 to get the remote one

Returns ERR_CONN for invalid connections ERR_OK if the information was retrieved

4.101.10.2 netconn_new_with_proto_and_callback()

```
struct netconn * netconn_new_with_proto_and_callback (enum netconn_type t, u8_t proto,
netconn_callback callback)
```

Create a new netconn (of a specific type) that has a callback function. The corresponding pcb is also created.

Parameters

t	the type of 'connection' to create (

See also enum netconn_type)

Parameters

proto	the IP protocol for RAW IP pcbs
callback	a function to call on status changes (RX available, TX'ed)

Returns a newly allocated struct netconn or NULL on memory error

4.101.10.3 netconn_recv_udp_raw_netbuf()

```
err_t netconn_recv_udp_raw_netbuf (struct netconn * conn, struct netbuf ** new_buf)
```

Receive data (in form of a netbuf) from a UDP or RAW netconn

Parameters

conn	the netconn from which to receive data
new_buf	pointer where a new netbuf is stored when received data

Returns ERR_OK if data has been received, an error code otherwise (timeout, memory error or another error) ERR_ARG if conn is not a UDP/RAW netconn

4.101.10.4 netconn_recv_udp_raw_netbuf_flags()

```
err_t netconn_recv_udp_raw_netbuf_flags (struct netconn * conn, struct netbuf ** new_buf,
u8_t apiflags)
```

Receive data (in form of a netbuf) from a UDP or RAW netconn

Parameters

Returns ERR_OK if data has been received, an error code otherwise (timeout, memory error or another error) ERR_ARG if conn is not a UDP/RAW netconn

4.101.10.5 netconn_write_vectors_partly()

```
err_t netconn_write_vectors_partly (struct netconn * conn, struct netvector * vectors,
u16_t vectorcnt, u8_t apiflags, size_t * bytes_written)
```

Send vectorized data atomically over a TCP netconn.

Parameters

Returns ERR_OK if data was sent, any other err_t on error

conn	the netconn from which to receive data
new_buf	pointer where a new netbuf is stored when received data
apiflags	flags that control function behaviour. For now only: • NETCONN_DONTBLOCK: only read data that is available now, don't wait for more data

conn	the TCP netconn over which to send data
vectors	array of vectors containing data to send
vectorcnt	number of vectors in the array
apiflags	combination of following flags : • NETCONN_COPY: data will be copied into memory belonging to the stack • NETCONN_MORE: for TCP connection, PSH flag will be set on last segment sent • NETCONN_DONTBLOCK: only write the data if all data can be written at once
bytes_written	pointer to a location that receives the number of written bytes

4.102 src/include/lwip/apps/altcp_proxyconnect.h File Reference

```
#include "lwip/opt.h"#include "lwip/ip_addr.h"
```

4.102.1 Functions

- struct altcp_pcb * altcp_proxyconnect_new (struct altcp_proxyconnect_config *config, struct altcp_pcb *inner_pcb)
- struct altcp_pcb * altcp_proxyconnect_new_tcp (struct altcp_proxyconnect_config *config, u8_t ip_type)
- struct altcp_pcb * altcp_proxyconnect_alloc (void *arg, u8_t ip_type)
- struct altcp_pcb * altcp_proxyconnect_tls_alloc (void *arg, u8_t ip_type)

4.102.2 Detailed Description

Application layered TCP connection API that executes a proxy-connect.

This file provides a starting layer that executes a proxy-connect e.g. to set up TLS connections through a http proxy.

4.102.3 Function Documentation

4.102.3.1 altcp_proxyconnect_alloc()

```
struct altcp_pcb * altcp_proxyconnect_alloc (void * arg, u8_t ip_type)
```

Allocator function to allocate a proxy connect altcp pcb connecting directly via tcp to the proxy.

The returned pcb is a chain: altcp_proxyconnect - altcp_tcp - tcp pcb

This function is meant for use with altcp_new.

Parameters

arg	struct altcp_proxyconnect_config that contains the proxy settings
ip_type	IP type of the connection (lwip_ip_addr_type)

4.102.3.2 altcp_proxyconnect_new()

```
struct altcp_pcb * altcp_proxyconnect_new (struct altcp_proxyconnect_config * config, stru
altcp_pcb * inner_pcb)
```

Allocate a new altcp layer connecting through a proxy. This function gets the inner pcb passed.

Parameters

config	struct altcp_proxyconnect_config that contains the proxy settings
inner_pcb	pcb that makes the connection to the proxy (i.e. tcp pcb)

4.102.3.3 altcp_proxyconnect_new_tcp()

```
struct altcp_pcb * altcp_proxyconnect_new_tcp (struct altcp_proxyconnect_config * config,
u8_t ip_type)
```

Allocate a new altcp layer connecting through a proxy. This function allocates the inner pcb as tcp pcb, resulting in a direct tcp connection to the proxy.

Parameters

config	struct altcp_proxyconnect_config that contains the proxy settings
ip_type	IP type of the connection (lwip_ip_addr_type)

4.102.3.4 altcp_proxyconnect_tls_alloc()

```
struct altcp_pcb * altcp_proxyconnect_tls_alloc (void * arg, u8_t ip_type)
```

Allocator function to allocate a TLS connection through a proxy.

The returned pcb is a chain: altcp_tls - altcp_proxyconnect - altcp_tcp - tcp pcb

This function is meant for use with altcp_new.

Parameters

4.103 src/include/lwip/apps/altcp_tls_mbedtls_opts.h File Reference

```
#include "lwip/opt.h"
```

4.103.1 Macros

- #define LWIP_ALTCP_TLS_MBEDTLS 0
- #define ALTCP_MBEDTLS_DEBUG LWIP_DBG_OFF
- #define ALTCP_MBEDTLS_LIB_DEBUG LWIP_DBG_OFF

arg	struct altcp_proxyconnect_tls_config that contains the proxy settings and tls settings
ip_type	IP type of the connection (lwip_ip_addr_type)

- #define ALTCP_MBEDTLS_LIB_DEBUG_LEVEL_MIN 0
- #define ALTCP_MBEDTLS_USE_SESSION_CACHE 0
- #define ALTCP_MBEDTLS_SESSION_CACHE_SIZE 30
- #define ALTCP_MBEDTLS_SESSION_CACHE_TIMEOUT_SECONDS (60 * 60)
- #define ALTCP_MBEDTLS_USE_SESSION_TICKETS 0
- #define ALTCP_MBEDTLS_SESSION_TICKET_CIPHER MBEDTLS_CIPHER_AES_256_GCM
- #define ALTCP_MBEDTLS_SESSION_TICKET_TIMEOUT_SECONDS (60 * 60 * 24)
- #define ALTCP_MBEDTLS_AUTHMODE MBEDTLS_SSL_VERIFY_OPTIONAL

4.103.2 Detailed Description

Application layered TCP/TLS connection API (to be used from TCPIP thread)

This file contains options for an mbedtls port of the TLS layer.

4.103.3 Macro Definition Documentation

4.103.3.1 ALTCP_MBEDTLS_AUTHMODE

```
#define ALTCP_MBEDTLS_AUTHMODE   MBEDTLS_SSL_VERIFY_OPTIONAL
```

Certificate verification mode: MBEDTLS_SSL_VERIFY_NONE, MBEDTLS_SSL_VERIFY_OPTIONAL (default), MBEDTLS_SSL_
(recommended)

4.103.3.2 ALTCP_MBEDTLS_DEBUG

```
#define ALTCP_MBEDTLS_DEBUG   LWIP_DBG_OFF
```

Configure debug level of this file

4.103.3.3 ALTCP_MBEDTLS_LIB_DEBUG

```
#define ALTCP_MBEDTLS_LIB_DEBUG   LWIP_DBG_OFF
```

Configure lwIP debug level of the mbedTLS library

4.103.3.4 ALTCP_MBEDTLS_LIB_DEBUG_LEVEL_MIN

```
#define ALTCP_MBEDTLS_LIB_DEBUG_LEVEL_MIN   0
```

Configure minimum internal debug level of the mbedTLS library

4.103.3.5 ALTCP_MBEDTLS_SESSION_CACHE_SIZE

```
#define ALTCP_MBEDTLS_SESSION_CACHE_SIZE   30
```

Maximum cache size of the basic session cache

4.103.3.6 ALTCP_MBEDTLS_SESSION_CACHE_TIMEOUT_SECONDS

```
#define ALTCP_MBEDTLS_SESSION_CACHE_TIMEOUT_SECONDS   (60 * 60)
```

Set a session timeout in seconds for the basic session cache

4.103.3.7 ALTCP_MBEDTLS_SESSION_TICKET_CIPHER

```
#define ALTCP_MBEDTLS_SESSION_TICKET_CIPHER   MBEDTLS_CIPHER_AES_256_GCM
```

Session ticket cipher

4.103.3.8 ALTCP_MBEDTLS_SESSION_TICKET_TIMEOUT_SECONDS

```
#define ALTCP_MBEDTLS_SESSION_TICKET_TIMEOUT_SECONDS   (60 * 60 * 24)
```

Maximum timeout for session tickets

4.103.3.9 ALTCP_MBEDTLS_USE_SESSION_CACHE

```
#define ALTCP_MBEDTLS_USE_SESSION_CACHE   0
```

Enable the basic session cache ATTENTION: Using a session cache can lower security by reusing keys!

4.103.3.10 ALTCP_MBEDTLS_USE_SESSION_TICKETS

```
#define ALTCP_MBEDTLS_USE_SESSION_TICKETS   0
```

Use session tickets to speed up connection setup (needs MBEDTLS_SSL_SESSION_TICKETS enabled in mbedTLS config). ATTENTION: Using session tickets can lower security by reusing keys!

4.103.3.11 LWIP_ALTCP_TLS_MBEDTLS

```
#define LWIP_ALTCP_TLS_MBEDTLS   0
```

LWIP_ALTCP_TLS_MBEDTLS==1: use mbedTLS for TLS support for altcp API mbedtls include directory must be reachable via include search path

4.104 src/include/lwip/apps/http_client.h File Reference

```
#include "lwip/opt.h"#include "lwip/ip_addr.h"#include "lwip/err.h"#include "lwip/altcp.h"# ↩
    include "lwip/prot/iana.h"#include "lwip/pbuf.h"
```

4.104.1 Macros

- #define LWIP_HTTPC_HAVE_FILE_IO 0
- #define HTTP_DEFAULT_PORT LWIP_IANA_PORT_HTTP

4.104.2 Typedefs

- typedef enum ehttpc_result httpc_result_t
- typedef void(* httpc_result_fn) (void *arg, httpc_result_t httpc_result, u32_t rx_content_len, u32_t srv_res, err_t err)
- typedef err_t(* httpc_headers_done_fn) (httpc_state_t *connection, void *arg, struct pbuf *hdr, u16_t hdr_len, u32_t content_len)

4.104.3 Enumerations

- enum ehttpc_result { HTTPC_RESULT_OK = 0 , HTTPC_RESULT_ERR_UNKNOWN = 1 , HTTPC_RESULT_ERR_CONNECT = 2 , HTTPC_RESULT_ERR_HOSTNAME = 3 , HTTPC_RESULT_ERR_CLOSED = 4 , HTTPC_RESULT_ERR_TIMEOUT = 5 , HTTPC_RESULT_ERR_SVR_RESP = 6 , HTTPC_RESULT_ERR_MEM = 7 , HTTPC_RESULT_LOCAL_ABORT = 8 , HTTPC_RESULT_ERR_CONTENT_LEN = 9 }

4.104.4 Functions

- err_t httpc_get_file (const ip_addr_t *server_addr, u16_t port, const char *uri, const httpc_connection_t *settings, altcp_recv_fn recv_fn, void *callback_arg, httpc_state_t **connection)
- err_t httpc_get_file_dns (const char *server_name, u16_t port, const char *uri, const httpc_connection_t *settings, altcp_recv_fn recv_fn, void *callback_arg, httpc_state_t **connection)

4.104.5 Detailed Description

HTTP client

4.105 src/include/lwip/apps/httpd.h File Reference

```
#include "httpd_opts.h"#include "lwip/err.h"#include "lwip/pbuf.h"
```

4.105.1 Data Structures

- struct tCGI

4.105.2 Macros

- #define HTTPD_SSI_TAG_UNKNOWN 0xFFFF

4.105.3 Typedefs

- typedef const char *(* tCGIHandler) (int iIndex, int iNumParams, char *pcParam[], char *pcValue[])
- typedef u16_t(* tSSIHandler) (const char *ssi_tag_name, char *pcInsert, int iInsertLen)

4.105.4 Functions

- void http_set_cgi_handlers (const tCGI *pCGIs, int iNumHandlers)
- void http_set_ssi_handler (tSSIHandler pfnSSIHandler, const char **ppcTags, int iNumTags)
- err_t httpd_post_begin (void *connection, const char *uri, const char *http_request, u16_t http_request_len, int content_len, char *response_uri, u16_t response_uri_len, u8_t *post_auto_wnd)
- err_t httpd_post_receive_data (void *connection, struct pbuf *p)
- void httpd_post_finished (void *connection, char *response_uri, u16_t response_uri_len)
- void httpd_post_data_recved (void *connection, u16_t recved_len)
- void httpd_init (void)
- void httpd_inits (struct altcp_tls_config *conf)

4.105.5 Detailed Description

HTTP server

4.105.6 Macro Definition Documentation

4.105.6.1 HTTPD_SSI_TAG_UNKNOWN

```
#define HTTPD_SSI_TAG_UNKNOWN   0xFFFF
```

For LWIP_HTTPD_SSI_RAW==1, return this to indicate the tag is unknown. In this case, the webserver writes a warning into the page. You can also just return 0 to write nothing for unknown tags.

4.106 src/include/lwip/apps/httpd_opts.h File Reference

```
#include "lwip/opt.h"#include "lwip/prot/iana.h"
```

4.106.1 Macros

- #define LWIP_HTTPD_CGI 0
- #define LWIP_HTTPD_CGI_SSI 0
- #define LWIP_HTTPD_SSI 0
- #define LWIP_HTTPD_SSI_RAW 0
- #define LWIP_HTTPD_SSI_BY_FILE_EXTENSION 1
- #define LWIP_HTTPD_SSI_EXTENSIONS ".shtml", ".shtm", ".ssi", ".xml", ".json"
- #define LWIP_HTTPD_SUPPORT_POST 0
- #define LWIP_HTTPD_SSI_MULTIPART 0
- #define HTTPD_SERVER_AGENT "lwIP/" LWIP_VERSION_STRING " (http://savannah.nongnu.org/projects/lwip)"
- #define LWIP_HTTPD_DYNAMIC_HEADERS 0
- #define HTTPD_USE_MEM_POOL 0
- #define HTTPD_SERVER_PORT LWIP_IANA_PORT_HTTP
- #define HTTPD_SERVER_PORT_HTTPS LWIP_IANA_PORT_HTTPS
- #define HTTPD_ENABLE_HTTPS 0
- #define HTTPD_MAX_RETRIES 4
- #define HTTPD_POLL_INTERVAL 4
- #define HTTPD_TCP_PRIO TCP_PRIO_MIN
- #define LWIP_HTTPD_TIMING 0
- #define HTTPD_DEBUG_TIMING LWIP_DBG_OFF
- #define LWIP_HTTPD_SUPPORT_EXTSTATUS 0
- #define LWIP_HTTPD_SUPPORT_V09 1

- #define LWIP_HTTPD_SUPPORT_11_KEEPALIVE 0
- #define LWIP_HTTPD_SUPPORT_REQUESTLIST 1
- #define LWIP_HTTPD_REQ_QUEUELEN 5
- #define LWIP_HTTPD_REQ_BUFSIZE LWIP_HTTPD_MAX_REQ_LENGTH
- #define LWIP_HTTPD_MAX_REQ_LENGTH LWIP_MIN(1023, (LWIP_HTTPD_REQ_QUEUELEN * PBUF_POOL_BUFSIZE)
- #define LWIP_HTTPD_MAX_REQUEST_URI_LEN 63
- #define LWIP_HTTPD_POST_MAX_RESPONSE_URI_LEN 63
- #define LWIP_HTTPD_SSI_INCLUDE_TAG 1
- #define LWIP_HTTPD_ABORT_ON_CLOSE_MEM_ERROR 0
- #define LWIP_HTTPD_KILL_OLD_ON_CONNECTIONS_EXCEEDED 0
- #define LWIP_HTTPD_OMIT_HEADER_FOR_EXTENSIONLESS_URI 0
- #define HTTP_IS_TAG_VOLATILE(ptr) TCP_WRITE_FLAG_COPY
- #define LWIP_HTTPD_CUSTOM_FILES 0
- #define LWIP_HTTPD_DYNAMIC_FILE_READ 0
- #define LWIP_HTTPD_FILE_STATE 0
- #define LWIP_HTTPD_FILE_EXTENSION 0
- #define HTTPD_PRECALCULATED_CHECKSUM 0
- #define LWIP_HTTPD_FS_ASYNC_READ 0
- #define HTTPD_FSDATA_FILE "fsdata.c"

4.106.2 Detailed Description

HTTP server options list

4.107 src/include/lwip/apps/lwiperf.h File Reference

```
#include "lwip/opt.h"#include "lwip/ip_addr.h"
```

4.107.1 Typedefs

- typedef void(* lwiperf_report_fn) (void *arg, enum lwiperf_report_type report_type, const ip_addr_t *local_addr, u16_t local_port, const ip_addr_t *remote_addr, u16_t remote_port, u32_t bytes_transferred, u32_t ms_duration, u32_t bandwidth_kbitpsec)

4.107.2 Enumerations

- enum lwiperf_report_type { LWIPERF_TCP_DONE_SERVER , LWIPERF_TCP_DONE_CLIENT , LWIPERF_TCP_ABORTED_L
  , LWIPERF_TCP_ABORTED_LOCAL_DATAERROR , LWIPERF_TCP_ABORTED_LOCAL_TXERROR , LWIPERF_TCP_ABC
  }
- enum lwiperf_client_type { LWIPERF_CLIENT , LWIPERF_DUAL , LWIPERF_TRADEOFF }

4.107.3 Functions

- void * lwiperf_start_tcp_server (const ip_addr_t *local_addr, u16_t local_port, lwiperf_report_fn report_fn, void *report_arg)
- void * lwiperf_start_tcp_server_default (lwiperf_report_fn report_fn, void *report_arg)
- void * lwiperf_start_tcp_client (const ip_addr_t *remote_addr, u16_t remote_port, enum lwiperf_client_type type, lwiperf_report_fn report_fn, void *report_arg)
- void * lwiperf_start_tcp_client_default (const ip_addr_t *remote_addr, lwiperf_report_fn report_fn, void *report_arg)
- void lwiperf_abort (void *lwiperf_session)

4.107.4 Detailed Description

lwIP iPerf server implementation

4.107.5 Typedef Documentation

4.107.5.1 lwiperf_report_fn

```
typedef void(* lwiperf_report_fn) (void *arg, enum lwiperf_report_type report_type, const
ip_addr_t *local_addr, u16_t local_port, const ip_addr_t *remote_addr, u16_t remote_port,
u32_t bytes_transferred, u32_t ms_duration, u32_t bandwidth_kbitpsec)
```

Prototype of a report function that is called when a session is finished. This report function can show the test results.

Parameters

arg	Report_arg from when the test was started.
report_type	contains the test result
local_addr	The local address from the session
local_port	The local port
remote_addr	The remote address from the session
remote_port	The remote port
bytes_transferred	Total transferred bytes
ms_duration	Total session duration, in milliseconds
bandwidth_kbitpsec	Average bandwidth during the session, in kbps

4.107.6 Enumeration Type Documentation

4.107.6.1 lwiperf_client_type

```
enum lwiperf_client_type
```

Control

LWIPERF_CLIENT	Unidirectional tx only test
LWIPERF_DUAL	Do a bidirectional test simultaneously
LWIPERF_TRADEOFF	Do a bidirectional test individually

4.107.6.2 lwiperf_report_type

```
enum lwiperf_report_type
```

lwIPerf test results

LWIPERF_TCP_DONE_SERVER	The server side test is done
LWIPERF_TCP_DONE_CLIENT	The client side test is done
LWIPERF_TCP_ABORTED_LOCAL	Local error lead to test abort
LWIPERF_TCP_ABORTED_LOCAL_DATAERROR	Data check error lead to test abort
LWIPERF_TCP_ABORTED_LOCAL_TXERROR	Transmit error lead to test abort
LWIPERF_TCP_ABORTED_REMOTE	Remote side aborted the test

4.108 src/include/lwip/apps/mdns.h File Reference

```
#include "lwip/apps/mdns_opts.h"#include "lwip/netif.h"
```

4.108.1 Data Structures

- struct mdns_rr_info

4.108.2 Macros

- #define mdns_resp_netif_settings_changed(netif)   mdns_resp_announce(netif)

4.108.3 Typedefs

- typedef void(* service_get_txt_fn_t) (struct mdns_service *service, void *txt_userdata)
- typedef void(* mdns_name_result_cb_t) (struct netif *netif, u8_t result, s8_t slot)

4.108.4 Functions

- void * mdns_get_service_txt_userdata (struct netif *netif, s8_t slot)
- void mdns_resp_init (void)
- void mdns_resp_register_name_result_cb (mdns_name_result_cb_t cb)
- err_t mdns_resp_add_netif (struct netif *netif, const char *hostname)
- err_t mdns_resp_remove_netif (struct netif *netif)
- err_t mdns_resp_rename_netif (struct netif *netif, const char *hostname)
- int mdns_resp_netif_active (struct netif *netif)
- s8_t mdns_resp_add_service (struct netif *netif, const char *name, const char *service, enum mdns_sd_proto proto, u16_t port, service_get_txt_fn_t txt_fn, void *txt_userdata)
- err_t mdns_resp_del_service (struct netif *netif, u8_t slot)
- err_t mdns_resp_rename_service (struct netif *netif, u8_t slot, const char *name)
- err_t mdns_resp_add_service_txtitem (struct mdns_service *service, const char *txt, u8_t txt_len)

- void mdns_resp_restart_delay (struct netif *netif, uint32_t delay)
- void mdns_resp_restart (struct netif *netif)
- void mdns_resp_announce (struct netif *netif)
- err_t mdns_search_service (const char *name, const char *service, enum mdns_sd_proto proto, struct netif *netif, search_result_fn_t result_fn, void *arg, u8_t *request_id)
- void mdns_search_stop (u8_t request_id)

4.108.5 Detailed Description

MDNS responder

4.108.6 Typedef Documentation

4.108.6.1 mdns_name_result_cb_t

```
typedef void(* mdns_name_result_cb_t) (struct netif *netif, u8_t result, s8_t slot)
```

Callback function to let application know the result of probing network for name uniqueness, called with result MDNS_PROBING_SUCC
if no other node claimed use for the name for the netif or a service and is safe to use, or MDNS_PROBING_CONFLICT if another node is already using it and mdns is disabled on this interface

4.108.6.2 service_get_txt_fn_t

```
typedef void(* service_get_txt_fn_t) (struct mdns_service *service, void *txt_userdata)
```

Callback function to add text to a reply, called when generating the reply

4.108.7 Function Documentation

4.108.7.1 mdns_resp_register_name_result_cb()

```
void mdns_resp_register_name_result_cb (mdns_name_result_cb_t cb)
```

Register a callback function that is called if probing is completed successfully or with a conflict.

4.109 src/include/lwip/apps/mdns_domain.h File Reference

```
#include "lwip/apps/mdns_opts.h"#include "lwip/apps/mdns_priv.h"
```

4.109.1 Functions

- err_t mdns_domain_add_label (struct mdns_domain *domain, const char *label, u8_t len)
- err_t mdns_domain_add_domain (struct mdns_domain *domain, struct mdns_domain *source)
- err_t mdns_domain_add_string (struct mdns_domain *domain, const char *source)
- u16_t mdns_readname (struct pbuf *p, u16_t offset, struct mdns_domain *domain)
- void mdns_domain_debug_print (struct mdns_domain *domain)

- int mdns_domain_eq (struct mdns_domain *a, struct mdns_domain *b)
- err_t mdns_build_reverse_v4_domain (struct mdns_domain *domain, const ip4_addr_t *addr)
- err_t mdns_build_reverse_v6_domain (struct mdns_domain *domain, const ip6_addr_t *addr)
- err_t mdns_build_host_domain (struct mdns_domain *domain, struct mdns_host *mdns)
- err_t mdns_build_dnssd_domain (struct mdns_domain *domain)
- err_t mdns_build_service_domain (struct mdns_domain *domain, struct mdns_service *service, int include_name)
- err_t mdns_build_request_domain (struct mdns_domain *domain, struct mdns_request *request, int include_name)
- u16_t mdns_compress_domain (struct pbuf *pbuf, u16_t *offset, struct mdns_domain *domain)
- err_t mdns_write_domain (struct mdns_outpacket *outpkt, struct mdns_domain *domain)

4.109.2 Detailed Description

MDNS responder - domain related functionalities

4.109.3 Function Documentation

4.109.3.1 mdns_build_dnssd_domain()

```
err_t mdns_build_dnssd_domain (struct mdns_domain * domain)
```

Build the lookup-all-services special DNS-SD domain name

Parameters

domain	Where to write the domain name

Returns ERR_OK if domain _services._dns-sd._udp.local. was written, an err_t otherwise

4.109.3.2 mdns_build_host_domain()

```
err_t mdns_build_host_domain (struct mdns_domain * domain, struct mdns_host * mdns)
```

Build the <hostname>.local. domain name

Parameters

domain	Where to write the domain name
mdns	TMDNS netif descriptor.

Returns ERR_OK if domain <hostname>.local. was written, an err_t otherwise

4.109.3.3 mdns_build_request_domain()

```
err_t mdns_build_request_domain (struct mdns_domain * domain, struct mdns_request * reques
int include_name)
```

Build domain name for a request

Parameters

Returns ERR_OK if domain was written. If service name is included, <name>.<type>.<proto>.local. will be written, otherwise <type>.<proto>.local. An err_t is returned on error.

domain	Where to write the domain name
request	The request struct, containing service name, type and protocol
include_name	Whether to include the service name in the domain

4.109.3.4 mdns_build_reverse_v4_domain()

```
err_t mdns_build_reverse_v4_domain (struct mdns_domain * domain, const ip4_addr_t * addr)
```

Build domain for reverse lookup of IPv4 address like 12.0.168.192.in-addr.arpa. for 192.168.0.12

Parameters

domain	Where to write the domain name
addr	Pointer to an IPv4 address to encode

Returns ERR_OK if domain was written, an err_t otherwise

4.109.3.5 mdns_build_reverse_v6_domain()

```
err_t mdns_build_reverse_v6_domain (struct mdns_domain * domain, const ip6_addr_t * addr)
```

Build domain for reverse lookup of IP address like b.a.9.8.7.6.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. for 2001:db8::567:89ab

Parameters

domain	Where to write the domain name
addr	Pointer to an IPv6 address to encode

Returns ERR_OK if domain was written, an err_t otherwise

4.109.3.6 mdns_build_service_domain()

```
err_t mdns_build_service_domain (struct mdns_domain * domain, struct mdns_service * servic
int include_name)
```

Build domain name for a service

Parameters

Returns ERR_OK if domain was written. If service name is included, <name>.<type>.<proto>.local. will be written, otherwise <type>.<proto>.local. An err_t is returned on error.

4.109.3.7 mdns_compress_domain()

```
u16_t mdns_compress_domain (struct pbuf * pbuf, u16_t * offset, struct mdns_domain * domai
```

Return bytes needed to write before jump for best result of compressing supplied domain against domain in outpacket starting at specified offset. If a match is found, offset is updated to where to jump to

Parameters

Returns Number of bytes to write of the new domain before writing a jump to the offset. If compression can not be done against this previous domain name, the full new domain length is returned.

domain	Where to write the domain name
service	The service struct, containing service name, type and protocol
include_name	Whether to include the service name in the domain

pbuf	Pointer to pbuf with the partially constructed DNS packet
offset	Start position of a domain written earlier. If this location is suitable for compression, the pointer is updated to where in the domain to jump to.
domain	The domain to write

4.109.3.8 mdns_domain_add_domain()

```
err_t mdns_domain_add_domain (struct mdns_domain * domain, struct mdns_domain * source)
```

Add a partial domain to a domain

Parameters

domain	The domain to add a label to
source	The domain to add, like <\x09_services\007_dns-sd\000>

Returns ERR_OK on success, an err_t otherwise if label too long

4.109.3.9 mdns_domain_add_label()

```
err_t mdns_domain_add_label (struct mdns_domain * domain, const char * label, u8_t len)
```

Add a label part to a domain

Parameters

Returns ERR_OK on success, an err_t otherwise if label too long

4.109.3.10 mdns_domain_add_string()

```
err_t mdns_domain_add_string (struct mdns_domain * domain, const char * source)
```

Add a string domain to a domain

Parameters

Returns ERR_OK on success, an err_t otherwise if label too long

4.109.3.11 mdns_domain_debug_print()

```
void mdns_domain_debug_print (struct mdns_domain * domain)
```

Print domain name to debug output

Parameters

4.109.3.12 mdns_domain_eq()

```
int mdns_domain_eq (struct mdns_domain * a, struct mdns_domain * b)
```

Return 1 if contents of domains match (case-insensitive)

Parameters

Returns 1 if domains are equal ignoring case, 0 otherwise

domain	The domain to add a label to
label	The label to add, like <hostname>, 'local', 'com' or "
len	The length of the label

domain	The domain to add a label to
source	The string to add, like <_services._dns-sd>

4.109.3.13 mdns_readname()

```
u16_t mdns_readname (struct pbuf * p, u16_t offset, struct mdns_domain * domain)
```

Read possibly compressed domain name from packet buffer

Parameters

Returns The new offset after the domain, or MDNS_READNAME_ERROR if reading failed

4.109.3.14 mdns_write_domain()

```
err_t mdns_write_domain (struct mdns_outpacket * outpkt, struct mdns_domain * domain)
```

Write domain to outpacket. Compression will be attempted, unless domain->skip_compression is set.

Parameters

Returns ERR_OK on success, an err_t otherwise

4.110 src/include/lwip/apps/mdns_opts.h File Reference

```
#include "lwip/opt.h"
```

4.110.1 Macros

- #define MDNS_MAX_SERVICES 1
- #define MDNS_PROBE_DELAY_MS 250
- #define MDNS_MAX_STORED_PKTS 4
- #define MDNS_OUTPUT_PACKET_SIZE ((MDNS_MAX_SERVICES == 1) ? 512 : 1450)
- #define MDNS_RESP_USENETIF_EXTCALLBACK LWIP_NETIF_EXT_STATUS_CALLBACK
- #define LWIP_MDNS_SEARCH 1
- #define MDNS_MAX_REQUESTS 2
- #define MDNS_DEBUG LWIP_DBG_OFF

4.110.2 Detailed Description

MDNS responder

domain	The domain name

a	Domain name to compare 1
b	Domain name to compare 2

p	The packet
offset	start position of domain name in packet
domain	The domain name destination

4.111 src/include/lwip/apps/mdns_out.h File Reference

```
#include "lwip/apps/mdns_opts.h"#include "lwip/apps/mdns_priv.h"#include "lwip/netif.h"# ↩
    include "lwip/timeouts.h"
```

4.111.1 Macros

- #define QUESTION_PROBE_HOST_ANY 0x10

4.111.2 Functions

- err_t mdns_create_outpacket (struct netif *netif, struct mdns_outmsg *msg, struct mdns_outpacket *outpkt)
- err_t mdns_send_outpacket (struct mdns_outmsg *msg, struct netif *netif)
- void mdns_set_timeout (struct netif *netif, u32_t msecs, sys_timeout_handler handler, u8_t *busy_flag)
- void mdns_multicast_timeout_reset_ipv4 (void *arg)
- void mdns_multicast_probe_timeout_reset_ipv4 (void *arg)
- void mdns_multicast_timeout_25ttl_reset_ipv4 (void *arg)
- void mdns_send_multicast_msg_delayed_ipv4 (void *arg)
- void mdns_send_unicast_msg_delayed_ipv4 (void *arg)
- void mdns_start_multicast_timeouts_ipv4 (struct netif *netif)
- void mdns_multicast_timeout_reset_ipv6 (void *arg)
- void mdns_multicast_probe_timeout_reset_ipv6 (void *arg)
- void mdns_multicast_timeout_25ttl_reset_ipv6 (void *arg)
- void mdns_send_multicast_msg_delayed_ipv6 (void *arg)
- void mdns_send_unicast_msg_delayed_ipv6 (void *arg)
- void mdns_start_multicast_timeouts_ipv6 (struct netif *netif)
- void mdns_prepare_txtdata (struct mdns_service *service)
- err_t mdns_send_request (struct mdns_request *req, struct netif *netif, const ip_addr_t *destination)

outpkt	The outpacket to write to
domain	The domain name to write

4.111.3 Detailed Description

MDNS responder - output related functionalities

4.111.4 Macro Definition Documentation

4.111.4.1 QUESTION_PROBE_HOST_ANY

```
#define QUESTION_PROBE_HOST_ANY   0x10
```

Bitmasks outmsg generation

4.111.5 Function Documentation

4.111.5.1 mdns_create_outpacket()

```
err_t mdns_create_outpacket (struct netif * netif, struct mdns_outmsg * msg, struct mdns_o
* outpkt)
```

Create packet with chosen answers as a reply

Add all selected answers / questions Add additional answers based on the selected answers

4.111.5.2 mdns_multicast_probe_timeout_reset_ipv4()

```
void mdns_multicast_probe_timeout_reset_ipv4 (void * arg)
```

Called by timeouts when timer is passed, allows direct multicast IPv4 probe response traffic again and sends out probe response if one was pending

Parameters

arg	pointer to netif of timeout.

4.111.5.3 mdns_multicast_probe_timeout_reset_ipv6()

```
void mdns_multicast_probe_timeout_reset_ipv6 (void * arg)
```

Called by timeouts when timer is passed, allows direct multicast IPv6 probe response traffic again and sends out probe response if one was pending

Parameters

arg	pointer to netif of timeout.

4.111.5.4 mdns_multicast_timeout_25ttl_reset_ipv4()

```
void mdns_multicast_timeout_25ttl_reset_ipv4 (void * arg)
```

Called by timeouts when timer is passed, allows to send an answer on a QU question via multicast.

Parameters

arg	pointer to netif of timeout.

4.111.5.5 mdns_multicast_timeout_25ttl_reset_ipv6()

`void mdns_multicast_timeout_25ttl_reset_ipv6 (void * arg)`

Called by timeouts when timer is passed, allows to send an answer on a QU question via multicast.

Parameters

arg	pointer to netif of timeout.

4.111.5.6 mdns_multicast_timeout_reset_ipv4()

`void mdns_multicast_timeout_reset_ipv4 (void * arg)`

Called by timeouts when timer is passed, allows multicast IPv4 traffic again.

Parameters

arg	pointer to netif of timeout.

4.111.5.7 mdns_multicast_timeout_reset_ipv6()

`void mdns_multicast_timeout_reset_ipv6 (void * arg)`

Called by timeouts when timer is passed, allows multicast IPv6 traffic again.

Parameters

4.111.5.8 mdns_prepare_txtdata()

`void mdns_prepare_txtdata (struct mdns_service * service)`

Call user supplied function to setup TXT data

Parameters

4.111.5.9 mdns_send_multicast_msg_delayed_ipv4()

`void mdns_send_multicast_msg_delayed_ipv4 (void * arg)`

Called by timeouts when timer is passed, sends out delayed multicast IPv4 response.

Parameters

4.111.5.10 mdns_send_multicast_msg_delayed_ipv6()

`void mdns_send_multicast_msg_delayed_ipv6 (void * arg)`

Called by timeouts when timer is passed, sends out delayed multicast IPv6 response.

Parameters

arg	pointer to netif of timeout.

service	The service to build TXT record for

4.111.5.11 mdns_send_outpacket()

`err_t mdns_send_outpacket (struct mdns_outmsg * msg, struct netif * netif)`

Send chosen answers as a reply

Create the packet Send the packet

4.111.5.12 mdns_send_request()

`err_t mdns_send_request (struct mdns_request * req, struct netif * netif, const ip_addr_t * destination)`

Send search request containing all our known data

Parameters

4.111.5.13 mdns_send_unicast_msg_delayed_ipv4()

`void mdns_send_unicast_msg_delayed_ipv4 (void * arg)`

Called by timeouts when timer is passed, sends out delayed unicast IPv4 response.

Parameters

4.111.5.14 mdns_send_unicast_msg_delayed_ipv6()

`void mdns_send_unicast_msg_delayed_ipv6 (void * arg)`

Called by timeouts when timer is passed, sends out delayed unicast IPv6 response.

Parameters

4.111.5.15 mdns_set_timeout()

`void mdns_set_timeout (struct netif * netif, u32_t msecs, sys_timeout_handler handler, u8_t * busy_flag)`

Sets a timer that calls the handler when finished. Depending on the busy_flag the timer is restarted or started. The flag is set before return. Sys_timeout does not give us this functionality.

Parameters

4.111.5.16 mdns_start_multicast_timeouts_ipv4()

`void mdns_start_multicast_timeouts_ipv4 (struct netif * netif)`

Start all multicast timeouts for IPv4 Timeouts started:

arg	pointer to netif of timeout.

arg	pointer to netif of timeout.

req	The request to send
netif	The network interface to send on
destination	The target address to send to (usually multicast address)

- do not multicast within one second
- do not multicast a probe response within 250ms
- send a multicast answer on a QU question if not send recently.

Parameters

4.111.5.17 mdns_start_multicast_timeouts_ipv6()

```
void mdns_start_multicast_timeouts_ipv6 (struct netif * netif)
```

Start all multicast timeouts for IPv6 Timeouts started:

- do not multicast within one second
- do not multicast a probe response within 250ms
- send a multicast answer on a QU question if not send recently.

Parameters

4.112 src/include/lwip/apps/mdns_priv.h File Reference

```
#include "lwip/apps/mdns.h"#include "lwip/apps/mdns_opts.h"#include "lwip/pbuf.h"
```

4.112.1 Data Structures

- struct mdns_request
- struct mdns_service
- struct mdns_outpacket
- struct mdns_outmsg
- struct mdns_delayed_msg
- struct mdns_host

4.112.2 Functions

- struct mdns_host * netif_mdns_data (struct netif *netif)
- struct udp_pcb * get_mdns_pcb (void)

arg	pointer to netif of timeout.

arg	pointer to netif of timeout.

netif	Network interface info
msecs	Time value to set
handler	Callback function to call
busy_flag	Pointer to flag that displays if the timer is running or not.

4.112.3 Detailed Description

MDNS responder private definitions

4.112.4 Function Documentation

4.112.4.1 get_mdns_pcb()

```
struct udp_pcb * get_mdns_pcb (void )
```

Construction to access the mdns udp pcb.

Returns udp_pcb struct of mdns

4.112.4.2 netif_mdns_data()

```
struct mdns_host * netif_mdns_data (struct netif * netif)
```

Construction to make mdns struct accessible from mdns_out.c TODO: can we add the mdns struct to the netif like we do for dhcp, autoip,...? Then this is not needed any more.

Parameters

Returns mdns struct

4.113 src/include/lwip/apps/mqtt.h File Reference

```
#include "lwip/apps/mqtt_opts.h"#include "lwip/err.h"#include "lwip/ip_addr.h"#include " ↩
    lwip/prot/iana.h"
```

4.113.1 Data Structures

- struct mqtt_connect_client_info_t

4.113.2 Macros

- #define MQTT_PORT LWIP_IANA_PORT_MQTT
- #define MQTT_TLS_PORT LWIP_IANA_PORT_SECURE_MQTT
- #define mqtt_subscribe(client, topic, qos, cb, arg)   mqtt_sub_unsub(client, topic, qos, cb, arg, 1)
- #define mqtt_unsubscribe(client, topic, cb, arg)   mqtt_sub_unsub(client, topic, 0, cb, arg, 0)

netif	network interface to start timeouts on

netif	network interface to start timeouts on

netif	The network interface

4.113.3 Typedefs

- typedef void(* mqtt_connection_cb_t) (mqtt_client_t *client, void *arg, mqtt_connection_status_t status)
- typedef void(* mqtt_incoming_data_cb_t) (void *arg, const u8_t *data, u16_t len, u8_t flags)
- typedef void(* mqtt_incoming_publish_cb_t) (void *arg, const char *topic, u32_t tot_len)
- typedef void(* mqtt_request_cb_t) (void *arg, err_t err)

4.113.4 Enumerations

- enum mqtt_connection_status_t { MQTT_CONNECT_ACCEPTED = 0 , MQTT_CONNECT_REFUSED_PROTOCOL_VERSION
  = 1 , MQTT_CONNECT_REFUSED_IDENTIFIER = 2 , MQTT_CONNECT_REFUSED_SERVER = 3 , MQTT_CONNECT_REFU
  = 4 , MQTT_CONNECT_REFUSED_NOT_AUTHORIZED_ = 5 , MQTT_CONNECT_DISCONNECTED = 256 , MQTT_CONNE
  = 257 }
- enum { MQTT_DATA_FLAG_LAST = 1 }

4.113.5 Functions

- err_t mqtt_client_connect (mqtt_client_t *client, const ip_addr_t *ipaddr, u16_t port, mqtt_connection_cb_t cb, void *arg, const struct mqtt_connect_client_info_t *client_info)
- void mqtt_disconnect (mqtt_client_t *client)
- mqtt_client_t * mqtt_client_new (void)
- void mqtt_client_free (mqtt_client_t *client)
- u8_t mqtt_client_is_connected (mqtt_client_t *client)
- void mqtt_set_inpub_callback (mqtt_client_t *client, mqtt_incoming_publish_cb_t pub_cb, mqtt_incoming_data_cb_t data_cb, void *arg)
- err_t mqtt_sub_unsub (mqtt_client_t *client, const char *topic, u8_t qos, mqtt_request_cb_t cb, void *arg, u8_t sub)
- err_t mqtt_publish (mqtt_client_t *client, const char *topic, const void *payload, u16_t payload_length, u8_t qos, u8_t retain, mqtt_request_cb_t cb, void *arg)

4.113.6 Detailed Description

MQTT client

4.114 src/include/lwip/apps/mqtt_opts.h File Reference

```
#include "lwip/opt.h"
```

4.114.1 Macros

- #define MQTT_OUTPUT_RINGBUF_SIZE 256
- #define MQTT_VAR_HEADER_BUFFER_LEN 128
- #define MQTT_REQ_MAX_IN_FLIGHT 4
- #define MQTT_CYCLIC_TIMER_INTERVAL 5
- #define MQTT_REQ_TIMEOUT 30
- #define MQTT_CONNECT_TIMOUT 100

4.114.2 Detailed Description

MQTT client options

4.115 src/include/lwip/apps/mqtt_priv.h File Reference

```
#include "lwip/apps/mqtt.h"#include "lwip/altcp.h"
```

4.115.1 Data Structures

- struct mqtt_request_t
- struct mqtt_ringbuf_t
- struct mqtt_client_s

4.115.2 Detailed Description

MQTT client (private interface)

4.116 src/include/lwip/apps/netbiosns.h File Reference

```
#include "lwip/apps/netbiosns_opts.h"
```

4.116.1 Functions

- void netbiosns_init (void)
- void netbiosns_stop (void)

4.116.2 Detailed Description

NETBIOS name service responder

4.117 src/include/lwip/apps/netbiosns_opts.h File Reference

```
#include "lwip/opt.h"
```

4.117.1 Macros

- #define NETBIOS_LWIP_NAME "NETBIOSLWIPDEV"
- #define LWIP_NETBIOS_RESPOND_NAME_QUERY 0

4.117.2 Detailed Description

NETBIOS name service responder options

4.118 src/include/lwip/apps/snmp.h File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/err.h"#include "lwip/apps/snmp_core.h"
```

4.118.1 Data Structures

- struct snmp_varbind

4.118.2 Macros

- #define SNMP_GENTRAP_COLDSTART 0
- #define SNMP_GENTRAP_WARMSTART 1
- #define SNMP_GENTRAP_LINKDOWN 2
- #define SNMP_GENTRAP_LINKUP 3
- #define SNMP_GENTRAP_AUTH_FAILURE 4
- #define SNMP_GENTRAP_EGP_NEIGHBOR_LOSS 5
- #define SNMP_GENTRAP_ENTERPRISE_SPECIFIC 6

4.118.3 Functions

- void snmp_init (void)
- void snmp_set_mibs (const struct snmp_mib **mibs, u8_t num_mibs)
- void snmp_set_device_enterprise_oid (const struct snmp_obj_id *device_enterprise_oid)
- const struct snmp_obj_id * snmp_get_device_enterprise_oid (void)
- void snmp_trap_dst_enable (u8_t dst_idx, u8_t enable)
- void snmp_trap_dst_ip_set (u8_t dst_idx, const ip_addr_t *dst)
- err_t snmp_send_trap_generic (s32_t generic_trap)

- err_t snmp_send_trap_specific (s32_t specific_trap, struct snmp_varbind *varbinds)
- err_t snmp_send_trap (const struct snmp_obj_id *oid, s32_t generic_trap, s32_t specific_trap, struct snmp_varbind *varbinds)
- err_t snmp_send_inform_generic (s32_t generic_trap, struct snmp_varbind *varbinds, s32_t *ptr_request_id)
- err_t snmp_send_inform_specific (s32_t specific_trap, struct snmp_varbind *varbinds, s32_t *ptr_request_id)
- err_t snmp_send_inform (const struct snmp_obj_id *oid, s32_t generic_trap, s32_t specific_trap, struct snmp_varbind *varbinds, s32_t *ptr_request_id)
- void snmp_set_inform_callback (snmp_inform_callback_fct inform_callback, void *callback_arg)
- void snmp_set_default_trap_version (u8_t snmp_version)
- u8_t snmp_get_default_trap_version (void)
- void snmp_set_auth_traps_enabled (u8_t enable)
- u8_t snmp_get_auth_traps_enabled (void)
- const char * snmp_get_community (void)
- const char * snmp_get_community_write (void)
- const char * snmp_get_community_trap (void)
- void snmp_set_community (const char *const community)
- void snmp_set_community_write (const char *const community)
- void snmp_set_community_trap (const char *const community)
- void snmp_coldstart_trap (void)
- void snmp_authfail_trap (void)
- void snmp_set_write_callback (snmp_write_callback_fct write_callback, void *callback_arg)

4.118.4 Detailed Description

SNMP server main API - start and basic configuration

4.118.5 Macro Definition Documentation

4.118.5.1 SNMP_GENTRAP_AUTH_FAILURE

```
#define SNMP_GENTRAP_AUTH_FAILURE   4
```

Generic trap: authentication failure

4.118.5.2 SNMP_GENTRAP_COLDSTART

```
#define SNMP_GENTRAP_COLDSTART   0
```

Generic trap: cold start

4.118.5.3 SNMP_GENTRAP_EGP_NEIGHBOR_LOSS

```
#define SNMP_GENTRAP_EGP_NEIGHBOR_LOSS   5
```

Generic trap: EGP neighbor lost

4.118.5.4 SNMP_GENTRAP_ENTERPRISE_SPECIFIC

```
#define SNMP_GENTRAP_ENTERPRISE_SPECIFIC   6
```

Generic trap: enterprise specific

4.118.5.5 SNMP_GENTRAP_LINKDOWN

```
#define SNMP_GENTRAP_LINKDOWN   2
```

Generic trap: link down

4.118.5.6 SNMP_GENTRAP_LINKUP

```
#define SNMP_GENTRAP_LINKUP   3
```

Generic trap: link up

4.118.5.7 SNMP_GENTRAP_WARMSTART

```
#define SNMP_GENTRAP_WARMSTART   1
```

Generic trap: warm start

4.119 src/include/lwip/snmp.h File Reference

```
#include "lwip/opt.h"#include "lwip/ip_addr.h"
```

4.119.1 Macros

- #define MIB2_COPY_SYSUPTIME_TO(ptrToVal)   (*(ptrToVal) = (sys_now() / 10))
- #define MIB2_STATS_NETIF_INC(n, x)   do { ++(n)->mib2_counters.x; } while(0)
- #define MIB2_STATS_NETIF_ADD(n, x, val)   do { (n)->mib2_counters.x += (val); } while(0)
- #define MIB2_INIT_NETIF(netif, type, speed)

4.119.2 Enumerations

- enum snmp_ifType

4.119.3 Detailed Description

SNMP support API for implementing netifs and statistics for MIB2

4.119.4 Macro Definition Documentation

4.119.4.1 MIB2_COPY_SYSUPTIME_TO

```
#define MIB2_COPY_SYSUPTIME_TO( ptrToVal)   (*(ptrToVal) = (sys_now() / 10))
```

This macro has a precision of ~49 days because sys_now returns u32_t. #define your own if you want ~490 days.

4.120 src/include/lwip/apps/snmp_core.h File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/ip_addr.h"#include "lwip/err.h"
```

4.120.1 Data Structures

- struct snmp_obj_id
- union snmp_variant_value
- struct snmp_node
- struct snmp_node_instance
- struct snmp_tree_node
- struct snmp_leaf_node
- struct snmp_mib
- struct snmp_oid_range
- struct snmp_next_oid_state

4.120.2 Macros

- #define SNMP_NODE_TREE 0x00

4.120.3 Enumerations

- enum snmp_err_t
- enum snmp_access_t

4.120.4 Functions

- u8_t snmp_oid_in_range (const u32_t *oid_in, u8_t oid_len, const struct snmp_oid_range *oid_ranges, u8_t oid_ranges_len)
- void snmp_next_oid_init (struct snmp_next_oid_state *state, const u32_t *start_oid, u8_t start_oid_len, u32_t *next_oid_buf, u8_t next_oid_max_len)
- u8_t snmp_next_oid_precheck (struct snmp_next_oid_state *state, const u32_t *oid, u8_t oid_len)
- u8_t snmp_next_oid_check (struct snmp_next_oid_state *state, const u32_t *oid, u8_t oid_len, void *reference)
- void snmp_oid_assign (struct snmp_obj_id *target, const u32_t *oid, u8_t oid_len)
- void snmp_oid_combine (struct snmp_obj_id *target, const u32_t *oid1, u8_t oid1_len, const u32_t *oid2, u8_t oid2_len)
- void snmp_oid_prefix (struct snmp_obj_id *target, const u32_t *oid, u8_t oid_len)
- void snmp_oid_append (struct snmp_obj_id *target, const u32_t *oid, u8_t oid_len)
- u8_t snmp_oid_equal (const u32_t *oid1, u8_t oid1_len, const u32_t *oid2, u8_t oid2_len)
- s8_t snmp_oid_compare (const u32_t *oid1, u8_t oid1_len, const u32_t *oid2, u8_t oid2_len)
- u8_t snmp_oid_to_ip4 (const u32_t *oid, ip4_addr_t *ip)

- void snmp_ip4_to_oid (const ip4_addr_t *ip, u32_t *oid)
- u8_t snmp_oid_to_ip6 (const u32_t *oid, ip6_addr_t *ip)
- void snmp_ip6_to_oid (const ip6_addr_t *ip, u32_t *oid)
- u8_t snmp_ip_to_oid (const ip_addr_t *ip, u32_t *oid)
- u8_t snmp_ip_port_to_oid (const ip_addr_t *ip, u16_t port, u32_t *oid)
- u8_t snmp_oid_to_ip (const u32_t *oid, u8_t oid_len, ip_addr_t *ip)
- u8_t snmp_oid_to_ip_port (const u32_t *oid, u8_t oid_len, ip_addr_t *ip, u16_t *port)
- u8_t netif_to_num (const struct netif *netif)
- err_t snmp_decode_bits (const u8_t *buf, u32_t buf_len, u32_t *bit_value)
- u8_t snmp_encode_bits (u8_t *buf, u32_t buf_len, u32_t bit_value, u8_t bit_count)

4.120.5 Detailed Description

SNMP core API for implementing MIBs

4.120.6 Macro Definition Documentation

4.120.6.1 SNMP_NODE_TREE

```
#define SNMP_NODE_TREE   0x00
```

SNMP MIB node types tree node is the only node the stack can process in order to walk the tree, all other nodes are assumed to be leaf nodes. This cannot be an enum because users may want to define their own node types.

4.120.7 Enumeration Type Documentation

4.120.7.1 snmp_access_t

```
enum snmp_access_t
```

SNMP node instance access types

4.120.7.2 snmp_err_t

```
enum snmp_err_t
```

error codes predefined by SNMP prot.

4.120.8 Function Documentation

4.120.8.1 netif_to_num()

```
u8_t netif_to_num (const struct netif * netif)
```

Convert netif to interface index

Parameters

Returns index

netif	netif

4.120.8.2 snmp_decode_bits()

```
err_t snmp_decode_bits (const u8_t * buf, u32_t buf_len, u32_t * bit_value)
```

Decodes BITS pseudotype value from ASN.1 OctetString.

Note
Because BITS pseudo type is encoded as OCTET STRING, it cannot directly be encoded/decoded by the agent. Instead call this function as required from get/test/set methods.

Parameters

buf	points to a buffer holding the ASN1 octet string
buf_len	length of octet string
bit_value	decoded Bit value with Bit0 == LSB

Returns ERR_OK if successful, ERR_ARG if bit value contains more than 32 bit

4.120.8.3 snmp_encode_bits()

```
u8_t snmp_encode_bits (u8_t * buf, u32_t buf_len, u32_t bit_value, u8_t bit_count)
```

Encodes BITS pseudotype value into ASN.1 OctetString.

Note
Because BITS pseudo type is encoded as OCTET STRING, it cannot directly be encoded/decoded by the agent. Instead call this function as required from get/test/set methods.

Parameters

buf	points to a buffer where the resulting ASN1 octet string is stored to
buf_len	max length of the buffer
bit_value	Bit value to encode with Bit0 == LSB
bit_count	Number of possible bits for the bit value (according to rfc we have to send all bits independent from their truth value)

Returns number of bytes used from buffer to store the resulting OctetString

4.120.8.4 snmp_ip4_to_oid()

```
void snmp_ip4_to_oid (const ip4_addr_t * ip, u32_t * oid)
```

Convert ip4_addr to InetAddressIPv4 (no InetAddressType)

Parameters

ip	points to input struct
oid	points to u32_t ident[4] output

ip	points to input struct
oid	points to u32_t ident[16] output

4.120.8.5 snmp_ip6_to_oid()

```
void snmp_ip6_to_oid (const ip6_addr_t * ip, u32_t * oid)
```

Convert ip6_addr to InetAddressIPv6 (no InetAddressType)

Parameters

4.120.8.6 snmp_ip_port_to_oid()

```
u8_t snmp_ip_port_to_oid (const ip_addr_t * ip, u16_t port, u32_t * oid)
```

Convert to InetAddressType+InetAddress+InetPortNumber

Parameters

ip	IP address
port	Port
oid	OID

Returns OID length

4.120.8.7 snmp_ip_to_oid()

```
u8_t snmp_ip_to_oid (const ip_addr_t * ip, u32_t * oid)
```

Convert to InetAddressType+InetAddress

Parameters

Returns OID length

4.120.8.8 snmp_next_oid_check()

```
u8_t snmp_next_oid_check (struct snmp_next_oid_state * state, const u32_t * oid, u8_t oid_
void * reference)
```

checks the passed OID if it is a candidate to be the next one (get_next); returns !=0 if passed oid is currently closest, otherwise 0

4.120.8.9 snmp_next_oid_init()

```
void snmp_next_oid_init (struct snmp_next_oid_state * state, const u32_t * start_oid, u8_t
start_oid_len, u32_t * next_oid_buf, u8_t next_oid_max_len)
```

initialize struct next_oid_state using this function before passing it to next_oid_check

ip	IP address
oid	OID

4.120.8.10 snmp_next_oid_precheck()

```
u8_t snmp_next_oid_precheck (struct snmp_next_oid_state * state, const u32_t * oid, u8_t
oid_len)
```

checks if the passed incomplete OID may be a possible candidate for snmp_next_oid_check(); this method is intended if the complete OID is not yet known but it is very expensive to build it up, so it is possible to test the starting part before building up the complete oid and pass it to snmp_next_oid_check()

4.120.8.11 snmp_oid_append()

```
void snmp_oid_append (struct snmp_obj_id * target, const u32_t * oid, u8_t oid_len)
```

Append OIDs to struct snmp_obj_id **Parameters**

target	Assignment target to append to
oid	OID
oid_len	OID length

4.120.8.12 snmp_oid_assign()

```
void snmp_oid_assign (struct snmp_obj_id * target, const u32_t * oid, u8_t oid_len)
```

Assign an OID to struct snmp_obj_id **Parameters**

target	Assignment target
oid	OID
oid_len	OID length

4.120.8.13 snmp_oid_combine()

```
void snmp_oid_combine (struct snmp_obj_id * target, const u32_t * oid1, u8_t oid1_len,
const u32_t * oid2, u8_t oid2_len)
```

Combine two OIDs into struct snmp_obj_id **Parameters**

4.120.8.14 snmp_oid_compare()

```
s8_t snmp_oid_compare (const u32_t * oid1, u8_t oid1_len, const u32_t * oid2, u8_t oid2_le
```

Compare two OIDs

Parameters

Returns -1: OID1<OID2 1: OID1 >OID2 0: equal

target	Assignment target
oid1	OID 1
oid1_len	OID 1 length
oid2	OID 2
oid2_len	OID 2 length

oid1	OID 1
oid1_len	OID 1 length
oid2	OID 2
oid2_len	OID 2 length

4.120.8.15 snmp_oid_equal()

`u8_t snmp_oid_equal (const u32_t * oid1, u8_t oid1_len, const u32_t * oid2, u8_t oid2_len)`

Check of two OIDs are equal

Parameters

Returns 1: equal 0: non-equal

4.120.8.16 snmp_oid_in_range()

`u8_t snmp_oid_in_range (const u32_t * oid_in, u8_t oid_len, const struct snmp_oid_range * oid_ranges, u8_t oid_ranges_len)`

checks if incoming OID length and values are in allowed ranges

4.120.8.17 snmp_oid_prefix()

`void snmp_oid_prefix (struct snmp_obj_id * target, const u32_t * oid, u8_t oid_len)`

Prefix an OID to OID in struct snmp_obj_id **Parameters**

4.120.8.18 snmp_oid_to_ip()

`u8_t snmp_oid_to_ip (const u32_t * oid, u8_t oid_len, ip_addr_t * ip)`

Convert from InetAddressType+InetAddress to ip_addr_t

Parameters

Returns Parsed OID length

4.120.8.19 snmp_oid_to_ip4()

`u8_t snmp_oid_to_ip4 (const u32_t * oid, ip4_addr_t * ip)`

Conversion from InetAddressIPv4 oid to lwIP ip4_addr **Parameters**

oid1	OID 1
oid1_len	OID 1 length
oid2	OID 2
oid2_len	OID 2 length

target	Assignment target to prefix
oid	OID
oid_len	OID length

oid	OID
oid_len	OID length
ip	IP address

4.120.8.20 snmp_oid_to_ip6()

```
u8_t snmp_oid_to_ip6 (const u32_t * oid, ip6_addr_t * ip)
```

Conversion from InetAddressIPv6 oid to lwIP ip6_addr **Parameters**

4.120.8.21 snmp_oid_to_ip_port()

```
u8_t snmp_oid_to_ip_port (const u32_t * oid, u8_t oid_len, ip_addr_t * ip, u16_t * port)
```

Convert from InetAddressType+InetAddress+InetPortNumber to ip_addr_t and u16_t

Parameters

Returns Parsed OID length

4.121 src/include/lwip/apps/snmp_mib2.h File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/apps/snmp_core.h"#include "lwip/apps/ ↩
    snmp_threadsync.h"
```

4.121.1 Functions

- void snmp_mib2_set_sysdescr (const u8_t *str, const u16_t *len)
- void snmp_mib2_set_syscontact (u8_t *ocstr, u16_t *ocstrlen, u16_t bufsize)
- void snmp_mib2_set_syscontact_readonly (const u8_t *ocstr, const u16_t *ocstrlen)
- void snmp_mib2_set_sysname (u8_t *ocstr, u16_t *ocstrlen, u16_t bufsize)
- void snmp_mib2_set_sysname_readonly (const u8_t *ocstr, const u16_t *ocstrlen)
- void snmp_mib2_set_syslocation (u8_t *ocstr, u16_t *ocstrlen, u16_t bufsize)
- void snmp_mib2_set_syslocation_readonly (const u8_t *ocstr, const u16_t *ocstrlen)

4.121.2 Detailed Description

SNMP MIB2 API

oid	points to u32_t ident[4] input
ip	points to output struct

oid	points to u32_t oid[16] input
ip	points to output struct

oid	OID
oid_len	OID length
ip	IP address
port	Port

4.122 src/include/lwip/apps/snmp_opts.h File Reference

```
#include "lwip/opt.h"
```

4.122.1 Macros

- #define LWIP_SNMP 0
- #define SNMP_USE_NETCONN 0
- #define SNMP_USE_RAW 1
- #define SNMP_STACK_SIZE DEFAULT_THREAD_STACKSIZE
- #define SNMP_THREAD_PRIO DEFAULT_THREAD_PRIO
- #define SNMP_TRAP_DESTINATIONS 1
- #define SNMP_SAFE_REQUESTS 1
- #define SNMP_MAX_OCTET_STRING_LEN 127
- #define SNMP_MAX_OBJ_ID_LEN 50
- #define SNMP_MIN_VALUE_SIZE (2 * sizeof(u32_t*)) /* size required to store the basic types (8 bytes for counter64) */
- #define SNMP_MAX_VALUE_SIZE LWIP_MAX(LWIP_MAX((SNMP_MAX_OCTET_STRING_LEN), sizeof(u32_t)*(SNMP_
  SNMP_MIN_VALUE_SIZE)
- #define SNMP_COMMUNITY "public"
- #define SNMP_COMMUNITY_WRITE "private"
- #define SNMP_COMMUNITY_TRAP "public"
- #define SNMP_MAX_COMMUNITY_STR_LEN LWIP_MAX(LWIP_MAX(sizeof(SNMP_COMMUNITY), sizeof(SNMP_COM
  sizeof(SNMP_COMMUNITY_TRAP))
- #define SNMP_LWIP_ENTERPRISE_OID 26381
- #define SNMP_DEVICE_ENTERPRISE_OID {1, 3, 6, 1, 4, 1, SNMP_LWIP_ENTERPRISE_OID}
- #define SNMP_DEVICE_ENTERPRISE_OID_LEN 7
- #define SNMP_DEBUG LWIP_DBG_OFF
- #define SNMP_MIB_DEBUG LWIP_DBG_OFF
- #define SNMP_LWIP_MIB2 LWIP_SNMP
- #define SNMP_LWIP_MIB2_SYSDESC "lwIP"

- #define SNMP_LWIP_MIB2_SYSNAME "FQDN-unk"
- #define SNMP_LWIP_MIB2_SYSCONTACT ""
- #define SNMP_LWIP_MIB2_SYSLOCATION ""
- #define SNMP_LWIP_GETBULK_MAX_REPETITIONS 0
- #define LWIP_SNMP_V3 0

4.122.2 Detailed Description

SNMP server options list

4.122.3 Macro Definition Documentation

4.122.3.1 LWIP_SNMP_V3

```
#define LWIP_SNMP_V3    0
```

LWIP_SNMP_V3==1: This enables EXPERIMENTAL SNMPv3 support. LWIP_SNMP must also be enabled. THIS IS UNDER DEVELOPMENT AND SHOULD NOT BE ENABLED IN PRODUCTS.

4.123 src/include/lwip/apps/snmp_scalar.h File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/apps/snmp_core.h"
```

4.123.1 Data Structures

- struct snmp_scalar_node
- struct snmp_scalar_array_node_def
- struct snmp_scalar_array_node

4.123.2 Detailed Description

SNMP server MIB API to implement scalar nodes

4.124 src/include/lwip/apps/snmp_table.h File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/apps/snmp_core.h"
```

4.124.1 Data Structures

- struct snmp_table_col_def
- struct snmp_table_node
- struct snmp_table_simple_node

4.124.2 Enumerations

- enum snmp_table_column_data_type_t

4.124.3 Detailed Description

SNMP server MIB API to implement table nodes

4.124.4 Enumeration Type Documentation

4.124.4.1 snmp_table_column_data_type_t

`enum snmp_table_column_data_type_t`

simple read-only table

4.125 src/include/lwip/apps/snmp_threadsync.h File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/apps/snmp_core.h"#include "lwip/sys.h"
```

4.125.1 Data Structures

- struct threadsync_data
- struct snmp_threadsync_instance
- struct snmp_threadsync_node

4.125.2 Macros

- #define SNMP_CREATE_THREAD_SYNC_NODE(oid, target_leaf_node, threadsync_instance)

4.125.3 Functions

- void snmp_threadsync_init (struct snmp_threadsync_instance *instance, snmp_threadsync_synchronizer_fn sync_fn)

4.125.4 Detailed Description

SNMP server MIB API to implement thread synchronization

4.125.5 Macro Definition Documentation

4.125.5.1 SNMP_CREATE_THREAD_SYNC_NODE

`#define SNMP_CREATE_THREAD_SYNC_NODE( oid, target_leaf_node, threadsync_instance)` **Value:**

```
  {{{ SNMP_NODE_THREADSYNC, (oid) }, \
    snmp_threadsync_get_instance, \
    snmp_threadsync_get_next_instance }, \
    (target_leaf_node), \
    (threadsync_instance) }
```

Create thread sync proxy node

4.125.6 Function Documentation

4.125.6.1 snmp_threadsync_init()

```
void snmp_threadsync_init (struct snmp_threadsync_instance * instance, snmp_threadsync_syn
sync_fn)
```

Create thread sync instance data

Initializes thread synchronization instance

4.126 src/include/lwip/apps/snmpv3.h File Reference

```
#include "lwip/apps/snmp_opts.h"#include "lwip/err.h"
```

4.126.1 Detailed Description

Additional SNMPv3 functionality RFC3414 and RFC3826.

4.127 src/include/lwip/apps/sntp.h File Reference

```
#include "lwip/apps/sntp_opts.h"#include "lwip/ip_addr.h"
```

4.127.1 Functions

- void sntp_setoperatingmode (u8_t operating_mode)
- u8_t sntp_getoperatingmode (void)
- void sntp_init (void)
- void sntp_stop (void)
- u8_t sntp_enabled (void)
- void sntp_setserver (u8_t idx, const ip_addr_t *addr)
- const ip_addr_t * sntp_getserver (u8_t idx)
- u8_t sntp_getkodreceived (u8_t idx)
- u8_t sntp_getreachability (u8_t idx)

4.127.2 Detailed Description

SNTP client API

4.128 src/include/lwip/apps/sntp_opts.h File Reference

```
#include "lwip/opt.h"#include "lwip/prot/iana.h"
```

4.128.1 Macros

- #define SNTP_SET_SYSTEM_TIME(sec)   LWIP_UNUSED_ARG(sec)
- #define SNTP_MAX_SERVERS LWIP_DHCP_MAX_NTP_SERVERS
- #define SNTP_GET_SERVERS_FROM_DHCP LWIP_DHCP_GET_NTP_SRV
- #define SNTP_GET_SERVERS_FROM_DHCPV6 LWIP_DHCP6_GET_NTP_SRV
- #define SNTP_SERVER_DNS   0
- #define SNTP_DEBUG LWIP_DBG_OFF
- #define SNTP_PORT LWIP_IANA_PORT_SNTP
- #define SNTP_CHECK_RESPONSE   0
- #define SNTP_COMP_ROUNDTRIP   0
- #define SNTP_STARTUP_DELAY   1
- #define SNTP_STARTUP_DELAY_FUNC   (LWIP_RAND() % 5000)
- #define SNTP_RECV_TIMEOUT   15000
- #define SNTP_UPDATE_DELAY   3600000
- #define SNTP_GET_SYSTEM_TIME(sec, us)   do { (sec) = 0; (us) = 0; } while(0)
- #define SNTP_RETRY_TIMEOUT SNTP_RECV_TIMEOUT
- #define SNTP_RETRY_TIMEOUT_MAX   (SNTP_RETRY_TIMEOUT * 10)
- #define SNTP_RETRY_TIMEOUT_EXP   1
- #define SNTP_MONITOR_SERVER_REACHABILITY   1

4.128.2 Detailed Description

SNTP client options list

4.129 src/include/lwip/apps/tftp_client.h File Reference

```
#include "lwip/apps/tftp_common.h"
```

4.129.1 Functions

- err_t tftp_init_client (const struct tftp_context *ctx)

4.129.2 Detailed Description

TFTP client header

4.130 src/include/lwip/apps/tftp_common.h File Reference

Trivial File Transfer Protocol (RFC 1350)

```
#include "lwip/apps/tftp_opts.h"#include "lwip/err.h"#include "lwip/pbuf.h"#include "lwip/ ↩
    ip_addr.h"
```

4.130.1 Data Structures

- struct tftp_context

4.130.2 Functions

- err_t tftp_init_common (u8_t mode, const struct tftp_context *ctx)
- void tftp_cleanup (void)

4.130.3 Detailed Description

Trivial File Transfer Protocol (RFC 1350)

Author Logan Gunthorpe logang@deltatee.com

Copyright (c) Deltatee Enterprises Ltd. 2013 All rights reserved.

4.130.4 Function Documentation

4.130.4.1 tftp_init_common()

```
err_t tftp_init_common (u8_t mode, const struct tftp_context * ctx)
```

Initialize TFTP client/server.

Parameters

mode	TFTP mode (client/server)
ctx	TFTP callback struct

4.131 src/include/lwip/apps/tftp_opts.h File Reference

Trivial File Transfer Protocol (RFC 1350) implementation options.

```
#include "lwip/opt.h"#include "lwip/prot/iana.h"
```

4.131.1 Macros

- #define TFTP_DEBUG LWIP_DBG_OFF
- #define TFTP_PORT LWIP_IANA_PORT_TFTP
- #define TFTP_TIMEOUT_MSECS 10000
- #define TFTP_MAX_RETRIES 5
- #define TFTP_TIMER_MSECS (TFTP_TIMEOUT_MSECS / 10)
- #define TFTP_MAX_FILENAME_LEN 20
- #define TFTP_MAX_MODE_LEN 10

4.131.2 Detailed Description

Trivial File Transfer Protocol (RFC 1350) implementation options.

Author Logan Gunthorpe logang@deltatee.com

Copyright (c) Deltatee Enterprises Ltd. 2013 All rights reserved.

4.132 src/include/lwip/apps/tftp_server.h File Reference

```
#include "lwip/apps/tftp_common.h"
```

4.132.1 Functions

- err_t tftp_init_server (const struct tftp_context *ctx)

4.132.2 Detailed Description

TFTP server header

4.133 src/include/lwip/arch.h File Reference

```
#include "arch/cc.h"#include <stdio.h>#include <stdlib.h>#include <stddef.h>#include < ↩
    stdint.h>#include <inttypes.h>#include <limits.h>#include <ctype.h>
```

4.133.1 Macros

- #define BYTE_ORDER LITTLE_ENDIAN
- #define LWIP_RAND() ((u32_t)rand())
- #define LWIP_PLATFORM_DIAG(x) do {printf x;} while(0)
- #define LWIP_PLATFORM_ASSERT(x)
- #define LWIP_NO_STDDEF_H 0
- #define LWIP_NO_STDINT_H 0

- #define LWIP_NO_INTTYPES_H 0
- #define LWIP_NO_LIMITS_H 0
- #define LWIP_NO_CTYPE_H 0
- #define LWIP_CONST_CAST(target_type, val) ((target_type)((ptrdiff_t)val))
- #define LWIP_ALIGNMENT_CAST(target_type, val) LWIP_CONST_CAST(target_type, val)
- #define LWIP_PTR_NUMERIC_CAST(target_type, val) LWIP_CONST_CAST(target_type, val)
- #define LWIP_PACKED_CAST(target_type, val) LWIP_CONST_CAST(target_type, val)
- #define LWIP_DECLARE_MEMORY_ALIGNED(variable_name, size) u8_t variable_name[LWIP_MEM_ALIGN_BUFFER(size)
- #define LWIP_MEM_ALIGN_SIZE(size) (((size) + MEM_ALIGNMENT - 1U) & ~(MEM_ALIGNMENT-1U))
- #define LWIP_MEM_ALIGN_BUFFER(size) (((size) + MEM_ALIGNMENT - 1U))
- #define LWIP_MEM_ALIGN(addr) ((void *)(((mem_ptr_t)(addr) + MEM_ALIGNMENT - 1) & ~(mem_ptr_t)(MEM_ALIGNMEN
  1)))
- #define PACK_STRUCT_BEGIN
- #define PACK_STRUCT_END
- #define PACK_STRUCT_STRUCT
- #define PACK_STRUCT_FIELD(x) x
- #define PACK_STRUCT_FLD_8(x) PACK_STRUCT_FIELD(x)
- #define PACK_STRUCT_FLD_S(x) PACK_STRUCT_FIELD(x)
- #define PACK_STRUCT_USE_INCLUDES
- #define LWIP_UNUSED_ARG(x) (void)x
- #define LWIP_PROVIDE_ERRNO

4.133.2 Detailed Description

Support for different processor and compiler architectures

4.134 src/include/lwip/autoip.h File Reference

```
#include "lwip/opt.h"#include "lwip/netif.h"#include "lwip/etharp.h"#include "lwip/acd.h"
```

4.134.1 Data Structures

- struct autoip

4.134.2 Functions

- void autoip_set_struct (struct netif *netif, struct autoip *autoip)
- void autoip_remove_struct (struct netif *netif)
- err_t autoip_start (struct netif *netif)
- err_t autoip_stop (struct netif *netif)
- void autoip_network_changed_link_up (struct netif *netif)
- void autoip_network_changed_link_down (struct netif *netif)
- u8_t autoip_supplied_address (struct netif *netif)

4.134.3 Detailed Description

AutoIP Automatic LinkLocal IP Configuration

4.134.4 Function Documentation

4.134.4.1 autoip_network_changed_link_down()

```
void autoip_network_changed_link_down (struct netif * netif)
```

Handle a possible change in the network configuration: link down

If there is an AutoIP address configured and AutoIP is in cooperation with DHCP, then stop the autoip module. When the link goes up, we do not want the autoip module to start again. DHCP will initiate autoip when needed.

4.134.4.2 autoip_network_changed_link_up()

```
void autoip_network_changed_link_up (struct netif * netif)
```

Handle a possible change in the network configuration: link up

If there is an AutoIP address configured and AutoIP is not in cooperation with DHCP, start probing for previous address.

4.134.4.3 autoip_supplied_address()

```
u8_t autoip_supplied_address (struct netif * netif)
```

check if AutoIP supplied netif->ip_addr

Parameters

netif	the netif to check

Returns 1 if AutoIP supplied netif->ip_addr (state BOUND), 0 otherwise

4.135 src/include/lwip/prot/autoip.h File Reference

4.135.1 Detailed Description

AutoIP protocol definitions

4.136 src/include/lwip/debug.h File Reference

```
#include "lwip/arch.h"#include "lwip/opt.h"
```

4.136.1 Macros

- #define LWIP_NOASSERT
- #define LWIP_DEBUG

4.136.2 Debug level (LWIP_DBG_MIN_LEVEL)

- #define LWIP_DBG_LEVEL_ALL 0x00
- #define LWIP_DBG_LEVEL_WARNING 0x01
- #define LWIP_DBG_LEVEL_SERIOUS 0x02
- #define LWIP_DBG_LEVEL_SEVERE 0x03

4.136.3 Enable/disable debug messages completely (LWIP_DBG_TYPES_ON)

- #define LWIP_DBG_ON 0x80U
- #define LWIP_DBG_OFF 0x00U

4.136.4 Debug message types (LWIP_DBG_TYPES_ON)

- #define LWIP_DBG_TRACE 0x40U
- #define LWIP_DBG_STATE 0x20U
- #define LWIP_DBG_FRESH 0x10U
- #define LWIP_DBG_HALT 0x08U

4.136.5 Detailed Description

Debug messages infrastructure

4.136.6 Macro Definition Documentation

4.136.6.1 LWIP_DEBUG

```
#define LWIP_DEBUG
```

Enable debug message printing, but only if debug message type is enabled AND is of correct type AND is at least LWIP_DBG_LEVEL.

4.137 src/include/lwip/def.h File Reference

```
#include "lwip/arch.h"#include "lwip/opt.h"
```

4.137.1 Macros

- #define LWIP_MAKEU32(a, b, c, d)

4.137.2 Functions

- u16_t lwip_htons (u16_t x)
- u32_t lwip_htonl (u32_t x)
- void lwip_itoa (char *result, size_t bufsize, int number)
- int lwip_strnicmp (const char *str1, const char *str2, size_t len)
- int lwip_stricmp (const char *str1, const char *str2)
- char * lwip_strnstr (const char *buffer, const char *token, size_t n)
- char * lwip_strnistr (const char *buffer, const char *token, size_t n)
- int lwip_memcmp_consttime (const void *s1, const void *s2, size_t len)

4.137.3 Detailed Description

various utility macros

4.137.4 Macro Definition Documentation

4.137.4.1 LWIP_MAKEU32

`#define LWIP_MAKEU32( a, b, c, d)` **Value:**

```
(((u32_t)((a) & 0xff) << 24) | \
((u32_t)((b) & 0xff) << 16) | \
((u32_t)((c) & 0xff) << 8)  | \
 (u32_t)((d) & 0xff))
```

Create u32_t value from bytes

4.137.5 Function Documentation

4.137.5.1 lwip_htonl()

`u32_t lwip_htonl (u32_t n)`

Convert an u32_t from host- to network byte order.

Parameters

n	u32_t in host byte order

Returns n in network byte order

n	u16_t in host byte order

4.137.5.2 lwip_htons()

```
u16_t lwip_htons (u16_t n)
```

Convert an u16_t from host- to network byte order.

Parameters

Returns n in network byte order

4.138 src/include/lwip/dhcp.h File Reference

```
#include "lwip/opt.h"#include "lwip/netif.h"#include "lwip/udp.h"#include "lwip/acd.h"
```

4.138.1 Macros

- #define DHCP_COARSE_TIMER_SECS   60
- #define DHCP_COARSE_TIMER_MSECS   (DHCP_COARSE_TIMER_SECS * 1000UL)
- #define DHCP_FINE_TIMER_MSECS   500
- #define dhcp_remove_struct(netif)   netif_set_client_data(netif, LWIP_NETIF_CLIENT_DATA_INDEX_DHCP, NULL)

4.138.2 Typedefs

- typedef u16_t dhcp_timeout_t

4.138.3 Functions

- void dhcp_set_struct (struct netif *netif, struct dhcp *dhcp)
- void dhcp_cleanup (struct netif *netif)
- err_t dhcp_start (struct netif *netif)
- err_t dhcp_renew (struct netif *netif)
- err_t dhcp_release (struct netif *netif)
- void dhcp_stop (struct netif *netif)
- void dhcp_release_and_stop (struct netif *netif)
- void dhcp_inform (struct netif *netif)
- void dhcp_network_changed_link_up (struct netif *netif)
- u8_t dhcp_supplied_address (const struct netif *netif)
- void dhcp_coarse_tmr (void)
- void dhcp_fine_tmr (void)

4.138.4 Detailed Description

DHCP client API

4.138.5 Macro Definition Documentation

4.138.5.1 DHCP_COARSE_TIMER_MSECS

```
#define DHCP_COARSE_TIMER_MSECS   (DHCP_COARSE_TIMER_SECS * 1000UL)
```

period (in milliseconds) of the application calling dhcp_coarse_tmr()

4.138.5.2 DHCP_COARSE_TIMER_SECS

```
#define DHCP_COARSE_TIMER_SECS   60
```

period (in seconds) of the application calling dhcp_coarse_tmr()

4.138.5.3 DHCP_FINE_TIMER_MSECS

```
#define DHCP_FINE_TIMER_MSECS   500
```

period (in milliseconds) of the application calling dhcp_fine_tmr()

4.138.5.4 dhcp_remove_struct

```
#define dhcp_remove_struct( netif)   netif_set_client_data(netif, LWIP_NETIF_CLIENT_DATA_I
NULL)
```

Remove a struct dhcp previously set to the netif using dhcp_set_struct()

4.138.6 Typedef Documentation

4.138.6.1 dhcp_timeout_t

```
typedef u16_t dhcp_timeout_t
```

Define DHCP_TIMEOUT_SIZE_T in opt.h if you want use a different integer than u16_t. Especially useful if DHCP_COARSE_TIMER
is in smaller units, so timeouts easily reach UINT16_MAX and more

4.138.7 Function Documentation

4.138.7.1 dhcp_coarse_tmr()

```
void dhcp_coarse_tmr (void )
```

The DHCP timer that checks for lease renewal/rebind timeouts. Must be called once a minute (see DHCP_COARSE_TIMER_SECS).

4.138.7.2 dhcp_fine_tmr()

```
void dhcp_fine_tmr (void )
```

DHCP transaction timeout handling (this function must be called every 500ms, see DHCP_FINE_TIMER_MSECS).

A DHCP server is expected to respond within a short period of time. This timer checks whether an outstanding DHCP request is timed out.

4.138.7.3 dhcp_network_changed_link_up()

```
void dhcp_network_changed_link_up (struct netif * netif)
```

Handle a possible change in the network configuration.

This enters the REBOOTING state to verify that the currently bound address is still valid.

4.138.7.4 dhcp_supplied_address()

```
u8_t dhcp_supplied_address (const struct netif * netif)
```

check if DHCP supplied netif->ip_addr

Parameters

netif	the netif to check

Returns 1 if DHCP supplied netif->ip_addr (states BOUND or RENEWING), 0 otherwise

4.139 src/include/lwip/prot/dhcp.h File Reference

```
#include "lwip/opt.h"#include "lwip/arch.h"#include "lwip/prot/ip4.h"#include "arch/ ↩
    bpstruct.h"#include "arch/epstruct.h"
```

4.139.1 Data Structures

- struct dhcp_msg

4.139.2 Macros

- #define DHCP_OPTIONS_LEN   DHCP_MIN_OPTIONS_LEN

4.139.3 Detailed Description

DHCP protocol definitions

4.139.4 Macro Definition Documentation

4.139.4.1 DHCP_OPTIONS_LEN

```
#define DHCP_OPTIONS_LEN   DHCP_MIN_OPTIONS_LEN
```

make sure user does not configure this too small allow this to be configured in lwipopts.h, but not too small set this to be sufficient for your options in outgoing DHCP msgs

4.140 src/include/lwip/dhcp6.h File Reference

```
#include "lwip/opt.h"#include "lwip/err.h"#include "lwip/netif.h"
```

4.140.1 Macros

- #define DHCP6_TIMER_MSECS 500
- #define dhcp6_remove_struct(netif) netif_set_client_data(netif, LWIP_NETIF_CLIENT_DATA_INDEX_DHCP6, NULL)

4.140.2 Functions

- void dhcp6_set_struct (struct netif *netif, struct dhcp6 *dhcp6)
- void dhcp6_cleanup (struct netif *netif)
- err_t dhcp6_enable_stateful (struct netif *netif)
- err_t dhcp6_enable_stateless (struct netif *netif)
- void dhcp6_disable (struct netif *netif)
- void dhcp6_tmr (void)
- void dhcp6_nd6_ra_trigger (struct netif *netif, u8_t managed_addr_config, u8_t other_config)

4.140.3 Detailed Description

DHCPv6 client: IPv6 address autoconfiguration as per RFC 3315 (stateful DHCPv6) and RFC 3736 (stateless DHCPv6).

4.140.4 Macro Definition Documentation

4.140.4.1 dhcp6_remove_struct

```
#define dhcp6_remove_struct( netif)    netif_set_client_data(netif, LWIP_NETIF_CLIENT_DATA_
NULL)
```

Remove a struct dhcp6 previously set to the netif using dhcp6_set_struct()

4.140.4.2 DHCP6_TIMER_MSECS

```
#define DHCP6_TIMER_MSECS   500
```

period (in milliseconds) of the application calling dhcp6_tmr()

4.140.5 Function Documentation

4.140.5.1 dhcp6_nd6_ra_trigger()

```
void dhcp6_nd6_ra_trigger (struct netif * netif, u8_t managed_addr_config, u8_t other_conf
```

This function is called from nd6 module when an RA message is received It triggers DHCPv6 requests (if enabled).

4.140.5.2 dhcp6_tmr()

```
void dhcp6_tmr (void )
```

DHCPv6 timeout handling (this function must be called every 500ms, see DHCP6_TIMER_MSECS).

A DHCPv6 server is expected to respond within a short period of time. This timer checks whether an outstanding DHCPv6 request is timed out.

4.141 src/include/lwip/prot/dhcp6.h File Reference

```
#include "lwip/opt.h"#include "arch/bpstruct.h"#include "arch/epstruct.h"
```

4.141.1 Data Structures

- struct dhcp6_msg

4.141.2 Macros

- #define DHCP6_STATUS_SUCCESS 0 /* Success. */
- #define DHCP6_DUID_LLT 1 /* LLT: Link-layer Address Plus Time */

4.141.3 Detailed Description

DHCPv6 protocol definitions

4.141.4 Macro Definition Documentation

4.141.4.1 DHCP6_DUID_LLT

```
#define DHCP6_DUID_LLT   1 /* LLT: Link-layer Address Plus Time */
```

DHCPv6 DUID types

4.141.4.2 DHCP6_STATUS_SUCCESS

```
#define DHCP6_STATUS_SUCCESS   0 /* Success.  */
```

DHCPv6 status codes

4.142 src/include/lwip/dns.h File Reference

```
#include "lwip/opt.h"#include "lwip/ip_addr.h"#include "lwip/err.h"
```

4.142.1 Macros

- #define DNS_TMR_INTERVAL 1000

4.142.2 Typedefs

- typedef void(* dns_found_callback) (const char *name, const ip_addr_t *ipaddr, void *callback_arg)

4.142.3 Functions

- void dns_init (void)
- void dns_tmr (void)
- void dns_setserver (u8_t numdns, const ip_addr_t *dnsserver)
- const ip_addr_t * dns_getserver (u8_t numdns)
- err_t dns_gethostbyname (const char *hostname, ip_addr_t *addr, dns_found_callback found, void *callback_arg)
- err_t dns_gethostbyname_addrtype (const char *hostname, ip_addr_t *addr, dns_found_callback found, void *callback_arg, u8_t dns_addrtype)

4.142.4 Detailed Description

DNS API

4.142.5 Macro Definition Documentation

4.142.5.1 DNS_TMR_INTERVAL

```
#define DNS_TMR_INTERVAL   1000
```

lwip DNS resolver header file.

Author: Jim Pettinato April 2007

ported from uIP resolv.c Copyright (c) 2002-2003, Adam Dunkels.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. DNS timer period

4.142.6 Typedef Documentation

4.142.6.1 dns_found_callback

```
typedef void(* dns_found_callback) (const char *name, const ip_addr_t *ipaddr, void *callb
```

Callback which is invoked when a hostname is found. A function of this type must be implemented by the application using the DNS resolver.

Parameters

name	pointer to the name that was looked up.
ipaddr	pointer to an ip_addr_t containing the IP address of the hostname, or NULL if the name could not be found (or on any other error).
callback_arg	a user-specified callback argument passed to dns_gethostbyname

4.142.7 Function Documentation

4.142.7.1 dns_init()

```
void dns_init (void )
```

Initialize the resolver: set up the UDP pcb and configure the default server (if DNS_SERVER_ADDRESS is set).

4.142.7.2 dns_tmr()

```
void dns_tmr (void )
```

The DNS resolver client timer - handle retries and timeouts and should be called every DNS_TMR_INTERVAL milliseconds (every second by default).

4.143 src/include/lwip/prot/dns.h File Reference

```
#include "lwip/arch.h"#include "arch/bpstruct.h"#include "arch/epstruct.h"
```

4.143.1 Data Structures

- struct dns_hdr

4.143.2 Macros

- #define DNS_SERVER_PORT 53
- #define DNS_MQUERY_PORT 5353

4.143.3 Detailed Description

DNS - host name to IP address resolver.

4.143.4 Macro Definition Documentation

4.143.4.1 DNS_MQUERY_PORT

```
#define DNS_MQUERY_PORT   5353
```

UDP port for multicast DNS queries

4.143.4.2 DNS_SERVER_PORT

```
#define DNS_SERVER_PORT   53
```

DNS server port address

4.144 src/include/lwip/err.h File Reference

```
#include "lwip/opt.h"#include "lwip/arch.h"
```

4.144.1 Typedefs

- typedef s8_t err_t

4.144.2 Enumerations

- enum err_enum_t { ERR_OK = 0 , ERR_MEM = -1 , ERR_BUF = -2 , ERR_TIMEOUT = -3 , ERR_RTE = -4 , ERR_INPROGRESS = -5 , ERR_VAL = -6 , ERR_WOULDBLOCK = -7 , ERR_USE = -8 , ERR_ALREADY = -9 , ERR_ISCONN = -10 , ERR_CONN = -11 , ERR_IF = -12 , ERR_ABRT = -13 , ERR_RST = -14 , ERR_CLSD = -15 , ERR_ARG = -16 }

4.144.3 Detailed Description

lwIP Error codes

4.145 src/include/lwip/etharp.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/ip4_addr.h"#include "lwip/netif.h ↩
    "#include "lwip/ip4.h"#include "lwip/prot/ethernet.h"#include "lwip/prot/etharp.h"
```

4.145.1 Data Structures

- struct etharp_q_entry

4.145.2 Macros

- #define ARP_TMR_INTERVAL   1000
- #define etharp_gratuitous(netif)   etharp_request((netif), netif_ip4_addr(netif))

4.145.3 Functions

- void etharp_tmr (void)
- ssize_t etharp_find_addr (struct netif *netif, const ip4_addr_t *ipaddr, struct eth_addr **eth_ret, const ip4_addr_t **ip_ret)
- int etharp_get_entry (size_t i, ip4_addr_t **ipaddr, struct netif **netif, struct eth_addr **eth_ret)
- err_t etharp_output (struct netif *netif, struct pbuf *q, const ip4_addr_t *ipaddr)
- err_t etharp_query (struct netif *netif, const ip4_addr_t *ipaddr, struct pbuf *q)
- err_t etharp_request (struct netif *netif, const ip4_addr_t *ipaddr)
- void etharp_cleanup_netif (struct netif *netif)
- err_t etharp_acd_probe (struct netif *netif, const ip4_addr_t *ipaddr)
- err_t etharp_acd_announce (struct netif *netif, const ip4_addr_t *ipaddr)
- void etharp_input (struct pbuf *p, struct netif *netif)

4.145.4 Detailed Description

Ethernet output function - handles OUTGOING ethernet level traffic, implements ARP resolving. To be used in most low-level netif implementations

4.145.5 Macro Definition Documentation

4.145.5.1 ARP_TMR_INTERVAL

```
#define ARP_TMR_INTERVAL   1000
```

1 seconds period

4.145.5.2 etharp_gratuitous

```
#define etharp_gratuitous( netif)   etharp_request((netif), netif_ip4_addr(netif))
```

For Ethernet network interfaces, we might want to send "gratuitous ARP"; this is an ARP packet sent by a node in order to spontaneously cause other nodes to update an entry in their ARP cache. From RFC 3220 "IP Mobility Support for IPv4" section 4.6.

4.145.6 Function Documentation

4.145.6.1 etharp_acd_announce()

```
err_t etharp_acd_announce (struct netif * netif, const ip4_addr_t * ipaddr)
```

Send an ARP request packet announcing an ipaddr. Used to send announce messages for address conflict detection.

Parameters

netif	the lwip network interface on which to send the request
ipaddr	the IP address to announce

Returns ERR_OK if the request has been sent ERR_MEM if the ARP packet couldn't be allocated any other err_t on failure

4.145.6.2 etharp_acd_probe()

```
err_t etharp_acd_probe (struct netif * netif, const ip4_addr_t * ipaddr)
```

Send an ARP request packet probing for an ipaddr. Used to send probe messages for address conflict detection.

Parameters

netif	the lwip network interface on which to send the request
ipaddr	the IP address to probe

Returns ERR_OK if the request has been sent ERR_MEM if the ARP packet couldn't be allocated any other err_t on failure

4.145.6.3 etharp_cleanup_netif()

```
void etharp_cleanup_netif (struct netif * netif)
```

Remove all ARP table entries of the specified netif.

Parameters

netif	points to a network interface

4.145.6.4 etharp_find_addr()

```
ssize_t etharp_find_addr (struct netif * netif, const ip4_addr_t * ipaddr, struct eth_addr
** eth_ret, const ip4_addr_t ** ip_ret)
```

Finds (stable) ethernet/IP address pair from ARP table using interface and IP address index.

Note
the addresses in the ARP table are in network order!

Parameters

netif	points to interface index
ipaddr	points to the (network order) IP address index
eth_ret	points to return pointer
ip_ret	points to return pointer

Returns table index if found, -1 otherwise

4.145.6.5 etharp_get_entry()

```
int etharp_get_entry (size_t i, ip4_addr_t ** ipaddr, struct netif ** netif, struct eth_ad
** eth_ret)
```

Possibility to iterate over stable ARP table entries

Parameters

i	entry number, 0 to ARP_TABLE_SIZE
ipaddr	return value: IP address
netif	return value: points to interface
eth_ret	return value: ETH address

Returns 1 on valid index, 0 otherwise

4.145.6.6 etharp_input()

```
void etharp_input (struct pbuf * p, struct netif * netif)
```

Responds to ARP requests to us. Upon ARP replies to us, add entry to cache send out queued IP packets. Updates cache with snooped address pairs.

Should be called for incoming ARP packets. The pbuf in the argument is freed by this function.

Parameters

See also pbuf_free()

p	The ARP packet that arrived on netif. Is freed by this function.
netif	The lwIP network interface on which the ARP packet pbuf arrived.

4.145.6.7 etharp_output()

```
err_t etharp_output (struct netif * netif, struct pbuf * q, const ip4_addr_t * ipaddr)
```

Resolve and fill-in Ethernet address header for outgoing IP packet.

For IP multicast and broadcast, corresponding Ethernet addresses are selected and the packet is transmitted on the link.

For unicast addresses, the packet is submitted to etharp_query(). In case the IP address is outside the local network, the IP address of the gateway is used.

Parameters

netif	The lwIP network interface which the IP packet will be sent on.
q	The pbuf(s) containing the IP packet to be sent.
ipaddr	The IP address of the packet destination.

Returns

- ERR_RTE No route to destination (no gateway to external networks), or the return type of either etharp_query() or ethernet_output().

4.145.6.8 etharp_query()

```
err_t etharp_query (struct netif * netif, const ip4_addr_t * ipaddr, struct pbuf * q)
```

Send an ARP request for the given IP address and/or queue a packet.

If the IP address was not yet in the cache, a pending ARP cache entry is added and an ARP request is sent for the given address. The packet is queued on this entry.

If the IP address was already pending in the cache, a new ARP request is sent for the given address. The packet is queued on this entry.

If the IP address was already stable in the cache, and a packet is given, it is directly sent and no ARP request is sent out.

If the IP address was already stable in the cache, and no packet is given, an ARP request is sent out.

Parameters

netif	The lwIP network interface on which ipaddr must be queried for.
ipaddr	The IP address to be resolved.
q	If non-NULL, a pbuf that must be delivered to the IP address. q is not freed by this function.

Note
q must only be ONE packet, not a packet queue!

Returns

- ERR_BUF Could not make room for Ethernet header.

- ERR_MEM Hardware address unknown, and no more ARP entries available to query for address or queue the packet.
- ERR_MEM Could not queue packet due to memory shortage.
- ERR_RTE No route to destination (no gateway to external networks).
- ERR_ARG Non-unicast address given, those will not appear in ARP cache.

4.145.6.9 etharp_request()

```
err_t etharp_request (struct netif * netif, const ip4_addr_t * ipaddr)
```

Send an ARP request packet asking for ipaddr.

Parameters

netif	the lwip network interface on which to send the request
ipaddr	the IP address for which to ask

Returns ERR_OK if the request has been sent ERR_MEM if the ARP packet couldn't be allocated any other err_t on failure

4.145.6.10 etharp_tmr()

```
void etharp_tmr (void )
```

Clears expired entries in the ARP table.

This function should be called every ARP_TMR_INTERVAL milliseconds (1 second), in order to expire entries in the ARP table.

4.146 src/include/lwip/prot/etharp.h File Reference

```
#include "lwip/arch.h"#include "lwip/prot/ethernet.h"#include "arch/bpstruct.h"#include " ↩
    arch/epstruct.h"
```

4.146.1 Data Structures

- struct ip4_addr_wordaligned
- struct etharp_hdr

4.146.2 Macros

- #define IPADDR_WORDALIGNED_COPY_TO_IP4_ADDR_T(dest, src)   SMEMCPY(dest, src, sizeof(ip4_addr_t))
- #define IPADDR_WORDALIGNED_COPY_FROM_IP4_ADDR_T(dest, src)   SMEMCPY(dest, src, sizeof(ip4_addr_t))

4.146.3 Detailed Description

ARP protocol definitions

4.146.4 Macro Definition Documentation

4.146.4.1 IPADDR_WORDALIGNED_COPY_FROM_IP4_ADDR_T

```
#define IPADDR_WORDALIGNED_COPY_FROM_IP4_ADDR_T( dest, src)   SMEMCPY(dest, src, sizeof(ip
```

MEMCPY-like copying of IP addresses where addresses are known to be 16-bit-aligned if the port is correctly configured (so a port could define this to copying 2 u16_t's) - no NULL-pointer-checking needed.

4.146.4.2 IPADDR_WORDALIGNED_COPY_TO_IP4_ADDR_T

```
#define IPADDR_WORDALIGNED_COPY_TO_IP4_ADDR_T( dest, src)   SMEMCPY(dest, src, sizeof(ip4_
```

MEMCPY-like copying of IP addresses where addresses are known to be 16-bit-aligned if the port is correctly configured (so a port could define this to copying 2 u16_t's) - no NULL-pointer-checking needed.

4.147 src/include/lwip/ethip6.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/ip6.h"#include "lwip/ip6_addr.h"# ↩
    include "lwip/netif.h"
```

4.147.1 Functions

- err_t ethip6_output (struct netif *netif, struct pbuf *q, const ip6_addr_t *ip6addr)

4.147.2 Detailed Description

Ethernet output for IPv6. Uses ND tables for link-layer addressing.

4.147.3 Function Documentation

4.147.3.1 ethip6_output()

```
err_t ethip6_output (struct netif * netif, struct pbuf * q, const ip6_addr_t * ip6addr)
```

Resolve and fill-in Ethernet address header for outgoing IPv6 packet.

For IPv6 multicast, corresponding Ethernet addresses are selected and the packet is transmitted on the link.

For unicast addresses, ask the ND6 module what to do. It will either let us send the the packet right away, or queue the packet for later itself, unless an error occurs.

Parameters

netif	The lwIP network interface which the IP packet will be sent on.
q	The pbuf(s) containing the IP packet to be sent.
ip6addr	The IP address of the packet destination.

Returns

- ERR_OK or the return value of nd6_get_next_hop_addr_or_queue.

4.148 src/include/lwip/icmp.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/ip_addr.h"#include "lwip/netif.h ↩
    "#include "lwip/prot/icmp.h"#include "lwip/icmp6.h"
```

4.148.1 Enumerations

- enum icmp_dur_type { ICMP_DUR_NET = 0 , ICMP_DUR_HOST = 1 , ICMP_DUR_PROTO = 2 , ICMP_DUR_PORT = 3 , ICMP_DUR_FRAG = 4 , ICMP_DUR_SR = 5 }
- enum icmp_te_type { ICMP_TE_TTL = 0 , ICMP_TE_FRAG = 1 }

4.148.2 Functions

- void icmp_input (struct pbuf *p, struct netif *inp)
- void icmp_dest_unreach (struct pbuf *p, enum icmp_dur_type t)
- void icmp_time_exceeded (struct pbuf *p, enum icmp_te_type t)

4.148.3 Detailed Description

ICMP API

4.148.4 Enumeration Type Documentation

4.148.4.1 icmp_dur_type

`enum icmp_dur_type`

ICMP destination unreachable codes

ICMP_DUR_NET	net unreachable
ICMP_DUR_HOST	host unreachable
ICMP_DUR_PROTO	protocol unreachable
ICMP_DUR_PORT	port unreachable
ICMP_DUR_FRAG	fragmentation needed and DF set
ICMP_DUR_SR	source route failed

4.148.4.2 icmp_te_type

`enum icmp_te_type`

ICMP time exceeded codes

ICMP_TE_TTL	time to live exceeded in transit
ICMP_TE_FRAG	fragment reassembly time exceeded

4.148.5 Function Documentation

4.148.5.1 icmp_dest_unreach()

```
void icmp_dest_unreach (struct pbuf * p, enum icmp_dur_type t)
```

Send an icmp ’destination unreachable’ packet, called from ip_input() if the transport layer protocol is unknown and from udp_input() if the local port is not bound.

Parameters

p	the input packet for which the ’unreachable’ should be sent, p->payload pointing to the IP header
t	type of the ’unreachable’ packet

4.148.5.2 icmp_input()

```
void icmp_input (struct pbuf * p, struct netif * inp)
```

Processes ICMP input packets, called from ip_input().

Currently only processes icmp echo requests and sends out the echo response.

Parameters

p	the icmp echo request packet, p->payload pointing to the icmp header
inp	the netif on which this packet was received

4.148.5.3 icmp_time_exceeded()

```
void icmp_time_exceeded (struct pbuf * p, enum icmp_te_type t)
```

Send a ’time exceeded’ packet, called from ip_forward() if TTL is 0.

Parameters

p	the input packet for which the ’time exceeded’ should be sent, p->payload pointing to the IP header
t	type of the ’time exceeded’ packet

4.149 src/include/lwip/prot/icmp.h File Reference

```
#include "lwip/arch.h"#include "arch/bpstruct.h"#include "arch/epstruct.h"
```

4.149.1 Data Structures

- struct icmp_hdr
- struct icmp_echo_hdr

4.149.2 Detailed Description

ICMP protocol definitions

4.150 src/include/lwip/icmp6.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/ip6_addr.h"#include "lwip/netif.h ↩
    "#include "lwip/prot/icmp6.h"
```

4.150.1 Functions

- void icmp6_input (struct pbuf *p, struct netif *inp)
- void icmp6_dest_unreach (struct pbuf *p, enum icmp6_dur_code c)
- void icmp6_packet_too_big (struct pbuf *p, u32_t mtu)
- void icmp6_time_exceeded (struct pbuf *p, enum icmp6_te_code c)
- void icmp6_time_exceeded_with_addrs (struct pbuf *p, enum icmp6_te_code c, const ip6_addr_t *src_addr, const ip6_addr_t *dest_addr)
- void icmp6_param_problem (struct pbuf *p, enum icmp6_pp_code c, const void *pointer)

4.150.2 Detailed Description

IPv6 version of ICMP, as per RFC 4443.

4.150.3 Function Documentation

4.150.3.1 icmp6_dest_unreach()

`void icmp6_dest_unreach (struct pbuf * p, enum icmp6_dur_code c)`

Send an icmpv6 'destination unreachable' packet.

This function must be used only in direct response to a packet that is being received right now. Otherwise, address zones would be lost.

Parameters

p	the input packet for which the 'unreachable' should be sent, p->payload pointing to the IPv6 header
c	ICMPv6 code for the unreachable type

4.150.3.2 icmp6_input()

`void icmp6_input (struct pbuf * p, struct netif * inp)`

Process an input ICMPv6 message. Called by ip6_input.

Will generate a reply for echo requests. Other messages are forwarded to nd6_input, or mld6_input.

Parameters

p	the mld packet, p->payload pointing to the icmpv6 header
inp	the netif on which this packet was received

4.150.3.3 icmp6_packet_too_big()

`void icmp6_packet_too_big (struct pbuf * p, u32_t mtu)`

Send an icmpv6 'packet too big' packet.

This function must be used only in direct response to a packet that is being received right now. Otherwise, address zones would be lost.

Parameters

p	the input packet for which the 'packet too big' should be sent, p->payload pointing to the IPv6 header
mtu	the maximum mtu that we can accept

4.150.3.4 icmp6_param_problem()

`void icmp6_param_problem (struct pbuf * p, enum icmp6_pp_code c, const void * pointer)`

Send an icmpv6 'parameter problem' packet.

This function must be used only in direct response to a packet that is being received right now. Otherwise, address zones would be lost and the calculated offset would be wrong (calculated against ip6_current_header()).

Parameters

p	the input packet for which the 'param problem' should be sent, p->payload pointing to the IP header
c	ICMPv6 code for the param problem type
pointer	the pointer to the byte where the parameter is found

4.150.3.5 icmp6_time_exceeded()

`void icmp6_time_exceeded (struct pbuf * p, enum icmp6_te_code c)`

Send an icmpv6 'time exceeded' packet.

This function must be used only in direct response to a packet that is being received right now. Otherwise, address zones would be lost.

Parameters

4.150.3.6 icmp6_time_exceeded_with_addrs()

```
void icmp6_time_exceeded_with_addrs (struct pbuf * p, enum icmp6_te_code c, const ip6_addr
* src_addr, const ip6_addr_t * dest_addr)
```

Send an icmpv6 'time exceeded' packet, with explicit source and destination addresses.

This function may be used to send a response sometime after receiving the packet for which this response is meant. The provided source and destination addresses are used primarily to retain their zone information.

Parameters

p	the input packet for which the ’time exceeded’ should be sent, p->payload pointing to the IPv6 header
c	ICMPv6 code for the time exceeded type

p	the input packet for which the ’time exceeded’ should be sent, p->payload pointing to the IPv6 header
c	ICMPv6 code for the time exceeded type
src_addr	source address of the original packet, with zone information
dest_addr	destination address of the original packet, with zone information

4.151 src/include/lwip/prot/icmp6.h File Reference

```
#include "lwip/arch.h"#include "arch/bpstruct.h"#include "arch/epstruct.h"
```

4.151.1 Data Structures

- struct icmp6_hdr
- struct icmp6_echo_hdr

4.151.2 Enumerations

- enum icmp6_type { ICMP6_TYPE_DUR = 1 , ICMP6_TYPE_PTB = 2 , ICMP6_TYPE_TE = 3 , ICMP6_TYPE_PP = 4 , ICMP6_TYPE_PE1 = 100 , ICMP6_TYPE_PE2 = 101 , ICMP6_TYPE_RSV_ERR = 127 , ICMP6_TYPE_EREQ = 128 , ICMP6_TYPE_EREP = 129 , ICMP6_TYPE_MLQ = 130 , ICMP6_TYPE_MLR = 131 , ICMP6_TYPE_MLD = 132 , ICMP6_TYPE_RS = 133 , ICMP6_TYPE_RA = 134 , ICMP6_TYPE_NS = 135 , ICMP6_TYPE_NA = 136 , ICMP6_TYPE_RD = 137 , ICMP6_TYPE_MRA = 151 , ICMP6_TYPE_MRS = 152 , ICMP6_TYPE_MRT = 153 , ICMP6_TYPE_PE3 = 200 , ICMP6_TYPE_PE4 = 201 , ICMP6_TYPE_RSV_INF = 255 }
- enum icmp6_dur_code { ICMP6_DUR_NO_ROUTE = 0 , ICMP6_DUR_PROHIBITED = 1 , ICMP6_DUR_SCOPE = 2 , ICMP6_DUR_ADDRESS = 3 , ICMP6_DUR_PORT = 4 , ICMP6_DUR_POLICY = 5 , ICMP6_DUR_REJECT_ROUTE = 6 }
- enum icmp6_te_code { ICMP6_TE_HL = 0 , ICMP6_TE_FRAG = 1 }
- enum icmp6_pp_code { ICMP6_PP_FIELD = 0 , ICMP6_PP_HEADER = 1 , ICMP6_PP_OPTION = 2 }

4.151.3 Detailed Description

ICMP6 protocol definitions

4.151.4 Enumeration Type Documentation

4.151.4.1 icmp6_dur_code

```
enum icmp6_dur_code
```

ICMP destination unreachable codes

ICMP6_DUR_NO_ROUTE	No route to destination
ICMP6_DUR_PROHIBITED	Communication with destination administratively prohibited
ICMP6_DUR_SCOPE	Beyond scope of source address
ICMP6_DUR_ADDRESS	Address unreachable
ICMP6_DUR_PORT	Port unreachable
ICMP6_DUR_POLICY	Source address failed ingress/egress policy
ICMP6_DUR_REJECT_ROUTE	Reject route to destination

4.151.4.2 icmp6_pp_code

```
enum icmp6_pp_code
```

ICMP parameter code

ICMP6_PP_FIELD	Erroneous header field encountered
ICMP6_PP_HEADER	Unrecognized next header type encountered
ICMP6_PP_OPTION	Unrecognized IPv6 option encountered

4.151.4.3 icmp6_te_code

```
enum icmp6_te_code
```

ICMP time exceeded codes

ICMP6_TE_HL	Hop limit exceeded in transit
ICMP6_TE_FRAG	Fragment reassembly time exceeded

4.151.4.4 icmp6_type

```
enum icmp6_type
```

ICMP type

ICMP6_TYPE_DUR	Destination unreachable
ICMP6_TYPE_PTB	Packet too big
ICMP6_TYPE_TE	Time exceeded
ICMP6_TYPE_PP	Parameter problem
ICMP6_TYPE_PE1	Private experimentation
ICMP6_TYPE_PE2	Private experimentation
ICMP6_TYPE_RSV_ERR	Reserved for expansion of error messages
ICMP6_TYPE_EREQ	Echo request
ICMP6_TYPE_EREP	Echo reply
ICMP6_TYPE_MLQ	Multicast listener query
ICMP6_TYPE_MLR	Multicast listener report
ICMP6_TYPE_MLD	Multicast listener done
ICMP6_TYPE_RS	Router solicitation
ICMP6_TYPE_RA	Router advertisement
ICMP6_TYPE_NS	Neighbor solicitation
ICMP6_TYPE_NA	Neighbor advertisement
ICMP6_TYPE_RD	Redirect
ICMP6_TYPE_MRA	Multicast router advertisement

ICMP6_TYPE_MRS	Multicast router solicitation
ICMP6_TYPE_MRT	Multicast router termination
ICMP6_TYPE_PE3	Private experimentation
ICMP6_TYPE_PE4	Private experimentation
ICMP6_TYPE_RSV_INF	Reserved for expansion of informational messages

4.152 src/include/lwip/if_api.h File Reference

```
#include "lwip/opt.h"#include "lwip/netif.h"
```

4.152.1 Functions

- char * lwip_if_indextoname (unsigned int ifindex, char *ifname)
- unsigned int lwip_if_nametoindex (const char *ifname)

4.152.2 Detailed Description

Interface Identification APIs from: RFC 3493: Basic Socket Interface Extensions for IPv6 Section 4: Interface Identification

4.153 src/include/lwip/igmp.h File Reference

```
#include "lwip/opt.h"#include "lwip/ip_addr.h"#include "lwip/netif.h"#include "lwip/pbuf.h"
```

4.153.1 Data Structures

- struct igmp_group

4.153.2 Macros

- #define netif_igmp_data(netif) ((struct igmp_group *)netif_get_client_data(netif, LWIP_NETIF_CLIENT_DATA_INDEX_IGMP))

4.153.3 Functions

- void igmp_init (void)
- err_t igmp_start (struct netif *netif)
- err_t igmp_stop (struct netif *netif)
- void igmp_report_groups (struct netif *netif)
- struct igmp_group * igmp_lookfor_group (struct netif *ifp, const ip4_addr_t *addr)
- void igmp_input (struct pbuf *p, struct netif *inp, const ip4_addr_t *dest)
- err_t igmp_joingroup (const ip4_addr_t *ifaddr, const ip4_addr_t *groupaddr)
- err_t igmp_joingroup_netif (struct netif *netif, const ip4_addr_t *groupaddr)
- err_t igmp_leavegroup (const ip4_addr_t *ifaddr, const ip4_addr_t *groupaddr)
- err_t igmp_leavegroup_netif (struct netif *netif, const ip4_addr_t *groupaddr)
- void igmp_tmr (void)

4.153.4 Detailed Description

IGMP API

4.153.5 Function Documentation

4.153.5.1 igmp_init()

```
void igmp_init (void )
```

Initialize the IGMP module

4.153.5.2 igmp_input()

```
void igmp_input (struct pbuf * p, struct netif * inp, const ip4_addr_t * dest)
```

Called from ip_input() if a new IGMP packet is received.

Parameters

p	received igmp packet, p->payload pointing to the igmp header
inp	network interface on which the packet was received
dest	destination ip address of the igmp packet

4.153.5.3 igmp_lookfor_group()

```
struct igmp_group * igmp_lookfor_group (struct netif * ifp, const ip4_addr_t * addr)
```

Search for a group in the netif's igmp group list

Parameters

ifp	the network interface for which to look
addr	the group ip address to search for

Returns a struct igmp_group* if the group has been found, NULL if the group wasn't found.

4.153.5.4 igmp_report_groups()

```
void igmp_report_groups (struct netif * netif)
```

Report IGMP memberships for this interface

Parameters

netif	network interface on which report IGMP memberships

4.153.5.5 igmp_start()

```
err_t igmp_start (struct netif * netif)
```

Start IGMP processing on interface

Parameters

netif	network interface on which start IGMP processing

4.153.5.6 igmp_stop()

```
err_t igmp_stop (struct netif * netif)
```

Stop IGMP processing on interface

Parameters

netif	network interface on which stop IGMP processing

4.153.5.7 igmp_tmr()

```
void igmp_tmr (void )
```

The igmp timer function (both for NO_SYS=1 and =0) Should be called every IGMP_TMR_INTERVAL milliseconds (100 ms is default).

4.154 src/include/lwip/prot/igmp.h File Reference

```
#include "lwip/arch.h"#include "lwip/prot/ip4.h"#include "arch/bpstruct.h"#include "arch/ ↩
    epstruct.h"
```

4.154.1 Data Structures

- struct igmp_msg

4.154.2 Detailed Description

IGMP protocol definitions

4.155 src/include/lwip/inet_chksum.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/ip_addr.h"
```

4.155.1 Macros

- #define SWAP_BYTES_IN_WORD(w) (((w) & 0xff) << 8) | (((w) & 0xff00) >> 8)
- #define FOLD_U32T(u) ((u32_t)(((u) >> 16) + ((u) & 0x0000ffffUL)))

4.155.2 Functions

- u16_t inet_chksum_pbuf (struct pbuf *p)
- u16_t ip6_chksum_pseudo (struct pbuf *p, u8_t proto, u16_t proto_len, const ip6_addr_t *src, const ip6_addr_t *dest)
- u16_t ip6_chksum_pseudo_partial (struct pbuf *p, u8_t proto, u16_t proto_len, u16_t chksum_len, const ip6_addr_t *src, const ip6_addr_t *dest)

4.155.3 Detailed Description

IP checksum calculation functions

4.155.4 Macro Definition Documentation

4.155.4.1 FOLD_U32T

```
#define FOLD_U32T( u)    ((u32_t)(((u) >> 16) + ((u) & 0x0000ffffUL)))
```

Split an u32_t in two u16_ts and add them up

4.155.4.2 SWAP_BYTES_IN_WORD

```
#define SWAP_BYTES_IN_WORD( w)    (((w) & 0xff) << 8) | (((w) & 0xff00) >> 8)
```

Swap the bytes in an u16_t: much like lwip_htons() for little-endian

4.155.5 Function Documentation

4.155.5.1 inet_chksum_pbuf()

```
u16_t inet_chksum_pbuf (struct pbuf * p)
```

Calculate a checksum over a chain of pbufs (without pseudo-header, much like inet_chksum only pbufs are used).

Parameters

p	pbuf chain over that the checksum should be calculated

Returns checksum (as u16_t) to be saved directly in the protocol header

4.155.5.2 ip6_chksum_pseudo()

```
u16_t ip6_chksum_pseudo (struct pbuf * p, u8_t proto, u16_t proto_len, const ip6_addr_t
* src, const ip6_addr_t * dest)
```

Calculates the checksum with IPv6 pseudo header used by TCP and UDP for a pbuf chain. IPv6 addresses are expected to be in network byte order.

Parameters

p	chain of pbufs over that a checksum should be calculated (ip data part)
proto	ipv6 protocol/next header (used for checksum of pseudo header)
proto_len	length of the ipv6 payload (used for checksum of pseudo header)
src	source ipv6 address (used for checksum of pseudo header)
dest	destination ipv6 address (used for checksum of pseudo header)

Returns checksum (as u16_t) to be saved directly in the protocol header

4.155.5.3 ip6_chksum_pseudo_partial()

```
u16_t ip6_chksum_pseudo_partial (struct pbuf * p, u8_t proto, u16_t proto_len, u16_t chksu
const ip6_addr_t * src, const ip6_addr_t * dest)
```

Calculates the checksum with IPv6 pseudo header used by TCP and UDP for a pbuf chain. IPv6 addresses are expected to be in network byte order. Will only compute for a portion of the payload.

Parameters

p	chain of pbufs over that a checksum should be calculated (ip data part)
proto	ipv6 protocol/next header (used for checksum of pseudo header)
proto_len	length of the ipv6 payload (used for checksum of pseudo header)
chksum_len	number of payload bytes used to compute chksum
src	source ipv6 address (used for checksum of pseudo header)
dest	destination ipv6 address (used for checksum of pseudo header)

Returns checksum (as u16_t) to be saved directly in the protocol header

4.156 src/include/lwip/init.h File Reference

```
#include "lwip/opt.h"
```

4.156.1 Macros

- #define LWIP_VERSION_MAJOR 2
- #define LWIP_VERSION_MINOR 2
- #define LWIP_VERSION_REVISION 2
- #define LWIP_VERSION_RC LWIP_RC_DEVELOPMENT
- #define LWIP_RC_RELEASE 255
- #define LWIP_RC_DEVELOPMENT 0
- #define LWIP_VERSION
- #define LWIP_VERSION_STRING LWIP_VERSTR(LWIP_VERSION_MAJOR) "." LWIP_VERSTR(LWIP_VERSION_MINOR)
  "." LWIP_VERSTR(LWIP_VERSION_REVISION) LWIP_VERSION_STRING_SUFFIX

4.156.2 Functions

- void lwip_init (void)

4.156.3 Detailed Description

lwIP initialization API

4.157 src/include/lwip/ip.h File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/pbuf.h"#include "lwip/ip_addr.h"# ↩
    include "lwip/err.h"#include "lwip/netif.h"#include "lwip/ip4.h"#include "lwip/ip6.h"# ↩
    include "lwip/prot/ip.h"
```

4.157.1 Data Structures

- struct ip_globals

4.157.2 Macros

- #define LWIP_IP_CHECK_PBUF_REF_COUNT_FOR_TX(p)   LWIP_ASSERT("p->ref == 1", (p)->ref == 1)
- #define IP_PCB
- #define ip_current_netif()   (ip_data.current_netif)
- #define ip_current_input_netif()   (ip_data.current_input_netif)
- #define ip_current_header_tot_len()   (ip_data.current_ip_header_tot_len)
- #define ip_current_src_addr()   (&ip_data.current_iphdr_src)
- #define ip_current_dest_addr()   (&ip_data.current_iphdr_dest)
- #define ip4_current_header()   ip_data.current_ip4_header
- #define ip6_current_header()   ((const struct ip6_hdr*)(ip_data.current_ip6_header))
- #define ip_current_is_v6()   (ip6_current_header() != NULL)
- #define ip6_current_src_addr()   (ip_2_ip6(&ip_data.current_iphdr_src))
- #define ip6_current_dest_addr()   (ip_2_ip6(&ip_data.current_iphdr_dest))
- #define ip_current_header_proto()
- #define ip_next_header_ptr()
- #define ip4_current_src_addr()   (ip_2_ip4(&ip_data.current_iphdr_src))
- #define ip4_current_dest_addr()   (ip_2_ip4(&ip_data.current_iphdr_dest))
- #define ip_current_src_addr()   (&ip_data.current_iphdr_src)
- #define ip_current_dest_addr()   (&ip_data.current_iphdr_dest)
- #define ip_get_option(pcb, opt)   ((pcb)->so_options & (opt))
- #define ip_set_option(pcb, opt)   ((pcb)->so_options = (u8_t)((pcb)->so_options | (opt)))
- #define ip_reset_option(pcb, opt)   ((pcb)->so_options = (u8_t)((pcb)->so_options & ~(opt)))
- #define ip_output(p, src, dest, ttl, tos, proto)
- #define ip_output_if(p, src, dest, ttl, tos, proto, netif)
- #define ip_output_if_src(p, src, dest, ttl, tos, proto, netif)
- #define ip_output_if_hdrincl(p, src, dest, netif)
- #define ip_output_hinted(p, src, dest, ttl, tos, proto, netif_hint)
- #define ip_route(src, dest)
- #define ip_netif_get_local_ip(netif, dest)

4.157.3 Functions

- err_t ip_input (struct pbuf *p, struct netif *inp)

4.157.4 Variables

- struct ip_globals ip_data

4.157.5 Detailed Description

IP API

4.157.6 Macro Definition Documentation

4.157.6.1 ip4_current_dest_addr

```
#define ip4_current_dest_addr( )    (ip_2_ip4(&ip_data.current_iphdr_dest))
```

Destination IP4 address of current_header

4.157.6.2 ip4_current_header

```
#define ip4_current_header( )    ip_data.current_ip4_header
```

Get the IPv4 header of the current packet. This function must only be called from a receive callback (udp_recv, raw_recv, tcp_accept). It will return NULL otherwise.

4.157.6.3 ip4_current_src_addr

```
#define ip4_current_src_addr( )    (ip_2_ip4(&ip_data.current_iphdr_src))
```

Source IP4 address of current_header

4.157.6.4 ip6_current_dest_addr

```
#define ip6_current_dest_addr( )    (ip_2_ip6(&ip_data.current_iphdr_dest))
```

Destination IPv6 address of current_header

4.157.6.5 ip6_current_header

```
#define ip6_current_header( )    ((const struct ip6_hdr*)(ip_data.current_ip6_header))
```

Get the IPv6 header of the current packet. This function must only be called from a receive callback (udp_recv, raw_recv, tcp_accept). It will return NULL otherwise.

4.157.6.6 ip6_current_src_addr

```
#define ip6_current_src_addr( )    (ip_2_ip6(&ip_data.current_iphdr_src))
```

Source IPv6 address of current_header

4.157.6.7 ip_current_dest_addr [1/2]

```
#define ip_current_dest_addr( )    (&ip_data.current_iphdr_dest)
```

Destination IP address of current_header

Union destination address of current_header

4.157.6.8 ip_current_dest_addr [2/2]

```
#define ip_current_dest_addr( )    (&ip_data.current_iphdr_dest)
```

Destination IP address of current_header

Union destination address of current_header

4.157.6.9 ip_current_header_proto

`#define ip_current_header_proto( )` **Value:**

```
(ip_current_is_v6() ? \
IP6H_NEXTH(ip6_current_header()) :\
IPH_PROTO(ip4_current_header()))
```

Get the transport layer protocol

4.157.6.10 ip_current_header_tot_len

```
#define ip_current_header_tot_len( )    (ip_data.current_ip_header_tot_len)
```

Total header length of ip(6)_current_header() (i.e. after this, the UDP/TCP header starts)

4.157.6.11 ip_current_input_netif

```
#define ip_current_input_netif( )    (ip_data.current_input_netif)
```

Get the interface that received the current packet. This function must only be called from a receive callback (udp_recv, raw_recv, tcp_accept). It will return NULL otherwise.

4.157.6.12 ip_current_is_v6

```
#define ip_current_is_v6( )    (ip6_current_header() != NULL)
```

Returns TRUE if the current IP input packet is IPv6, FALSE if it is IPv4

4.157.6.13 ip_current_netif

```
#define ip_current_netif( )    (ip_data.current_netif)
```

Get the interface that accepted the current packet. This may or may not be the receiving netif, depending on your netif/network setup. This function must only be called from a receive callback (udp_recv, raw_recv, tcp_accept). It will return NULL otherwise.

4.157.6.14 ip_current_src_addr [1/2]

```
#define ip_current_src_addr( )    (&ip_data.current_iphdr_src)
```

Source IP address of current_header

Union source address of current_header

4.157.6.15 ip_current_src_addr [2/2]

```
#define ip_current_src_addr( )    (&ip_data.current_iphdr_src)
```

Source IP address of current_header

Union source address of current_header

4.157.6.16 ip_get_option

`#define ip_get_option( pcb, opt)   ((pcb)->so_options & (opt))`

Gets an IP pcb option (SOF_* flags)

4.157.6.17 ip_next_header_ptr

`#define ip_next_header_ptr( )` **Value:**

```
((const void*)((ip_current_is_v6() ? \
(const u8_t*)ip6_current_header() : (const u8_t*)ip4_current_header())  +  ↩
    ip_current_header_tot_len()))
```

Get the transport layer header

4.157.6.18 ip_output_hinted

`#define ip_output_hinted( p, src, dest, ttl, tos, proto, netif_hint)` **Value:**

```
        (IP_IS_V6(dest) ? \
        ip6_output_hinted(p, ip_2_ip6(src), ip_2_ip6(dest), ttl, tos, proto, netif_hint) :  ↩
            \
        ip4_output_hinted(p, ip_2_ip4(src), ip_2_ip4(dest), ttl, tos, proto, netif_hint))
```

Output IP packet with netif_hint

4.157.6.19 ip_output_if_hdrincl

`#define ip_output_if_hdrincl( p, src, dest, netif)` **Value:**

```
        (IP_IS_V6(dest) ? \
        ip6_output_if(p, ip_2_ip6(src), LWIP_IP_HDRINCL, 0, 0, 0, netif) : \
        ip4_output_if(p, ip_2_ip4(src), LWIP_IP_HDRINCL, 0, 0, 0, netif))
```

Output IP packet that already includes an IP header.

4.157.6.20 IP_PCB

`#define IP_PCB` **Value:**

```
  /* ip addresses in network byte order */ \
  ip_addr_t local_ip;                      \
  ip_addr_t remote_ip;                     \
  /* Bound netif index */                  \
  u8_t netif_idx;                          \
  /* Socket options */                     \
  u8_t so_options;                         \
  /* Type Of Service */                    \
  u8_t tos;                                \
  /* Time To Live */                       \
  u8_t ttl                                 \
  /* link layer address resolution hint */ \
  IP_PCB_NETIFHINT
```

This is the common part of all PCB types. It needs to be at the beginning of a PCB type definition. It is located here so that changes to this common part are made in one location instead of having to change all PCB structs.

4.157.6.21 ip_reset_option

```
#define ip_reset_option( pcb, opt)   ((pcb)->so_options = (u8_t)((pcb)->so_options & ~(opt
```

Resets an IP pcb option (SOF_* flags)

4.157.6.22 ip_set_option

```
#define ip_set_option( pcb, opt)   ((pcb)->so_options = (u8_t)((pcb)->so_options | (opt)))
```

Sets an IP pcb option (SOF_* flags)

4.157.6.23 LWIP_IP_CHECK_PBUF_REF_COUNT_FOR_TX

```
#define LWIP_IP_CHECK_PBUF_REF_COUNT_FOR_TX( p)   LWIP_ASSERT("p->ref == 1", (p)->ref ==
1)
```

pbufs passed to IP must have a ref-count of 1 as their payload pointer gets altered as the packet is passed down the stack

4.157.7 Variable Documentation

4.157.7.1 ip_data

```
struct ip_globals ip_data[extern]
```

Global data for both IPv4 and IPv6

4.158 src/include/lwip/prot/ip.h File Reference

```
#include "lwip/arch.h"
```

4.158.1 Macros

- #define IP_HDR_GET_VERSION(ptr)   ((*(u8_t*)(ptr)) >> 4)

4.158.2 Detailed Description

IP protocol definitions

4.158.3 Macro Definition Documentation

4.158.3.1 IP_HDR_GET_VERSION

```
#define IP_HDR_GET_VERSION( ptr)   ((*(u8_t*)(ptr)) >> 4)
```

This operates on a void* by loading the first byte

4.159 src/include/lwip/ip4.h File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/pbuf.h"#include "lwip/ip4_addr.h"# ↩
    include "lwip/err.h"#include "lwip/netif.h"#include "lwip/prot/ip4.h"
```

4.159.1 Macros

- #define IP_OPTIONS_SEND (LWIP_IPV4 && LWIP_IGMP)

4.159.2 Functions

- struct netif * ip4_route (const ip4_addr_t *dest)
- struct netif * ip4_route_src (const ip4_addr_t *src, const ip4_addr_t *dest)
- err_t ip4_input (struct pbuf *p, struct netif *inp)
- err_t ip4_output (struct pbuf *p, const ip4_addr_t *src, const ip4_addr_t *dest, u8_t ttl, u8_t tos, u8_t proto)
- err_t ip4_output_if (struct pbuf *p, const ip4_addr_t *src, const ip4_addr_t *dest, u8_t ttl, u8_t tos, u8_t proto, struct netif *netif)
- err_t ip4_output_if_src (struct pbuf *p, const ip4_addr_t *src, const ip4_addr_t *dest, u8_t ttl, u8_t tos, u8_t proto, struct netif *netif)
- err_t ip4_output_if_opt (struct pbuf *p, const ip4_addr_t *src, const ip4_addr_t *dest, u8_t ttl, u8_t tos, u8_t proto, struct netif *netif, void *ip_options, u16_t optlen)
- err_t ip4_output_if_opt_src (struct pbuf *p, const ip4_addr_t *src, const ip4_addr_t *dest, u8_t ttl, u8_t tos, u8_t proto, struct netif *netif, void *ip_options, u16_t optlen)
- void ip4_set_default_multicast_netif (struct netif *default_multicast_netif)

4.159.3 Detailed Description

IPv4 API

4.159.4 Macro Definition Documentation

4.159.4.1 IP_OPTIONS_SEND

```
#define IP_OPTIONS_SEND   (LWIP_IPV4 && LWIP_IGMP)
```

Currently, the function ip_output_if_opt() is only used with IGMP

4.159.5 Function Documentation

4.159.5.1 ip4_input()

```
err_t ip4_input (struct pbuf * p, struct netif * inp)
```

This function is called by the network interface device driver when an IP packet is received. The function does the basic checks of the IP header such as packet size being at least larger than the header size etc. If the packet was not destined for us, the packet is forwarded (using ip_forward). The IP checksum is always checked.

Finally, the packet is sent to the upper layer protocol input function.

Parameters

p	the received IP packet (p->payload points to IP header)
inp	the netif on which this packet was received

Returns ERR_OK if the packet was processed (could return ERR_* if it wasn't processed, but currently always returns ERR_OK)

4.159.5.2 ip4_output()

```
err_t ip4_output (struct pbuf * p, const ip4_addr_t * src, const ip4_addr_t * dest, u8_t
ttl, u8_t tos, u8_t proto)
```

Simple interface to ip_output_if. It finds the outgoing network interface and calls upon ip_output_if to do the actual work.

Parameters

p	the packet to send (p->payload points to the data, e.g. next protocol header; if dest == LWIP_IP_HDRINCL, p already includes an IP header and p->payload points to that IP header)
src	the source IP address to send from (if src == IP4_ADDR_ANY, the IP address of the netif used to send is used as source address)
dest	the destination IP address to send the packet to
ttl	the TTL value to be set in the IP header
tos	the TOS value to be set in the IP header
proto	the PROTOCOL to be set in the IP header

Returns ERR_RTE if no route is found see ip_output_if() for more return values

4.159.5.3 ip4_output_if()

```
err_t ip4_output_if (struct pbuf * p, const ip4_addr_t * src, const ip4_addr_t * dest,
u8_t ttl, u8_t tos, u8_t proto, struct netif * netif)
```

Sends an IP packet on a network interface. This function constructs the IP header and calculates the IP header checksum. If the source IP address is NULL, the IP address of the outgoing network interface is filled in as source address. If the destination IP address is LWIP_IP_HDRINCL, p is assumed to already include an IP header and p->payload points to it instead of the data.

Parameters

p	the packet to send (p->payload points to the data, e.g. next protocol header; if dest == LWIP_IP_HDRINCL, p already includes an IP header and p->payload points to that IP header)
src	the source IP address to send from (if src == IP4_ADDR_ANY, the IP address of the netif used to send is used as source address)
dest	the destination IP address to send the packet to
ttl	the TTL value to be set in the IP header
tos	the TOS value to be set in the IP header
proto	the PROTOCOL to be set in the IP header
netif	the netif on which to send this packet

Returns ERR_OK if the packet was sent OK ERR_BUF if p doesn't have enough space for IP/LINK headers returns errors returned by netif->output

Note
ip_id: RFC791 "some host may be able to simply use unique identifiers independent of destination"

4.159.5.4 ip4_output_if_opt()

```
err_t ip4_output_if_opt (struct pbuf * p, const ip4_addr_t * src, const ip4_addr_t * dest,
u8_t ttl, u8_t tos, u8_t proto, struct netif * netif, void * ip_options, u16_t optlen)
```

Same as ip_output_if() but with the possibility to include IP options:

@ param ip_options pointer to the IP options, copied into the IP header @ param optlen length of ip_options

4.159.5.5 ip4_output_if_opt_src()

`err_t ip4_output_if_opt_src (struct pbuf * p, const ip4_addr_t * src, const ip4_addr_t * dest, u8_t ttl, u8_t tos, u8_t proto, struct netif * netif, void * ip_options, u16_t optlen)`

Same as ip_output_if_opt() but 'src' address is not replaced by netif address when it is 'any'.

4.159.5.6 ip4_output_if_src()

`err_t ip4_output_if_src (struct pbuf * p, const ip4_addr_t * src, const ip4_addr_t * dest, u8_t ttl, u8_t tos, u8_t proto, struct netif * netif)`

Same as ip_output_if() but 'src' address is not replaced by netif address when it is 'any'.

4.159.5.7 ip4_route()

`struct netif * ip4_route (const ip4_addr_t * dest)`

Finds the appropriate network interface for a given IP address. It searches the list of network interfaces linearly. A match is found if the masked IP address of the network interface equals the masked IP address given to the function.

Parameters

dest	the destination IP address for which to find the route

Returns the netif on which to send to reach dest

4.159.5.8 ip4_route_src()

`struct netif * ip4_route_src (const ip4_addr_t * src, const ip4_addr_t * dest)`

Source based IPv4 routing must be fully implemented in LWIP_HOOK_IP4_ROUTE_SRC(). This function only provides the parameters.

4.160 src/include/lwip/prot/ip4.h File Reference

```
#include "lwip/arch.h"#include "lwip/ip4_addr.h"#include "arch/bpstruct.h"#include "arch/ ↩
    epstruct.h"
```

4.160.1 Data Structures

- struct ip4_addr_packed

4.160.2 Detailed Description

IPv4 protocol definitions

4.161 src/include/lwip/ip4_addr.h File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"
```

4.161.1 Data Structures

- struct ip4_addr

4.161.2 Macros

- #define IPADDR_NONE ((u32_t)0xffffffffUL)
- #define IPADDR_LOOPBACK ((u32_t)0x7f000001UL)
- #define IPADDR_ANY ((u32_t)0x00000000UL)
- #define IPADDR_BROADCAST ((u32_t)0xffffffffUL)
- #define IP4_ADDR(ipaddr, a, b, c, d) (ipaddr)->addr = PP_HTONL(LWIP_MAKEU32(a,b,c,d))
- #define ip4_addr_copy(dest, src) ((dest).addr = (src).addr)
- #define ip4_addr_set(dest, src)
- #define ip4_addr_set_zero(ipaddr) ((ipaddr)->addr = 0)
- #define ip4_addr_set_any(ipaddr) ((ipaddr)->addr = IPADDR_ANY)
- #define ip4_addr_set_loopback(ipaddr) ((ipaddr)->addr = PP_HTONL(IPADDR_LOOPBACK))
- #define ip4_addr_isloopback(ipaddr) (((ipaddr)->addr & PP_HTONL(IP_CLASSA_NET)) == PP_HTONL(((u32_t)IP_LOOPBACK
  << 24))
- #define ip4_addr_set_hton(dest, src)
- #define ip4_addr_set_u32(dest_ipaddr, src_u32) ((dest_ipaddr)->addr = (src_u32))
- #define ip4_addr_get_u32(src_ipaddr) ((src_ipaddr)->addr)
- #define ip4_addr_get_network(target, host, netmask) do { ((target)->addr = ((host)->addr) & ((netmask)->addr)); } while(0)
- #define ip4_addr_netcmp(addr1, addr2, mask) ip4_addr_net_eq(addr1, addr2, mask)
- #define ip4_addr_net_eq(addr1, addr2, mask)
- #define ip4_addr_cmp(addr1, addr2) ip4_addr_eq(addr1, addr2)
- #define ip_ntoa(ipaddr) ipaddr_ntoa(ipaddr)

4.161.3 Typedefs

- typedef struct ip4_addr ip4_addr_t

4.161.4 Functions

- u8_t ip4_addr_isbroadcast_u32 (u32_t addr, const struct netif *netif)
- u8_t ip4_addr_netmask_valid (u32_t netmask)
- u32_t ipaddr_addr (const char *cp)
- int ip4addr_aton (const char *cp, ip4_addr_t *addr)
- char * ip4addr_ntoa (const ip4_addr_t *addr)
- char * ip4addr_ntoa_r (const ip4_addr_t *addr, char *buf, int buflen)

4.161.5 Detailed Description

IPv4 address API

4.161.6 Macro Definition Documentation

4.161.6.1 IP4_ADDR

```
#define IP4_ADDR( ipaddr, a, b, c, d)   (ipaddr)->addr = PP_HTONL(LWIP_MAKEU32(a,b,c,d))
```

Set an IP address given by the four byte-parts

4.161.6.2 ip4_addr_cmp

```
#define ip4_addr_cmp( addr1, addr2)   ip4_addr_eq(addr1, addr2)
```

Deprecated Renamed to ip4_addr_eq

4.161.6.3 ip4_addr_copy

```
#define ip4_addr_copy( dest, src)   ((dest).addr = (src).addr)
```

Copy IP address - faster than ip4_addr_set: no NULL check

4.161.6.4 ip4_addr_get_network

```
#define ip4_addr_get_network( target, host, netmask)   do { ((target)->addr = ((host)->add
& ((netmask)->addr)); } while(0)
```

Get the network address by combining host address with netmask

4.161.6.5 ip4_addr_get_u32

```
#define ip4_addr_get_u32( src_ipaddr)   ((src_ipaddr)->addr)
```

IPv4 only: get the IP address as an u32_t

4.161.6.6 ip4_addr_isloopback

```
#define ip4_addr_isloopback( ipaddr)   (((ipaddr)->addr & PP_HTONL(IP_CLASSA_NET)) == PP_H
<< 24))
```

Check if an address is in the loopback region

4.161.6.7 ip4_addr_net_eq

`#define ip4_addr_net_eq( addr1, addr2, mask)` **Value:**

```
 (((addr1)->addr & \
 (mask)->addr) == \
((addr2)->addr & \
 (mask)->addr))
```

Determine if two address are on the same network.

- addr1 IP address 1
- addr2 IP address 2
- mask network identifier mask **Returns** !0 if the network identifiers of both address match

4.161.6.8 ip4_addr_netcmp

`#define ip4_addr_netcmp( addr1, addr2, mask)    ip4_addr_net_eq(addr1, addr2, mask)`

Determine if two address are on the same network. Deprecated Renamed to ip4_addr_net_eq

4.161.6.9 ip4_addr_set

`#define ip4_addr_set( dest, src)` **Value:**

```
((dest)->addr = \
((src) == NULL ? 0 : \
(src)->addr))
```

Safely copy one IP address to another (src may be NULL)

4.161.6.10 ip4_addr_set_any

`#define ip4_addr_set_any( ipaddr)    ((ipaddr)->addr = IPADDR_ANY)`

Set address to IPADDR_ANY (no need for lwip_htonl())

4.161.6.11 ip4_addr_set_hton

`#define ip4_addr_set_hton( dest, src)` **Value:**

```
((dest)->addr = \
((src) == NULL ? 0:\
lwip_htonl((src)->addr)))
```

Safely copy one IP address to another and change byte order from host- to network-order.

4.161.6.12 ip4_addr_set_loopback

`#define ip4_addr_set_loopback( ipaddr)    ((ipaddr)->addr = PP_HTONL(IPADDR_LOOPBACK))`

Set address to loopback address

4.161.6.13 ip4_addr_set_u32

`#define ip4_addr_set_u32( dest_ipaddr, src_u32)    ((dest_ipaddr)->addr = (src_u32))`

IPv4 only: set the IP address given as an u32_t

4.161.6.14 ip4_addr_set_zero

`#define ip4_addr_set_zero( ipaddr)    ((ipaddr)->addr = 0)`

Set complete address to zero

4.161.6.15 ip_ntoa

`#define ip_ntoa( ipaddr)    ipaddr_ntoa(ipaddr)`

For backwards compatibility

4.161.6.16 IPADDR_ANY

```
#define IPADDR_ANY   ((u32_t)0x00000000UL)
```

0.0.0.0

4.161.6.17 IPADDR_BROADCAST

```
#define IPADDR_BROADCAST   ((u32_t)0xffffffffUL)
```

255.255.255.255

4.161.6.18 IPADDR_LOOPBACK

```
#define IPADDR_LOOPBACK   ((u32_t)0x7f000001UL)
```

127.0.0.1

4.161.6.19 IPADDR_NONE

```
#define IPADDR_NONE   ((u32_t)0xffffffffUL)
```

255.255.255.255

4.161.7 Typedef Documentation

4.161.7.1 ip4_addr_t

```
typedef struct ip4_addr ip4_addr_t
```

ip4_addr_t uses a struct for convenience only, so that the same defines can operate both on ip4_addr_t as well as on ip4_addr_p_t.

4.161.8 Function Documentation

4.161.8.1 ip4_addr_isbroadcast_u32()

```
u8_t ip4_addr_isbroadcast_u32 (u32_t addr, const struct netif * netif)
```

Determine if an address is a broadcast address on a network interface

Parameters

addr	address to be checked
netif	the network interface against which the address is checked

Returns returns non-zero if the address is a broadcast address

4.161.8.2 ip4_addr_netmask_valid()

```
u8_t ip4_addr_netmask_valid (u32_t netmask)
```

Checks if a netmask is valid (starting with ones, then only zeros)

Parameters

Returns 1 if the netmask is valid, 0 if it is not

netmask	the IPv4 netmask to check (in network byte order!)

4.161.8.3 ip4addr_aton()

```
int ip4addr_aton (const char * cp, ip4_addr_t * addr)
```

Check whether "cp" is a valid ascii representation of an Internet address and convert to a binary address. Returns 1 if the address is valid, 0 if not. This replaces inet_addr, the return value from which cannot distinguish between failure and a local broadcast address.

Parameters

cp	IP address in ascii representation (e.g. "127.0.0.1")
addr	pointer to which to save the ip address in network order

Returns 1 if cp could be converted to addr, 0 on failure

4.161.8.4 ip4addr_ntoa()

```
char * ip4addr_ntoa (const ip4_addr_t * addr)
```

returns ptr to static buffer; not reentrant!

Convert numeric IP address into decimal dotted ASCII representation. returns ptr to static buffer; not reentrant!

Parameters

addr	ip address in network order to convert

Returns pointer to a global static (!) buffer that holds the ASCII representation of addr

4.161.8.5 ip4addr_ntoa_r()

```
char * ip4addr_ntoa_r (const ip4_addr_t * addr, char * buf, int buflen)
```

Same as ip4addr_ntoa, but reentrant since a user-supplied buffer is used.

Parameters

Returns either pointer to buf which now holds the ASCII representation of addr or NULL if buf was too small

4.161.8.6 ipaddr_addr()

```
u32_t ipaddr_addr (const char * cp)
```

Ascii internet address interpretation routine. The value returned is in network order.

Parameters

Returns ip address in network order

4.162 src/include/lwip/ip4_frag.h File Reference

```
#include "lwip/opt.h"#include "lwip/err.h"#include "lwip/pbuf.h"#include "lwip/netif.h"# ↩
    include "lwip/ip_addr.h"#include "lwip/ip.h"
```

addr	ip address in network order to convert
buf	target buffer where the string is stored
buflen	length of buf

cp	IP address in ascii representation (e.g. "127.0.0.1")

4.162.1 Data Structures

- struct ip_reassdata
- struct pbuf_custom_ref

4.162.2 Functions

- void ip_reass_tmr (void)
- struct pbuf * ip4_reass (struct pbuf *p)
- err_t ip4_frag (struct pbuf *p, struct netif *netif, const ip4_addr_t *dest)

4.162.3 Detailed Description

IP fragmentation/reassembly

4.162.4 Function Documentation

4.162.4.1 ip4_frag()

```
err_t ip4_frag (struct pbuf * p, struct netif * netif, const ip4_addr_t * dest)
```

Fragment an IP datagram if too large for the netif.

Chop the datagram in MTU sized chunks and send them in order by pointing PBUF_REFs into p.

Parameters

p	ip packet to send
netif	the netif on which to send
dest	destination ip address to which to send

Returns ERR_OK if sent successfully, err_t otherwise

4.162.4.2 ip4_reass()

```
struct pbuf * ip4_reass (struct pbuf * p)
```

Reassembles incoming IP fragments into an IP datagram.

Parameters

Returns NULL if reassembly is incomplete, ? otherwise

p	points to a pbuf chain of the fragment

4.162.4.3 ip_reass_tmr()

```
void ip_reass_tmr (void )
```

Reassembly timer base function for both NO_SYS == 0 and 1 (!).

Should be called every 1000 msec (defined by IP_TMR_INTERVAL).

4.163 src/include/lwip/ip6.h File Reference

```
#include "lwip/opt.h"#include "lwip/ip6_addr.h"#include "lwip/prot/ip6.h"#include "lwip/def ↩
    .h"#include "lwip/pbuf.h"#include "lwip/netif.h"#include "lwip/err.h"
```

4.163.1 Functions

- struct netif * ip6_route (const ip6_addr_t *src, const ip6_addr_t *dest)
- const ip_addr_t * ip6_select_source_address (struct netif *netif, const ip6_addr_t *dest)
- err_t ip6_input (struct pbuf *p, struct netif *inp)
- err_t ip6_output (struct pbuf *p, const ip6_addr_t *src, const ip6_addr_t *dest, u8_t hl, u8_t tc, u8_t nexth)
- err_t ip6_output_if (struct pbuf *p, const ip6_addr_t *src, const ip6_addr_t *dest, u8_t hl, u8_t tc, u8_t nexth, struct netif *netif)
- err_t ip6_output_if_src (struct pbuf *p, const ip6_addr_t *src, const ip6_addr_t *dest, u8_t hl, u8_t tc, u8_t nexth, struct netif *netif)
- err_t ip6_options_add_hbh_ra (struct pbuf *p, u8_t nexth, u8_t value)

4.163.2 Detailed Description

IPv6 layer.

4.163.3 Function Documentation

4.163.3.1 ip6_input()

```
err_t ip6_input (struct pbuf * p, struct netif * inp)
```

This function is called by the network interface device driver when an IPv6 packet is received. The function does the basic checks of the IP header such as packet size being at least larger than the header size etc. If the packet was not destined for us, the packet is forwarded (using ip6_forward).

Finally, the packet is sent to the upper layer protocol input function.

Parameters

p	the received IPv6 packet (p->payload points to IPv6 header)
inp	the netif on which this packet was received

Returns ERR_OK if the packet was processed (could return ERR_* if it wasn't processed, but currently always returns ERR_OK)

4.163.3.2 ip6_options_add_hbh_ra()

```
err_t ip6_options_add_hbh_ra (struct pbuf * p, u8_t nexth, u8_t value)
```

Add a hop-by-hop options header with a router alert option and padding.

Used by MLD when sending a Multicast listener report/done message.

Parameters

p	the packet to which we will prepend the options header
nexth	the next header protocol number (e.g. IP6_NEXTH_ICMP6)
value	the value of the router alert option data (e.g. IP6_ROUTER_ALERT_VALUE_MLD)

Returns ERR_OK if hop-by-hop header was added, ERR_* otherwise

4.163.3.3 ip6_output()

```
err_t ip6_output (struct pbuf * p, const ip6_addr_t * src, const ip6_addr_t * dest, u8_t
hl, u8_t tc, u8_t nexth)
```

Simple interface to ip6_output_if. It finds the outgoing network interface and calls upon ip6_output_if to do the actual work.

Parameters

p	the packet to send (p->payload points to the data, e.g. next protocol header; if dest == LWIP_IP_HDRINCL, p already includes an IPv6 header and p->payload points to that IPv6 header)
src	the source IPv6 address to send from (if src == IP6_ADDR_ANY, an IP address of the netif is selected and used as source address. if src == NULL, IP6_ADDR_ANY is used as source)
dest	the destination IPv6 address to send the packet to
hl	the Hop Limit value to be set in the IPv6 header
tc	the Traffic Class value to be set in the IPv6 header
nexth	the Next Header to be set in the IPv6 header

Returns ERR_RTE if no route is found see ip_output_if() for more return values

4.163.3.4 ip6_output_if()

```
err_t ip6_output_if (struct pbuf * p, const ip6_addr_t * src, const ip6_addr_t * dest,
u8_t hl, u8_t tc, u8_t nexth, struct netif * netif)
```

Sends an IPv6 packet on a network interface. This function constructs the IPv6 header. If the source IPv6 address is NULL, the IPv6 "ANY" address is used as source (usually during network startup). If the source IPv6 address it IP6_ADDR_ANY, the most appropriate IPv6 address of the outgoing network interface is filled in as source address. If the destination IPv6 address is LWIP_IP_HDRINCL, p is assumed to already include an IPv6 header and p->payload points to it instead of the data.

Parameters

Returns ERR_OK if the packet was sent OK ERR_BUF if p doesn't have enough space for IPv6/LINK headers returns errors returned by netif->output_ip6

4.163.3.5 ip6_output_if_src()

```
err_t ip6_output_if_src (struct pbuf * p, const ip6_addr_t * src, const ip6_addr_t * dest,
u8_t hl, u8_t tc, u8_t nexth, struct netif * netif)
```

Same as ip6_output_if() but 'src' address is not replaced by netif address when it is 'any'.

p	the packet to send (p->payload points to the data, e.g. next protocol header; if dest == LWIP_IP_HDRINCL, p already includes an IPv6 header and p->payload points to that IPv6 header)
src	the source IPv6 address to send from (if src == IP6_ADDR_ANY, an IP address of the netif is selected and used as source address. if src == NULL, IP6_ADDR_ANY is used as source) (src is possibly not properly zoned)
dest	the destination IPv6 address to send the packet to (possibly not properly zoned)
hl	the Hop Limit value to be set in the IPv6 header
tc	the Traffic Class value to be set in the IPv6 header
nexth	the Next Header to be set in the IPv6 header
netif	the netif on which to send this packet

4.163.3.6 ip6_route()

```
struct netif * ip6_route (const ip6_addr_t * src, const ip6_addr_t * dest)
```

Finds the appropriate network interface for a given IPv6 address. It tries to select a netif following a sequence of heuristics: 1) if there is only 1 netif, return it 2) if the destination is a zoned address, match its zone to a netif 3) if the either the source or destination address is a scoped address, match the source address's zone (if set) or address (if not) to a netif 4) tries to match the destination subnet to a configured address 5) tries to find a router-announced route 6) tries to match the (unscoped) source address to the netif 7) returns the default netif, if configured

Note that each of the two given addresses may or may not be properly zoned.

Parameters

src	the source IPv6 address, if known
dest	the destination IPv6 address for which to find the route

Returns the netif on which to send to reach dest

4.164 src/include/lwip/prot/ip6.h File Reference

```
#include "lwip/arch.h"#include "lwip/ip6_addr.h"#include "arch/bpstruct.h"#include "arch/ ↩
    epstruct.h"
```

4.164.1 Data Structures

- struct ip6_addr_packed
- struct ip6_hdr

4.164.2 Detailed Description

IPv6 protocol definitions

4.165 src/include/lwip/ip6_addr.h File Reference

```
#include "lwip/opt.h"#include "def.h"#include "lwip/ip6_zone.h"
```

4.165.1 Data Structures

- struct ip6_addr

4.165.2 Macros

- #define IP6_ADDR_PART(ip6addr, index, a, b, c, d)   (ip6addr)->addr[index] = PP_HTONL(LWIP_MAKEU32(a,b,c,d))
- #define IP6_ADDR(ip6addr, idx0, idx1, idx2, idx3)
- #define IP6_ADDR_BLOCK1(ip6addr)   ((u16_t)((lwip_htonl((ip6addr)->addr[0]) >> 16) & 0xffff))
- #define IP6_ADDR_BLOCK2(ip6addr)   ((u16_t)((lwip_htonl((ip6addr)->addr[0])) & 0xffff))
- #define IP6_ADDR_BLOCK3(ip6addr)   ((u16_t)((lwip_htonl((ip6addr)->addr[1]) >> 16) & 0xffff))
- #define IP6_ADDR_BLOCK4(ip6addr)   ((u16_t)((lwip_htonl((ip6addr)->addr[1])) & 0xffff))
- #define IP6_ADDR_BLOCK5(ip6addr)   ((u16_t)((lwip_htonl((ip6addr)->addr[2]) >> 16) & 0xffff))
- #define IP6_ADDR_BLOCK6(ip6addr)   ((u16_t)((lwip_htonl((ip6addr)->addr[2])) & 0xffff))
- #define IP6_ADDR_BLOCK7(ip6addr)   ((u16_t)((lwip_htonl((ip6addr)->addr[3]) >> 16) & 0xffff))
- #define IP6_ADDR_BLOCK8(ip6addr)   ((u16_t)((lwip_htonl((ip6addr)->addr[3])) & 0xffff))
- #define ip6_addr_copy(dest, src)
- #define ip6_addr_set(dest, src)
- #define ip6_addr_copy_from_packed(dest, src)
- #define ip6_addr_copy_to_packed(dest, src)
- #define ip6_addr_set_zero(ip6addr)
- #define ip6_addr_set_any(ip6addr)   ip6_addr_set_zero(ip6addr)
- #define ip6_addr_set_loopback(ip6addr)
- #define ip6_addr_set_hton(dest, src)
- #define ip6_addr_netcmp_zoneless(addr1, addr2)   ip6_addr_net_zoneless_eq(addr1, addr2)
- #define ip6_addr_net_zoneless_eq(addr1, addr2)
- #define ip6_addr_netcmp(addr1, addr2)   ip6_addr_net_eq(addr1, addr2)
- #define ip6_addr_net_eq(addr1, addr2)
- #define ip6_addr_cmp_zoneless(addr1, addr2)   ip6_addr_zoneless_eq(addr1, addr2)
- #define ip6_addr_zoneless_eq(addr1, addr2)
- #define ip6_addr_cmp(addr1, addr2)   ip6_addr_eq(addr1, addr2)
- #define ip6_addr_eq(addr1, addr2)
- #define ip6_addr_cmp_packed(ip6addr, paddr, zone_idx)   ip6_addr_packed_eq(ip6addr, paddr, zone_idx)
- #define ip6_addr_packed_eq(ip6addr, paddr, zone_idx)

4.165.3 Typedefs

- typedef struct ip6_addr ip6_addr_t

4.165.4 Functions

- int ip6addr_aton (const char *cp, ip6_addr_t *addr)
- char * ip6addr_ntoa (const ip6_addr_t *addr)
- char * ip6addr_ntoa_r (const ip6_addr_t *addr, char *buf, int buflen)

4.165.5 Detailed Description

IPv6 addresses.

4.165.6 Macro Definition Documentation

4.165.6.1 IP6_ADDR

`#define IP6_ADDR( ip6addr, idx0, idx1, idx2, idx3)` **Value:**

```
do { \
(ip6addr)->addr[0] = idx0; \
(ip6addr)->addr[1] = idx1; \
(ip6addr)->addr[2] = idx2; \
(ip6addr)->addr[3] = idx3; \
ip6_addr_clear_zone(ip6addr); } while(0)
```

Set a full IPv6 address by passing the 4 u32_t indices in network byte order (use PP_HTONL() for constants)

4.165.6.2 IP6_ADDR_BLOCK1

`#define IP6_ADDR_BLOCK1( ip6addr)    ((u16_t)((lwip_htonl((ip6addr)->addr[0]) >> 16) & 0xff`

Access address in 16-bit block

4.165.6.3 IP6_ADDR_BLOCK2

`#define IP6_ADDR_BLOCK2( ip6addr)    ((u16_t)((lwip_htonl((ip6addr)->addr[0])) & 0xffff))`

Access address in 16-bit block

4.165.6.4 IP6_ADDR_BLOCK3

`#define IP6_ADDR_BLOCK3( ip6addr)    ((u16_t)((lwip_htonl((ip6addr)->addr[1]) >> 16) & 0xff`

Access address in 16-bit block

4.165.6.5 IP6_ADDR_BLOCK4

`#define IP6_ADDR_BLOCK4( ip6addr)    ((u16_t)((lwip_htonl((ip6addr)->addr[1])) & 0xffff))`

Access address in 16-bit block

4.165.6.6 IP6_ADDR_BLOCK5

`#define IP6_ADDR_BLOCK5( ip6addr)    ((u16_t)((lwip_htonl((ip6addr)->addr[2]) >> 16) & 0xff`

Access address in 16-bit block

4.165.6.7 IP6_ADDR_BLOCK6

`#define IP6_ADDR_BLOCK6( ip6addr) ((u16_t)((lwip_htonl((ip6addr)->addr[2])) & 0xffff))`

Access address in 16-bit block

4.165.6.8 IP6_ADDR_BLOCK7

`#define IP6_ADDR_BLOCK7( ip6addr) ((u16_t)((lwip_htonl((ip6addr)->addr[3]) >> 16) & 0xff`

Access address in 16-bit block

4.165.6.9 IP6_ADDR_BLOCK8

`#define IP6_ADDR_BLOCK8( ip6addr) ((u16_t)((lwip_htonl((ip6addr)->addr[3])) & 0xffff))`

Access address in 16-bit block

4.165.6.10 ip6_addr_cmp

`#define ip6_addr_cmp( addr1, addr2) ip6_addr_eq(addr1, addr2)`

Deprecated Renamed to ip6_addr_eq

4.165.6.11 ip6_addr_cmp_packed

`#define ip6_addr_cmp_packed( ip6addr, paddr, zone_idx) ip6_addr_packed_eq(ip6addr, paddr`
`zone_idx)`

Deprecated Renamed to ip6_addr_packed_eq

4.165.6.12 ip6_addr_cmp_zoneless

`#define ip6_addr_cmp_zoneless( addr1, addr2) ip6_addr_zoneless_eq(addr1, addr2)`

Deprecated Renamed to ip6_addr_zoneless_eq

4.165.6.13 ip6_addr_copy

`#define ip6_addr_copy( dest, src)` **Value:**

```
do{(dest).addr[0] = (src).addr[0]; \
(dest).addr[1] = (src).addr[1]; \
(dest).addr[2] = (src).addr[2]; \
(dest).addr[3] = (src).addr[3]; \
ip6_addr_copy_zone((dest), (src)); }while(0)
```

Copy IPv6 address - faster than ip6_addr_set: no NULL check

4.165.6.14 ip6_addr_copy_from_packed

`#define ip6_addr_copy_from_packed( dest, src)` **Value:**

```
do{(dest).addr[0] = (src).addr[0]; \
(dest).addr[1] = (src).addr[1]; \
(dest).addr[2] = (src).addr[2]; \
(dest).addr[3] = (src).addr[3]; \
ip6_addr_clear_zone(&dest); }while(0)
```

Copy packed IPv6 address to unpacked IPv6 address; zone is not set

4.165.6.15 ip6_addr_copy_to_packed

`#define ip6_addr_copy_to_packed( dest, src)` **Value:**

```
do{(dest).addr[0] = (src).addr[0]; \
(dest).addr[1] = (src).addr[1]; \
(dest).addr[2] = (src).addr[2]; \
(dest).addr[3] = (src).addr[3]; }while(0)
```

Copy unpacked IPv6 address to packed IPv6 address; zone is lost

4.165.6.16 ip6_addr_eq

`#define ip6_addr_eq( addr1, addr2)` **Value:**

```
(ip6_addr_zoneless_eq((addr1), (addr2)) && \
ip6_addr_zone_eq((addr1), (addr2)))
```

Determine if two IPv6 addresses are the same. In particular, the address part of both must be the same, and the zone must be compatible.

Parameters

addr1	IPv6 address 1
addr2	IPv6 address 2

Returns 1 if the addresses are considered equal, 0 if not

4.165.6.17 ip6_addr_net_eq

`#define ip6_addr_net_eq( addr1, addr2)` **Value:**

```
(ip6_addr_net_zoneless_eq((addr1), (addr2)) && \
ip6_addr_zone_eq((addr1), (addr2)))
```

Determine if two IPv6 address are on the same network.

Parameters

addr1	IPv6 address 1
addr2	IPv6 address 2

Returns 1 if the network identifiers of both address match, 0 if not

4.165.6.18 ip6_addr_net_zoneless_eq

`#define ip6_addr_net_zoneless_eq( addr1, addr2)` **Value:**

```
(((addr1)->addr[0] == (addr2)->addr[0]) && \
((addr1)->addr[1] == (addr2)->addr[1]))
```

Compare IPv6 networks, ignoring zone information. To be used sparingly!

4.165.6.19 ip6_addr_netcmp

```
#define ip6_addr_netcmp( addr1, addr2)   ip6_addr_net_eq(addr1, addr2)
```

Determine if two IPv6 address are on the same network. Deprecated Renamed to ip6_addr_net_eq

4.165.6.20 ip6_addr_netcmp_zoneless

```
#define ip6_addr_netcmp_zoneless( addr1, addr2)   ip6_addr_net_zoneless_eq(addr1, addr2)
```

Deprecated Renamed to ip6_addr_net_zoneless_eq

4.165.6.21 ip6_addr_packed_eq

`#define ip6_addr_packed_eq( ip6addr, paddr, zone_idx)` **Value:**

```
(((ip6addr)->addr[0] == (paddr)->addr[0]) && \
((ip6addr)->addr[1] == (paddr)->addr[1]) && \
((ip6addr)->addr[2] == (paddr)->addr[2]) && \
((ip6addr)->addr[3] == (paddr)->addr[3]) && \
ip6_addr_equals_zone((ip6addr), (zone_idx)))
```

Compare IPv6 address to packed address and zone

4.165.6.22 IP6_ADDR_PART

```
#define IP6_ADDR_PART( ip6addr, index, a, b, c, d)    (ip6addr)->addr[index] = PP_HTONL(LW
```

Set an IPv6 partial address given by byte-parts

4.165.6.23 ip6_addr_set

`#define ip6_addr_set( dest, src)` **Value:**

```
do{(dest)->addr[0] = (src) == NULL ? 0 : (src)->addr[0]; ↩
     \
(dest)->addr[1] = (src) == NULL ? 0 : (src)->addr[1]; \
(dest)->addr[2] = (src) == NULL ? 0 : (src)->addr[2]; \
(dest)->addr[3] = (src) == NULL ? 0 : (src)->addr[3]; \
ip6_addr_set_zone((dest), (src) == NULL ? IP6_NO_ZONE :  ↩
    ip6_addr_zone(src)); }while(0)
```

Safely copy one IPv6 address to another (src may be NULL)

4.165.6.24 ip6_addr_set_any

```
#define ip6_addr_set_any( ip6addr)   ip6_addr_set_zero(ip6addr)
```

Set address to ipv6 ’any’ (no need for lwip_htonl())

4.165.6.25 ip6_addr_set_hton

`#define ip6_addr_set_hton( dest, src)` **Value:**

```
do{(dest)->addr[0] = (src) == NULL ? 0 : lwip_htonl ↩
    ((src)->addr[0]); \
(dest)->addr[1] = (src) == NULL ? 0 : lwip_htonl(( ↩
    src)->addr[1]); \
(dest)->addr[2] = (src) == NULL ? 0 : lwip_htonl(( ↩
    src)->addr[2]); \
(dest)->addr[3] = (src) == NULL ? 0 : lwip_htonl(( ↩
    src)->addr[3]); \
ip6_addr_set_zone((dest), (src) == NULL ? ↩
    IP6_NO_ZONE : ip6_addr_zone(src));}while(0)
```

Safely copy one IPv6 address to another and change byte order from host- to network-order.

4.165.6.26 ip6_addr_set_loopback

`#define ip6_addr_set_loopback( ip6addr)` **Value:**

```
do{(ip6addr)->addr[0] = 0; \
(ip6addr)->addr[1] = 0; \
(ip6addr)->addr[2] = 0; \
(ip6addr)->addr[3] = PP_HTONL(0x00000001UL); \
ip6_addr_clear_zone(ip6addr);}while(0)
```

Set address to ipv6 loopback address

4.165.6.27 ip6_addr_set_zero

`#define ip6_addr_set_zero( ip6addr)` **Value:**

```
do{(ip6addr)->addr[0] = 0; \
(ip6addr)->addr[1] = 0; \
(ip6addr)->addr[2] = 0; \
(ip6addr)->addr[3] = 0; \
ip6_addr_clear_zone(ip6addr);}while(0)
```

Set complete address to zero

4.165.6.28 ip6_addr_zoneless_eq

`#define ip6_addr_zoneless_eq( addr1, addr2)` **Value:**

```
(((addr1)->addr[0] == (addr2)->addr[0]) && \
((addr1)->addr[1] == (addr2)->addr[1]) && \
((addr1)->addr[2] == (addr2)->addr[2]) && \
((addr1)->addr[3] == (addr2)->addr[3]))
```

Compare IPv6 addresses, ignoring zone information. To be used sparingly!

4.165.7 Typedef Documentation

4.165.7.1 ip6_addr_t

`typedef struct ip6_addr ip6_addr_t`

IPv6 address

4.165.8 Function Documentation

4.165.8.1 ip6addr_aton()

```
int ip6addr_aton (const char * cp, ip6_addr_t * addr)
```

Check whether "cp" is a valid ascii representation of an IPv6 address and convert to a binary address. Returns 1 if the address is valid, 0 if not.

Parameters

cp	IPv6 address in ascii representation (e.g. "FF01::1")
addr	pointer to which to save the ip address in network order

Returns 1 if cp could be converted to addr, 0 on failure

4.165.8.2 ip6addr_ntoa()

```
char * ip6addr_ntoa (const ip6_addr_t * addr)
```

returns ptr to static buffer; not reentrant!

Convert numeric IPv6 address into ASCII representation. returns ptr to static buffer; not reentrant!

Parameters

addr	ip6 address in network order to convert

Returns pointer to a global static (!) buffer that holds the ASCII representation of addr

4.165.8.3 ip6addr_ntoa_r()

```
char * ip6addr_ntoa_r (const ip6_addr_t * addr, char * buf, int buflen)
```

Same as ipaddr_ntoa, but reentrant since a user-supplied buffer is used.

Parameters

addr	ip6 address in network order to convert
buf	target buffer where the string is stored
buflen	length of buf

Returns either pointer to buf which now holds the ASCII representation of addr or NULL if buf was too small

4.166 src/include/lwip/ip6_frag.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/ip6_addr.h"#include "lwip/ip6.h"# ↩
    include "lwip/netif.h"
```

4.166.1 Data Structures

- struct ip6_reassdata
- struct pbuf_custom_ref

4.166.2 Macros

- #define IP6_REASS_TMR_INTERVAL 1000
- #define IPV6_FRAG_COPYHEADER 0

4.166.3 Functions

- struct pbuf * ip6_reass (struct pbuf *p)
- err_t ip6_frag (struct pbuf *p, struct netif *netif, const ip6_addr_t *dest)

4.166.4 Detailed Description

IPv6 fragmentation and reassembly.

4.166.5 Macro Definition Documentation

4.166.5.1 IP6_REASS_TMR_INTERVAL

```
#define IP6_REASS_TMR_INTERVAL   1000
```

The IPv6 reassembly timer interval in milliseconds.

4.166.5.2 IPV6_FRAG_COPYHEADER

```
#define IPV6_FRAG_COPYHEADER   0
```

IP6_FRAG_COPYHEADER==1: for platforms where sizeof(void*) > 4, "struct ip6_reass_helper" is too large to be stored in the IPv6 fragment header, and will bleed into the header before it, which may be the IPv6 header or an extension header. This means that for each first fragment packet, we need to 1) make a copy of some IPv6 header fields (src+dest) that we need later on, just in case we do overwrite part of the IPv6 header, and 2) make a copy of the header data that we overwrote, so that we can restore it before either completing reassembly or sending an ICMPv6 reply. The last part is true even if this setting is disabled, but if it is enabled, we need to save a bit more data (up to the size of a pointer) because we overwrite more.

4.166.6 Function Documentation

4.166.6.1 ip6_frag()

```
err_t ip6_frag (struct pbuf * p, struct netif * netif, const ip6_addr_t * dest)
```

Fragment an IPv6 datagram if too large for the netif or path MTU.

Chop the datagram in MTU sized chunks and send them in order by pointing PBUF_REFs into p

Parameters

p	ipv6 packet to send
netif	the netif on which to send
dest	destination ipv6 address to which to send

Returns ERR_OK if sent successfully, err_t otherwise

4.166.6.2 ip6_reass()

```
struct pbuf * ip6_reass (struct pbuf * p)
```

Reassembles incoming IPv6 fragments into an IPv6 datagram.

Parameters

p	points to the IPv6 Fragment Header

Returns NULL if reassembly is incomplete, pbuf pointing to IPv6 Header if reassembly is complete

4.167 src/include/lwip/ip6_zone.h File Reference

4.167.1 Macros

- #define IP6_NO_ZONE 0
- #define IPADDR6_ZONE_INIT , IP6_NO_ZONE
- #define ip6_addr_zone(ip6addr) ((ip6addr)->zone)
- #define ip6_addr_has_zone(ip6addr) (ip6_addr_zone(ip6addr) != IP6_NO_ZONE)
- #define ip6_addr_set_zone(ip6addr, zone_idx) ((ip6addr)->zone = (zone_idx))
- #define ip6_addr_clear_zone(ip6addr) ((ip6addr)->zone = IP6_NO_ZONE)
- #define ip6_addr_copy_zone(ip6addr1, ip6addr2) ((ip6addr1).zone = (ip6addr2).zone)
- #define ip6_addr_equals_zone(ip6addr, zone_idx) ((ip6addr)->zone == (zone_idx))
- #define ip6_addr_cmp_zone(addr1, addr2) ip6_addr_zone_eq(ip6addr1, ip6addr2)
- #define ip6_addr_zone_eq(ip6addr1, ip6addr2) ((ip6addr1)->zone == (ip6addr2)->zone)
- #define IPV6_CUSTOM_SCOPES 0
- #define ip6_addr_has_scope(ip6addr, type)
- #define ip6_addr_assign_zone(ip6addr, type, netif)
- #define ip6_addr_test_zone(ip6addr, netif) (ip6_addr_equals_zone((ip6addr), netif_get_index(netif)))
- #define ip6_addr_lacks_zone(ip6addr, type) (!ip6_addr_has_zone(ip6addr) && ip6_addr_has_scope((ip6addr), (type)))
- #define ip6_addr_select_zone(dest, src)

4.167.2 Enumerations

- enum lwip_ipv6_scope_type { IP6_UNKNOWN = 0 , IP6_UNICAST = 1 , IP6_MULTICAST = 2 }

4.167.3 Detailed Description

IPv6 address scopes, zones, and scoping policy.

This header provides the means to implement support for IPv6 address scopes, as per RFC 4007. An address scope can be either global or more constrained. In lwIP, we say that an address "has a scope" or "is scoped" when its scope is constrained, in which case the address is meaningful only in a specific "zone." For unicast addresses, only link-local addresses have a scope; in that case, the scope is the link. For multicast addresses, there are various scopes defined by RFC 4007 and others. For any constrained scope, a system must establish a (potentially one-to-many) mapping between zones and local interfaces. For example, a link-local address is valid on only one link (its zone). That link may be attached to one or more local interfaces. The decisions on which scopes are constrained and the mapping between zones and interfaces is together what we refer to as the "scoping policy" - more on this in a bit.

In lwIP, each IPv6 address has an associated zone index. This zone index may be set to "no zone" (IP6_NO_ZONE, 0) or an actual zone. We say that an address "has a zone" or "is zoned" when its zone index is *not* set to "no zone." In lwIP, in principle, each address should be "properly zoned," which means that if the address has a zone if and only if has a scope. As such, it is a rule that an unscoped (e.g., global) address must never have a zone. Even though one could argue that there is always one zone even for global scopes, this rule exists for implementation simplicity. Violation of the rule will trigger assertions or otherwise result in undesired behavior.

Backward compatibility prevents us from requiring that applications always provide properly zoned addresses. We do enforce the rule that the in the lwIP link layer (everything below netif->output_ip6() and in particular ND6) *all* addresses are properly zoned. Thus, on the output paths down the stack, various places deal with the case of addresses that lack a zone. Some of them are best-effort for efficiency (e.g. the PCB bind and connect API calls' attempts to add missing zones); ultimately the IPv6 output handler (ip6_output_if_src) will set a zone if necessary.

Aside from dealing with scoped addresses lacking a zone, a proper IPv6 implementation must also ensure that a packet with a scoped source and/or destination address does not leave its zone. This is currently implemented in the input and forward functions. However, for output, these checks are deliberately omitted in order to keep the implementation lightweight. The routing algorithm in ip6_route will take decisions such that it will not cause zone violations unless the application sets bad addresses, though.

In terms of scoping policy, lwIP implements the default policy from RFC 4007 using macros in this file. This policy considers link-local unicast addresses and (only) interface-local and link-local multicast addresses as having a scope. For all these addresses, the zone is equal to the interface. As shown below in this file, it is possible to implement a custom policy.

4.168 src/include/lwip/ip_addr.h File Reference

```
#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/ip4_addr.h"#include "lwip/ip6_addr ↩
    .h"
```

4.168.1 Data Structures

- struct ip_addr

4.168.2 Macros

- #define ip_2_ip6(ipaddr)   (&((ipaddr)->u_addr.ip6))
- #define ip_2_ip4(ipaddr)   (&((ipaddr)->u_addr.ip4))
- #define ip_addr_netcmp(addr1, addr2, mask)   ip_addr_net_eq((addr1), (addr2), (mask))
- #define ip_addr_net_eq(addr1, addr2, mask)
- #define ip_addr_cmp(addr1, addr2)   ip_addr_eq((addr1), (addr2))
- #define ip_addr_eq(addr1, addr2)

- #define ip_addr_cmp_zoneless(addr1, addr2)   ip_addr_zoneless_eq((addr1), (addr2))
- #define ip_addr_zoneless_eq(addr1, addr2)
- #define ip_addr_isany(ipaddr)
- #define ip_addr_isany_val(ipaddr)
- #define ip_addr_isbroadcast(ipaddr, netif)
- #define ip_addr_ismulticast(ipaddr)
- #define ip_addr_isloopback(ipaddr)
- #define ip_addr_islinklocal(ipaddr)
- #define IP_ADDR_ANY IP4_ADDR_ANY
- #define IP4_ADDR_ANY   (&ip_addr_any)
- #define IP4_ADDR_ANY4   (ip_2_ip4(&ip_addr_any))
- #define IP6_ADDR_ANY   (&ip6_addr_any)
- #define IP6_ADDR_ANY6   (ip_2_ip6(&ip6_addr_any))
- #define IP_ANY_TYPE   (&ip_addr_any_type)

4.168.3 Typedefs

- typedef struct ip_addr ip_addr_t

4.168.4 Enumerations

- enum lwip_ip_addr_type { IPADDR_TYPE_V4 = 0U , IPADDR_TYPE_V6 = 6U , IPADDR_TYPE_ANY = 46U }

4.168.5 Functions

- char * ipaddr_ntoa (const ip_addr_t *addr)
- char * ipaddr_ntoa_r (const ip_addr_t *addr, char *buf, int buflen)
- int ipaddr_aton (const char *cp, ip_addr_t *addr)

4.168.6 Detailed Description

IP address API (common IPv4 and IPv6)

4.169 src/include/lwip/mem.h File Reference

```
#include "lwip/opt.h"
```

4.169.1 Functions

- void mem_init (void)
- void * mem_trim (void *mem, mem_size_t size)
- void * mem_malloc (mem_size_t size)
- void * mem_calloc (mem_size_t count, mem_size_t size)
- void mem_free (void *mem)

4.169.2 Detailed Description

Heap API

4.169.3 Function Documentation

4.169.3.1 mem_calloc()

```
void * mem_calloc (mem_size_t count, mem_size_t size)
```

Contiguously allocates enough space for count objects that are size bytes of memory each and returns a pointer to the allocated memory.

The allocated memory is filled with bytes of value zero.

Parameters

count	number of objects to allocate
size	size of the objects to allocate

Returns pointer to allocated memory / NULL pointer if there is an error

4.169.3.2 mem_free()

```
void mem_free (void * rmem)
```

Put a struct mem back on the heap

Parameters

rmem	is the data portion of a struct mem as returned by a previous call to mem_malloc()

4.169.3.3 mem_init()

```
void mem_init (void )
```

Zero the heap and initialize start, end and lowest-free

size_in	is the minimum size of the requested block in bytes.

4.169.3.4 mem_malloc()

```
void * mem_malloc (mem_size_t size_in)
```

Allocate a block of memory with a minimum of 'size' bytes.

Parameters

Returns pointer to allocated memory or NULL if no free memory was found.

Note that the returned value will always be aligned (as defined by MEM_ALIGNMENT).

4.169.3.5 mem_trim()

```
void * mem_trim (void * rmem, mem_size_t new_size)
```

Shrink memory returned by mem_malloc().

Parameters

rmem	pointer to memory allocated by mem_malloc the is to be shrunk
new_size	required size after shrinking (needs to be smaller than or equal to the previous size)

Returns for compatibility reasons: is always == rmem, at the moment or NULL if newsize is > old size, in which case rmem is NOT touched or freed!

4.170 src/include/lwip/memp.h File Reference

```
#include "lwip/opt.h"#include "lwip/priv/memp_std.h"#include "lwip/priv/memp_priv.h"# ↩
    include "lwip/stats.h"
```

4.170.1 Macros

- #define LWIP_MEMPOOL_PROTOTYPE(name) extern const struct memp_desc memp_ ## name
- #define LWIP_MEMPOOL_DECLARE(name, num, size, desc)
- #define LWIP_MEMPOOL_INIT(name) memp_init_pool(&memp_ ## name)
- #define LWIP_MEMPOOL_ALLOC(name) memp_malloc_pool(&memp_ ## name)
- #define LWIP_MEMPOOL_FREE(name, x) memp_free_pool(&memp_ ## name, (x))

4.170.2 Enumerations

- enum memp_t

4.170.3 Functions

- void memp_init (void)
- void * memp_malloc (memp_t type)
- void memp_free (memp_t type, void *mem)

4.170.4 Detailed Description

Memory pool API

4.170.5 Enumeration Type Documentation

4.170.5.1 memp_t

`enum memp_t`

Create the list of all memory pools managed by memp. MEMP_MAX represents a NULL pool at the end

4.170.6 Function Documentation

4.170.6.1 memp_free()

`void memp_free (memp_t type, void * mem)`

Put an element back into its pool.

Parameters

type	the pool where to put mem
mem	the memp element to free

4.170.6.2 memp_init()

`void memp_init (void )`

Initializes lwIP built-in pools. Related functions: memp_malloc, memp_free

Carves out memp_memory into linked lists for each pool-type.

4.170.6.3 memp_malloc()

`void * memp_malloc (memp_t type)`

Get an element from a specific pool.

Parameters

type	the pool to get an element from

Returns a pointer to the allocated memory or a NULL pointer on error

4.171 src/include/lwip/mld6.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/netif.h"
```

4.171.1 Data Structures

- struct mld_group

4.171.2 Macros

- #define netif_mld6_data(netif) ((struct mld_group *)netif_get_client_data(netif, LWIP_NETIF_CLIENT_DATA_INDEX_MLD6))

4.171.3 Functions

- err_t mld6_stop (struct netif *netif)
- void mld6_report_groups (struct netif *netif)
- void mld6_tmr (void)
- struct mld_group * mld6_lookfor_group (struct netif *ifp, const ip6_addr_t *addr)
- void mld6_input (struct pbuf *p, struct netif *inp)
- err_t mld6_joingroup (const ip6_addr_t *srcaddr, const ip6_addr_t *groupaddr)
- err_t mld6_joingroup_netif (struct netif *netif, const ip6_addr_t *groupaddr)
- err_t mld6_leavegroup (const ip6_addr_t *srcaddr, const ip6_addr_t *groupaddr)
- err_t mld6_leavegroup_netif (struct netif *netif, const ip6_addr_t *groupaddr)

4.171.4 Detailed Description

Multicast listener discovery for IPv6. Aims to be compliant with RFC 2710. No support for MLDv2.

4.171.5 Function Documentation

4.171.5.1 mld6_input()

```
void mld6_input (struct pbuf * p, struct netif * inp)
```

Process an input MLD message. Called by icmp6_input.

Parameters

p	the mld packet, p->payload pointing to the icmpv6 header
inp	the netif on which this packet was received

4.171.5.2 mld6_lookfor_group()

```
struct mld_group * mld6_lookfor_group (struct netif * ifp, const ip6_addr_t * addr)
```

Search for a group that is joined on a netif

Parameters

ifp	the network interface for which to look
addr	the group ipv6 address to search for

Returns a struct mld_group* if the group has been found, NULL if the group wasn't found.

4.171.5.3 mld6_report_groups()

```
void mld6_report_groups (struct netif * netif)
```

Report MLD memberships for this interface

Parameters

netif	network interface on which report MLD memberships

4.171.5.4 mld6_stop()

```
err_t mld6_stop (struct netif * netif)
```

Stop MLD processing on interface

Parameters

netif	network interface on which stop MLD processing

4.171.5.5 mld6_tmr()

```
void mld6_tmr (void )
```

Periodic timer for mld processing. Must be called every MLD6_TMR_INTERVAL milliseconds (100).

When a delaying member expires, a membership report is sent.

4.172 src/include/lwip/prot/mld6.h File Reference

```
#include "lwip/arch.h"#include "lwip/prot/ip6.h"#include "arch/bpstruct.h"#include "arch/ ↩
    epstruct.h"
```

4.172.1 Data Structures

- struct mld_header

4.172.2 Detailed Description

MLD6 protocol definitions

4.173 src/include/lwip/nd6.h File Reference

```
#include "lwip/opt.h"#include "lwip/ip6_addr.h"#include "lwip/err.h"
```

4.173.1 Macros

- #define ND6_TMR_INTERVAL 1000
- #define ND6_RTR_SOLICITATION_INTERVAL 4000

4.173.2 Functions

- void nd6_tmr (void)
- void nd6_input (struct pbuf *p, struct netif *inp)
- void nd6_clear_destination_cache (void)
- struct netif * nd6_find_route (const ip6_addr_t *ip6addr)
- err_t nd6_get_next_hop_addr_or_queue (struct netif *netif, struct pbuf *q, const ip6_addr_t *ip6addr, const u8_t **hwaddrp)
- u16_t nd6_get_destination_mtu (const ip6_addr_t *ip6addr, struct netif *netif)
- void nd6_reachability_hint (const ip6_addr_t *ip6addr)
- void nd6_cleanup_netif (struct netif *netif)
- void nd6_adjust_mld_membership (struct netif *netif, s8_t addr_idx, u8_t new_state)
- void nd6_restart_netif (struct netif *netif)

4.173.3 Detailed Description

Neighbor discovery and stateless address autoconfiguration for IPv6. Aims to be compliant with RFC 4861 (Neighbor discovery) and RFC 4862 (Address autoconfiguration).

4.173.4 Macro Definition Documentation

4.173.4.1 ND6_RTR_SOLICITATION_INTERVAL

```
#define ND6_RTR_SOLICITATION_INTERVAL   4000
```

Router solicitations are sent in 4 second intervals (see RFC 4861, ch. 6.3.7)

4.173.4.2 ND6_TMR_INTERVAL

```
#define ND6_TMR_INTERVAL   1000
```

1 second period

4.173.5 Function Documentation

4.173.5.1 nd6_adjust_mld_membership()

```
void nd6_adjust_mld_membership (struct netif * netif, s8_t addr_idx, u8_t new_state)
```

The state of a local IPv6 address entry is about to change. If needed, join or leave the solicited-node multicast group for the address.

Parameters

netif	The netif that owns the address.
addr_idx	The index of the address.
new_state	The new (IP6_ADDR_) state for the address.

4.173.5.2 nd6_cleanup_netif()

`void nd6_cleanup_netif (struct netif * netif)`

Remove all prefix, neighbor_cache and router entries of the specified netif.

Parameters

netif	points to a network interface

4.173.5.3 nd6_clear_destination_cache()

`void nd6_clear_destination_cache (void )`

Clear the destination cache.

This operation may be necessary for consistency in the light of changing local addresses and/or use of the gateway hook.

4.173.5.4 nd6_find_route()

`struct netif * nd6_find_route (const ip6_addr_t * ip6addr)`

Find a router-announced route to the given destination. This route may be based on an on-link prefix or a default router.

If a suitable route is found, the returned netif is guaranteed to be in a suitable state (up, link up) to be used for packet transmission.

Parameters

ip6addr	the destination IPv6 address

Returns the netif to use for the destination, or NULL if none found

4.173.5.5 nd6_get_destination_mtu()

`u16_t nd6_get_destination_mtu (const ip6_addr_t * ip6addr, struct netif * netif)`

Get the Path MTU for a destination.

Parameters

ip6addr	the destination address
netif	the netif on which the packet will be sent

Returns the Path MTU, if known, or the netif default MTU

4.173.5.6 nd6_get_next_hop_addr_or_queue()

```
err_t nd6_get_next_hop_addr_or_queue (struct netif * netif, struct pbuf * q, const ip6_add
* ip6addr, const u8_t ** hwaddrp)
```

A packet is to be transmitted to a specific IPv6 destination on a specific interface. Check if we can find the hardware address of the next hop to use for the packet. If so, give the hardware address to the caller, which should use it to send the packet right away. Otherwise, enqueue the packet for later transmission while looking up the hardware address, if possible.

As such, this function returns one of three different possible results:

- ERR_OK with a non-NULL 'hwaddrp': the caller should send the packet now.
- ERR_OK with a NULL 'hwaddrp': the packet has been enqueued for later.
- not ERR_OK: something went wrong; forward the error upward in the stack.

Parameters

netif	The lwIP network interface on which the IP packet will be sent.
q	The pbuf(s) containing the IP packet to be sent.
ip6addr	The destination IPv6 address of the packet.
hwaddrp	On success, filled with a pointer to a HW address or NULL (meaning the packet has been queued).

Returns

- ERR_OK on success, ERR_RTE if no route was found for the packet, or ERR_MEM if low memory conditions prohibit sending the packet at all.

4.173.5.7 nd6_input()

```
void nd6_input (struct pbuf * p, struct netif * inp)
```

Process an incoming neighbor discovery message

Parameters

p	the nd packet, p->payload pointing to the icmpv6 header
inp	the netif on which this packet was received

4.173.5.8 nd6_reachability_hint()

```
void nd6_reachability_hint (const ip6_addr_t * ip6addr)
```

Provide the Neighbor discovery process with a hint that a destination is reachable. Called by tcp_receive when ACKs are received or sent (as per RFC). This is useful to avoid sending NS messages every 30 seconds.

Parameters

ip6addr	the destination address which is know to be reachable by an upper layer protocol (TCP)

4.173.5.9 nd6_restart_netif()

```
void nd6_restart_netif (struct netif * netif)
```

Netif was added, set up, or reconnected (link up)

4.173.5.10 nd6_tmr()

```
void nd6_tmr (void )
```

Periodic timer for Neighbor discovery functions:

- Update neighbor reachability states
- Update destination cache entries age
- Update invalidation timers of default routers and on-link prefixes
- Update lifetimes of our addresses
- Perform duplicate address detection (DAD) for our addresses
- Send router solicitations

4.174 src/include/lwip/prot/nd6.h File Reference

```
#include "lwip/arch.h"#include "lwip/ip6_addr.h"#include "lwip/prot/ip6.h"#include "arch/ ↩
    bpstruct.h"#include "arch/epstruct.h"
```

4.174.1 Data Structures

- struct ns_header
- struct na_header
- struct rs_header
- struct redirect_header

4.174.2 Macros

- #define ND6_RA_FLAG_MANAGED_ADDR_CONFIG (0x80)
- #define ND6_OPTION_TYPE_SOURCE_LLADDR (0x01)
- #define ND6_OPTION_TYPE_PREFIX_INFO (0x03)
- #define ND6_OPTION_TYPE_REDIR_HDR (0x04)
- #define ND6_OPTION_TYPE_MTU (0x05)
- #define ND6_OPTION_TYPE_ROUTE_INFO (24)
- #define ND6_OPTION_TYPE_RDNSS (25)

4.174.3 Detailed Description

ND6 protocol definitions

4.174.4 Macro Definition Documentation

4.174.4.1 ND6_OPTION_TYPE_MTU

```
#define ND6_OPTION_TYPE_MTU   (0x05)
```

MTU option.

4.174.4.2 ND6_OPTION_TYPE_PREFIX_INFO

```
#define ND6_OPTION_TYPE_PREFIX_INFO   (0x03)
```

Prefix information option.

4.174.4.3 ND6_OPTION_TYPE_RDNSS

```
#define ND6_OPTION_TYPE_RDNSS   (25)
```

Recursive DNS Server Option.

4.174.4.4 ND6_OPTION_TYPE_REDIR_HDR

```
#define ND6_OPTION_TYPE_REDIR_HDR   (0x04)
```

Redirected header option.

4.174.4.5 ND6_OPTION_TYPE_ROUTE_INFO

```
#define ND6_OPTION_TYPE_ROUTE_INFO   (24)
```

Route information option.

4.174.4.6 ND6_OPTION_TYPE_SOURCE_LLADDR

```
#define ND6_OPTION_TYPE_SOURCE_LLADDR   (0x01)
```

Link-layer address option.

4.174.4.7 ND6_RA_FLAG_MANAGED_ADDR_CONFIG

```
#define ND6_RA_FLAG_MANAGED_ADDR_CONFIG   (0x80)
```

Router advertisement message header.

4.175 src/include/lwip/netbuf.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/ip_addr.h"#include "lwip/ip6_addr ↩
    .h"
```

4.175.1 Data Structures

- struct netbuf

4.175.2 Macros

- #define NETBUF_FLAG_DESTADDR 0x01
- #define NETBUF_FLAG_CHKSUM 0x02

4.175.3 Functions

- struct netbuf * netbuf_new (void)
- void netbuf_delete (struct netbuf *buf)
- void * netbuf_alloc (struct netbuf *buf, u16_t size)
- void netbuf_free (struct netbuf *buf)
- err_t netbuf_ref (struct netbuf *buf, const void *dataptr, u16_t size)
- void netbuf_chain (struct netbuf *head, struct netbuf *tail)
- err_t netbuf_data (struct netbuf *buf, void **dataptr, u16_t *len)
- s8_t netbuf_next (struct netbuf *buf)
- void netbuf_first (struct netbuf *buf)

4.175.4 Detailed Description

netbuf API (for netconn API)

4.175.5 Macro Definition Documentation

4.175.5.1 NETBUF_FLAG_CHKSUM

```
#define NETBUF_FLAG_CHKSUM   0x02
```

This netbuf includes a checksum

4.175.5.2 NETBUF_FLAG_DESTADDR

```
#define NETBUF_FLAG_DESTADDR   0x01
```

This netbuf has dest-addr/port set

4.176 src/include/lwip/netif.h File Reference

```
#include "lwip/opt.h"#include "lwip/err.h"#include "lwip/ip_addr.h"#include "lwip/def.h"# ↩
    include "lwip/pbuf.h"#include "lwip/stats.h"
```

4.176.1 Data Structures

- struct netif
- union netif_ext_callback_args_t
- struct netif_ext_callback_args_t::link_changed_s
- struct netif_ext_callback_args_t::status_changed_s
- struct netif_ext_callback_args_t::ipv4_changed_s
- struct netif_ext_callback_args_t::ipv6_set_s
- struct netif_ext_callback_args_t::ipv6_addr_state_changed_s

4.176.2 Macros

- #define NETIF_MAX_HWADDR_LEN 6U
- #define NETIF_NAMESIZE 6
- #define NETIF_FLAG_UP 0x01U
- #define NETIF_FLAG_BROADCAST 0x02U
- #define NETIF_FLAG_LINK_UP 0x04U
- #define NETIF_FLAG_ETHARP 0x08U
- #define NETIF_FLAG_ETHERNET 0x10U
- #define NETIF_FLAG_IGMP 0x20U
- #define NETIF_FLAG_MLD6 0x40U
- #define netif_set_client_data(netif, id, data) netif_get_client_data(netif, id) = (data)
- #define netif_get_client_data(netif, id) (netif)->client_data[(id)]
- #define netif_is_up(netif) (((netif)->flags & NETIF_FLAG_UP) ? (u8_t)1 : (u8_t)0)
- #define netif_is_link_up(netif) (((netif)->flags & NETIF_FLAG_LINK_UP) ? (u8_t)1 : (u8_t)0)
- #define netif_set_igmp_mac_filter(netif, function) do { if((netif) != NULL) { (netif)->igmp_mac_filter = function; }}while(0)
- #define netif_get_igmp_mac_filter(netif) (((netif) != NULL) ? ((netif)->igmp_mac_filter) : NULL)
- #define netif_set_mld_mac_filter(netif, function) do { if((netif) != NULL) { (netif)->mld_mac_filter = function; }}while(0)
- #define netif_get_mld_mac_filter(netif) (((netif) != NULL) ? ((netif)->mld_mac_filter) : NULL)
- #define LWIP_NSC_NETIF_ADDED 0x0001
- #define LWIP_NSC_NETIF_REMOVED 0x0002
- #define LWIP_NSC_LINK_CHANGED 0x0004
- #define LWIP_NSC_STATUS_CHANGED 0x0008
- #define LWIP_NSC_IPV4_ADDRESS_CHANGED 0x0010
- #define LWIP_NSC_IPV4_GATEWAY_CHANGED 0x0020
- #define LWIP_NSC_IPV4_NETMASK_CHANGED 0x0040
- #define LWIP_NSC_IPV4_SETTINGS_CHANGED 0x0080
- #define LWIP_NSC_IPV6_SET 0x0100
- #define LWIP_NSC_IPV6_ADDR_STATE_CHANGED 0x0200
- #define LWIP_NSC_IPV4_ADDR_VALID 0x0400

4.176.3 Typedefs

- typedef err_t(* netif_init_fn) (struct netif *netif)
- typedef err_t(* netif_input_fn) (struct pbuf *p, struct netif *inp)
- typedef err_t(* netif_output_fn) (struct netif *netif, struct pbuf *p, const ip4_addr_t *ipaddr)
- typedef err_t(* netif_output_ip6_fn) (struct netif *netif, struct pbuf *p, const ip6_addr_t *ipaddr)
- typedef err_t(* netif_linkoutput_fn) (struct netif *netif, struct pbuf *p)
- typedef void(* netif_status_callback_fn) (struct netif *netif)
- typedef err_t(* netif_igmp_mac_filter_fn) (struct netif *netif, const ip4_addr_t *group, enum netif_mac_filter_action action)
- typedef err_t(* netif_mld_mac_filter_fn) (struct netif *netif, const ip6_addr_t *group, enum netif_mac_filter_action action)
- typedef u16_t netif_nsc_reason_t
- typedef void(* netif_ext_callback_fn) (struct netif *netif, netif_nsc_reason_t reason, const netif_ext_callback_args_t *args)

4.176.4 Enumerations

- enum netif_mac_filter_action { NETIF_DEL_MAC_FILTER = 0 , NETIF_ADD_MAC_FILTER = 1 }

4.176.5 Functions

- u8_t netif_alloc_client_data_id (void)
- struct netif * netif_add_noaddr (struct netif *netif, void *state, netif_init_fn init, netif_input_fn input)
- struct netif * netif_add (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw, void *state, netif_init_fn init, netif_input_fn input)
- void netif_set_addr (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw)
- void netif_remove (struct netif *netif)
- struct netif * netif_find (const char *name)
- void netif_set_default (struct netif *netif)
- void netif_set_ipaddr (struct netif *netif, const ip4_addr_t *ipaddr)
- void netif_set_netmask (struct netif *netif, const ip4_addr_t *netmask)
- void netif_set_gw (struct netif *netif, const ip4_addr_t *gw)
- void netif_set_up (struct netif *netif)
- void netif_set_down (struct netif *netif)
- void netif_set_status_callback (struct netif *netif, netif_status_callback_fn status_callback)
- void netif_set_remove_callback (struct netif *netif, netif_status_callback_fn remove_callback)
- void netif_set_link_up (struct netif *netif)
- void netif_set_link_down (struct netif *netif)
- void netif_set_link_callback (struct netif *netif, netif_status_callback_fn link_callback)
- err_t netif_loop_output (struct netif *netif, struct pbuf *p)

- void netif_poll (struct netif *netif)
- err_t netif_input (struct pbuf *p, struct netif *inp)
- void netif_ip6_addr_set (struct netif *netif, s8_t addr_idx, const ip6_addr_t *addr6)
- void netif_ip6_addr_set_state (struct netif *netif, s8_t addr_idx, u8_t state)
- s8_t netif_get_ip6_addr_match (struct netif *netif, const ip6_addr_t *ip6addr)
- void netif_create_ip6_linklocal_address (struct netif *netif, u8_t from_mac_48bit)
- err_t netif_add_ip6_address (struct netif *netif, const ip6_addr_t *ip6addr, s8_t *chosen_idx)
- u8_t netif_name_to_index (const char *name)
- char * netif_index_to_name (u8_t idx, char *name)
- struct netif * netif_get_by_index (u8_t idx)
- void netif_add_ext_callback (netif_ext_callback_t *callback, netif_ext_callback_fn fn)
- void netif_remove_ext_callback (netif_ext_callback_t *callback)
- void netif_invoke_ext_callback (struct netif *netif, netif_nsc_reason_t reason, const netif_ext_callback_args_t *args)

4.176.6 Variables

- struct netif * netif_list
- struct netif * netif_default

4.176.7 Detailed Description

netif API (to be used from TCPIP thread)

4.176.8 Macro Definition Documentation

4.176.8.1 LWIP_NSC_IPV4_ADDR_VALID

```
#define LWIP_NSC_IPV4_ADDR_VALID   0x0400
```

IPv4 settings: valid address set, application may start to communicate

4.176.8.2 LWIP_NSC_IPV4_ADDRESS_CHANGED

```
#define LWIP_NSC_IPV4_ADDRESS_CHANGED   0x0010
```

IPv4 address has changed

4.176.8.3 LWIP_NSC_IPV4_GATEWAY_CHANGED

```
#define LWIP_NSC_IPV4_GATEWAY_CHANGED   0x0020
```

IPv4 gateway has changed

4.176.8.4 LWIP_NSC_IPV4_NETMASK_CHANGED

```
#define LWIP_NSC_IPV4_NETMASK_CHANGED   0x0040
```

IPv4 netmask has changed

4.176.8.5 LWIP_NSC_IPV4_SETTINGS_CHANGED

```
#define LWIP_NSC_IPV4_SETTINGS_CHANGED   0x0080
```

called AFTER IPv4 address/gateway/netmask changes have been applied

4.176.8.6 LWIP_NSC_IPV6_ADDR_STATE_CHANGED

```
#define LWIP_NSC_IPV6_ADDR_STATE_CHANGED   0x0200
```

IPv6 address state has changed

4.176.8.7 LWIP_NSC_IPV6_SET

```
#define LWIP_NSC_IPV6_SET   0x0100
```

IPv6 address was added

4.176.8.8 LWIP_NSC_LINK_CHANGED

```
#define LWIP_NSC_LINK_CHANGED   0x0004
```

link changed

4.176.8.9 LWIP_NSC_NETIF_ADDED

```
#define LWIP_NSC_NETIF_ADDED   0x0001
```

netif was added. arg: NULL. Called AFTER netif was added.

4.176.8.10 LWIP_NSC_NETIF_REMOVED

```
#define LWIP_NSC_NETIF_REMOVED   0x0002
```

netif was removed. arg: NULL. Called BEFORE netif is removed.

4.176.8.11 LWIP_NSC_STATUS_CHANGED

```
#define LWIP_NSC_STATUS_CHANGED   0x0008
```

netif administrative status changed. up is called AFTER netif is set up. down is called BEFORE the netif is actually set down.

4.176.8.12 netif_get_igmp_mac_filter

```
#define netif_get_igmp_mac_filter( netif)   (((netif) != NULL) ? ((netif)->igmp_mac_filter
:  NULL)
```

Get the igmp mac filter function for a netif.

4.176.8.13 netif_get_mld_mac_filter

```
#define netif_get_mld_mac_filter( netif)   (((netif) != NULL) ? ((netif)->mld_mac_filter)
:  NULL)
```

Get the mld mac filter function for a netif.

4.176.8.14 netif_is_link_up

```
#define netif_is_link_up( netif)   (((netif)->flags & NETIF_FLAG_LINK_UP) ? (u8_t)1 :  (u8
```

Ask if a link is up

4.176.8.15 NETIF_MAX_HWADDR_LEN

```
#define NETIF_MAX_HWADDR_LEN   6U
```

Must be the maximum of all used hardware address lengths across all types of interfaces in use. This does not have to be changed, normally.

4.176.8.16 NETIF_NAMESIZE

```
#define NETIF_NAMESIZE   6
```

The size of a fully constructed netif name which the netif can be identified by in APIs. Composed of 2 chars, 3 (max) digits, and 1 \0

4.176.9 Typedef Documentation

4.176.9.1 netif_igmp_mac_filter_fn

```
typedef err_t(* netif_igmp_mac_filter_fn) (struct netif *netif, const ip4_addr_t *group,
enum netif_mac_filter_action action)
```

Function prototype for netif igmp_mac_filter functions

4.176.9.2 netif_init_fn

```
typedef err_t(* netif_init_fn) (struct netif *netif)
```

Function prototype for netif init functions. Set up flags and output/linkoutput callback functions in this function.

Parameters

netif	The netif to initialize

4.176.9.3 netif_input_fn

```
typedef err_t(* netif_input_fn) (struct pbuf *p, struct netif *inp)
```

Function prototype for netif->input functions. This function is saved as 'input' callback function in the netif struct. Call it when a packet has been received.

Parameters

Returns ERR_OK if the packet was handled != ERR_OK is the packet was NOT handled, in this case, the caller has to free the pbuf

p	The received packet, copied into a pbuf
inp	The netif which received the packet

4.176.9.4 netif_linkoutput_fn

```
typedef err_t(* netif_linkoutput_fn) (struct netif *netif, struct pbuf *p)
```

Function prototype for netif->linkoutput functions. Only used for ethernet netifs. This function is called by ARP when a packet shall be sent.

Parameters

netif	The netif which shall send a packet
p	The packet to send (raw ethernet packet)

4.176.9.5 netif_mld_mac_filter_fn

```
typedef err_t(* netif_mld_mac_filter_fn) (struct netif *netif, const ip6_addr_t *group,
enum netif_mac_filter_action action)
```

Function prototype for netif mld_mac_filter functions

4.176.9.6 netif_output_fn

```
typedef err_t(* netif_output_fn) (struct netif *netif, struct pbuf *p, const ip4_addr_t
*ipaddr)
```

Function prototype for netif->output functions. Called by lwIP when a packet shall be sent. For ethernet netif, set this to 'etharp_output' and set 'linkoutput'.

Parameters

netif	The netif which shall send a packet
p	The packet to send (p->payload points to IP header)
ipaddr	The IP address to which the packet shall be sent

4.176.9.7 netif_output_ip6_fn

```
typedef err_t(* netif_output_ip6_fn) (struct netif *netif, struct pbuf *p, const ip6_addr_
*ipaddr)
```

Function prototype for netif->output_ip6 functions. Called by lwIP when a packet shall be sent. For ethernet netif, set this to 'ethip6_output' and set 'linkoutput'.

Parameters

4.176.9.8 netif_status_callback_fn

```
typedef void(* netif_status_callback_fn) (struct netif *netif)
```

Function prototype for netif status- or link-callback functions.

netif	The netif which shall send a packet
p	The packet to send (p->payload points to IP header)
ipaddr	The IPv6 address to which the packet shall be sent

4.176.10 Enumeration Type Documentation

4.176.10.1 netif_mac_filter_action

`enum netif_mac_filter_action`

MAC Filter Actions, these are passed to a netif's igmp_mac_filter or mld_mac_filter callback function.

NETIF_DEL_MAC_FILTER	Delete a filter entry
NETIF_ADD_MAC_FILTER	Add a filter entry

4.176.11 Function Documentation

4.176.11.1 netif_get_ip6_addr_match()

`s8_t netif_get_ip6_addr_match (struct netif * netif, const ip6_addr_t * ip6addr)`

Checks if a specific local address is present on the netif and returns its index. Depending on its state, it may or may not be assigned to the interface (as per RFC terminology).

The given address may or may not be zoned (i.e., have a zone index other than IP6_NO_ZONE). If the address is zoned, it must have the correct zone for the given netif, or no match will be found.

Parameters

netif	the netif to check
ip6addr	the IPv6 address to find

Returns >= 0: address found, this is its index -1: address not found on this netif

4.176.11.2 netif_invoke_ext_callback()

`void netif_invoke_ext_callback (struct netif * netif, netif_nsc_reason_t reason, const netif_ext_callback_args_t * args)`

Invoke extended netif status event

Parameters

netif	netif that is affected by change
reason	change reason
args	depends on reason, see reason description

4.176.11.3 netif_poll()

`void netif_poll (struct netif * netif)`

Call netif_poll() in the main loop of your application. This is to prevent reentering non-reentrant functions like tcp_input(). Packets passed to netif_loop_output() are put on a list that is passed to netif->input() by netif_poll().

4.176.12 Variable Documentation

4.176.12.1 netif_default

`struct netif* netif_default[extern]`

The default network interface.

4.176.12.2 netif_list

```
struct netif* netif_list[extern]
```

The list of network interfaces.

4.177 src/include/lwip/netifapi.h File Reference

```
#include "lwip/opt.h"#include "lwip/sys.h"#include "lwip/netif.h"#include "lwip/dhcp.h"# ↩
    include "lwip/autoip.h"#include "lwip/priv/tcpip_priv.h"#include "lwip/priv/api_msg.h"# ↩
    include "lwip/prot/ethernet.h"
```

4.177.1 Macros

- #define netifapi_netif_remove(n) netifapi_netif_common(n, netif_remove, NULL)
- #define netifapi_netif_set_up(n) netifapi_netif_common(n, netif_set_up, NULL)
- #define netifapi_netif_set_down(n) netifapi_netif_common(n, netif_set_down, NULL)
- #define netifapi_netif_set_default(n) netifapi_netif_common(n, netif_set_default, NULL)
- #define netifapi_netif_set_link_up(n) netifapi_netif_common(n, netif_set_link_up, NULL)
- #define netifapi_netif_set_link_down(n) netifapi_netif_common(n, netif_set_link_down, NULL)
- #define netifapi_dhcp_start(n) netifapi_netif_common(n, NULL, dhcp_start)
- #define netifapi_dhcp_stop(n) netifapi_netif_common(n, dhcp_stop, NULL)
- #define netifapi_dhcp_inform(n) netifapi_netif_common(n, dhcp_inform, NULL)
- #define netifapi_dhcp_renew(n) netifapi_netif_common(n, NULL, dhcp_renew)
- #define netifapi_dhcp_release(n) netifapi_netif_common(n, NULL, dhcp_release)
- #define netifapi_dhcp_release_and_stop(n) netifapi_netif_common(n, dhcp_release_and_stop, NULL)
- #define netifapi_autoip_start(n) netifapi_netif_common(n, NULL, autoip_start)
- #define netifapi_autoip_stop(n) netifapi_netif_common(n, NULL, autoip_stop)

4.177.2 Functions

- err_t netifapi_arp_add (const ip4_addr_t *ipaddr, struct eth_addr *ethaddr, enum netifapi_arp_entry type)
- err_t netifapi_arp_remove (const ip4_addr_t *ipaddr, enum netifapi_arp_entry type)
- err_t netifapi_netif_add (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw, void *state, netif_init_fn init, netif_input_fn input)
- err_t netifapi_netif_set_addr (struct netif *netif, const ip4_addr_t *ipaddr, const ip4_addr_t *netmask, const ip4_addr_t *gw)
- err_t netifapi_netif_common (struct netif *netif, netifapi_void_fn voidfunc, netifapi_errt_fn errtfunc)
- err_t netifapi_netif_name_to_index (const char *name, u8_t *idx)
- err_t netifapi_netif_index_to_name (u8_t idx, char *name)

4.177.3 Detailed Description

netif API (to be used from non-TCPIP threads)

4.177.4 Function Documentation

4.177.4.1 netifapi_arp_add()

```
err_t netifapi_arp_add (const ip4_addr_t * ipaddr, struct eth_addr * ethaddr, enum netifap
type)
```

Add or update an entry in the ARP cache. For an update, ipaddr is used to find the cache entry.

Parameters

ipaddr	IPv4 address of cache entry
ethaddr	hardware address mapped to ipaddr
type	type of ARP cache entry

Returns ERR_OK: entry added/updated, else error from err_t

4.177.4.2 netifapi_arp_remove()

```
err_t netifapi_arp_remove (const ip4_addr_t * ipaddr, enum netifapi_arp_entry type)
```

Remove an entry in the ARP cache identified by ipaddr

Parameters

ipaddr	IPv4 address of cache entry
type	type of ARP cache entry

Returns ERR_OK: entry removed, else error from err_t

4.177.4.3 netifapi_netif_common()

```
err_t netifapi_netif_common (struct netif * netif, netifapi_void_fn voidfunc, netifapi_err
errtfunc)
```

call the "errtfunc" (or the "voidfunc" if "errtfunc" is NULL) in a thread-safe way by running that function inside the tcpip_thread context.

Note
use only for functions where there is only "netif" parameter.

4.178 src/include/lwip/opt.h File Reference

```
#include "lwipopts.h"#include "lwip/debug.h"
```

4.178.1 Macros

- #define NO_SYS 0
- #define LWIP_TIMERS 1
- #define LWIP_TIMERS_CUSTOM 0
- #define MEMCPY(dst, src, len) memcpy(dst,src,len)
- #define SMEMCPY(dst, src, len) memcpy(dst,src,len)
- #define MEMMOVE(dst, src, len) memmove(dst,src,len)
- #define LWIP_MPU_COMPATIBLE 0
- #define LWIP_TCPIP_CORE_LOCKING 1
- #define LWIP_TCPIP_CORE_LOCKING_INPUT 0
- #define SYS_LIGHTWEIGHT_PROT 1
- #define LWIP_ASSERT_CORE_LOCKED()
- #define MEM_LIBC_MALLOC 0
- #define MEM_CUSTOM_ALLOCATOR 0
- #define MEMP_MEM_MALLOC 0
- #define MEMP_MEM_INIT 0
- #define MEM_ALIGNMENT 1
- #define MEM_SIZE 1600
- #define MEMP_OVERFLOW_CHECK 0
- #define MEMP_SANITY_CHECK 0
- #define MEM_OVERFLOW_CHECK 0
- #define MEM_SANITY_CHECK 0
- #define MEM_USE_POOLS 0
- #define MEM_USE_POOLS_TRY_BIGGER_POOL 0
- #define MEMP_USE_CUSTOM_POOLS 0
- #define LWIP_ALLOW_MEM_FREE_FROM_OTHER_CONTEXT 0
- #define MEMP_NUM_PBUF 16
- #define MEMP_NUM_RAW_PCB 4
- #define MEMP_NUM_UDP_PCB 4
- #define MEMP_NUM_TCP_PCB 5
- #define MEMP_NUM_TCP_PCB_LISTEN 8
- #define MEMP_NUM_TCP_SEG 16
- #define MEMP_NUM_ALTCP_PCB MEMP_NUM_TCP_PCB
- #define MEMP_NUM_REASSDATA 5
- #define MEMP_NUM_FRAG_PBUF 15

- #define MEMP_NUM_ARP_QUEUE   30
- #define MEMP_NUM_IGMP_GROUP   8
- #define LWIP_NUM_SYS_TIMEOUT_INTERNAL   (LWIP_TCP + IP_REASSEMBLY + LWIP_ARP + (2*LWIP_DHCP) + LWIP_ACD + LWIP_IGMP + LWIP_DNS + PPP_NUM_TIMEOUTS + (LWIP_IPV6 * (1 + LWIP_IPV6_REASS + LWIP_IPV6_MLD + LWIP_IPV6_DHCP6)))
- #define MEMP_NUM_SYS_TIMEOUT LWIP_NUM_SYS_TIMEOUT_INTERNAL
- #define MEMP_NUM_NETBUF   2
- #define MEMP_NUM_NETCONN   4
- #define MEMP_NUM_SELECT_CB   4
- #define MEMP_NUM_TCPIP_MSG_API   8
- #define MEMP_NUM_TCPIP_MSG_INPKT   8
- #define MEMP_NUM_NETDB   1
- #define MEMP_NUM_LOCALHOSTLIST   1
- #define PBUF_POOL_SIZE   16
- #define MEMP_NUM_API_MSG MEMP_NUM_TCPIP_MSG_API
- #define MEMP_NUM_DNS_API_MSG MEMP_NUM_TCPIP_MSG_API
- #define MEMP_NUM_SOCKET_SETGETSOCKOPT_DATA MEMP_NUM_TCPIP_MSG_API
- #define MEMP_NUM_NETIFAPI_MSG MEMP_NUM_TCPIP_MSG_API
- #define LWIP_ARP   1
- #define ARP_TABLE_SIZE   10
- #define ARP_MAXAGE   300
- #define ARP_QUEUEING   0
- #define ARP_QUEUE_LEN   3
- #define ETHARP_SUPPORT_VLAN   0
- #define LWIP_ETHERNET LWIP_ARP
- #define ETH_PAD_SIZE   0
- #define ETHARP_SUPPORT_STATIC_ENTRIES   0
- #define ETHARP_TABLE_MATCH_NETIF   !LWIP_SINGLE_NETIF
- #define LWIP_IPV4   1
- #define IP_FORWARD   0
- #define IP_REASSEMBLY   1
- #define IP_FRAG   1
- #define IP_OPTIONS_ALLOWED   1
- #define IP_REASS_MAXAGE   15
- #define IP_REASS_MAX_PBUFS   10
- #define IP_DEFAULT_TTL   255

- #define IP_SOF_BROADCAST 0
- #define IP_SOF_BROADCAST_RECV 0
- #define IP_FORWARD_ALLOW_TX_ON_RX_NETIF 0
- #define LWIP_ICMP 1
- #define ICMP_TTL IP_DEFAULT_TTL
- #define LWIP_BROADCAST_PING 0
- #define LWIP_MULTICAST_PING 0
- #define LWIP_RAW 0
- #define RAW_TTL IP_DEFAULT_TTL
- #define LWIP_DHCP 0
- #define LWIP_DHCP_DOES_ACD_CHECK LWIP_DHCP
- #define LWIP_DHCP_BOOTP_FILE 0
- #define LWIP_DHCP_GET_NTP_SRV 0
- #define LWIP_DHCP_MAX_NTP_SERVERS 1
- #define LWIP_DHCP_MAX_DNS_SERVERS DNS_MAX_SERVERS
- #define LWIP_DHCP_DISCOVER_ADD_HOSTNAME 1
- #define LWIP_AUTOIP 0
- #define LWIP_DHCP_AUTOIP_COOP 0
- #define LWIP_DHCP_AUTOIP_COOP_TRIES 9
- #define LWIP_ACD (LWIP_AUTOIP || LWIP_DHCP_DOES_ACD_CHECK)
- #define LWIP_MIB2_CALLBACKS 0
- #define LWIP_MULTICAST_TX_OPTIONS ((LWIP_IGMP || LWIP_IPV6_MLD) && (LWIP_UDP || LWIP_RAW))
- #define LWIP_IGMP 0
- #define LWIP_DNS 0
- #define DNS_TABLE_SIZE 4
- #define DNS_MAX_NAME_LENGTH 256
- #define DNS_MAX_SERVERS 2
- #define DNS_MAX_RETRIES 4
- #define DNS_DOES_NAME_CHECK 1
- #define LWIP_DNS_SECURE (LWIP_DNS_SECURE_RAND_XID | LWIP_DNS_SECURE_NO_MULTIPLE_OUTSTANDING | LWIP_DNS_SECURE_RAND_SRC_PORT)
- #define DNS_LOCAL_HOSTLIST 0
- #define DNS_LOCAL_HOSTLIST_IS_DYNAMIC 0
- #define LWIP_DNS_SUPPORT_MDNS_QUERIES 0
- #define LWIP_UDP 1

- #define LWIP_UDPLITE 0
- #define UDP_TTL IP_DEFAULT_TTL
- #define LWIP_NETBUF_RECVINFO 0
- #define LWIP_TCP 1
- #define TCP_TTL IP_DEFAULT_TTL
- #define TCP_WND (4 * TCP_MSS)
- #define TCP_MAXRTX 12
- #define TCP_SYNMAXRTX 6
- #define TCP_QUEUE_OOSEQ LWIP_TCP
- #define LWIP_TCP_SACK_OUT 0
- #define LWIP_TCP_MAX_SACK_NUM 4
- #define TCP_MSS 536
- #define TCP_CALCULATE_EFF_SEND_MSS 1
- #define LWIP_TCP_RTO_TIME 3000
- #define TCP_SND_BUF (2 * TCP_MSS)
- #define TCP_SND_QUEUELEN ((4 * (TCP_SND_BUF) + (TCP_MSS - 1))/(TCP_MSS))
- #define TCP_SNDLOWAT LWIP_MIN(LWIP_MAX(((TCP_SND_BUF)/2), (2 * TCP_MSS) + 1), (TCP_SND_BUF) - 1)
- #define TCP_SNDQUEUELOWAT LWIP_MAX(((TCP_SND_QUEUELEN)/2), 5)
- #define TCP_OOSEQ_MAX_BYTES 0
- #define TCP_OOSEQ_MAX_PBUFS 0
- #define TCP_LISTEN_BACKLOG 0
- #define TCP_DEFAULT_LISTEN_BACKLOG 0xff
- #define TCP_OVERSIZE TCP_MSS
- #define LWIP_TCP_TIMESTAMPS 0
- #define TCP_WND_UPDATE_THRESHOLD LWIP_MIN((TCP_WND / 4), (TCP_MSS * 4))
- #define LWIP_EVENT_API 0
- #define LWIP_WND_SCALE 0
- #define LWIP_TCP_PCB_NUM_EXT_ARGS 0
- #define LWIP_ALTCP 0
- #define LWIP_ALTCP_TLS 0
- #define PBUF_LINK_HLEN (14 + ETH_PAD_SIZE)
- #define PBUF_LINK_ENCAPSULATION_HLEN 0
- #define PBUF_POOL_BUFSIZE LWIP_MEM_ALIGN_SIZE(TCP_MSS+PBUF_IP_HLEN+PBUF_TRANSPORT_HLEN+PBUF_
- #define LWIP_PBUF_REF_T u8_t
- #define LWIP_PBUF_CUSTOM_DATA

- #define LWIP_PBUF_CUSTOM_DATA_INIT(p)
- #define LWIP_SINGLE_NETIF 0
- #define LWIP_NETIF_HOSTNAME 0
- #define LWIP_NETIF_API 0
- #define LWIP_NETIF_STATUS_CALLBACK 0
- #define LWIP_NETIF_EXT_STATUS_CALLBACK 0
- #define LWIP_NETIF_LINK_CALLBACK 0
- #define LWIP_NETIF_REMOVE_CALLBACK 0
- #define LWIP_NETIF_HWADDRHINT 0
- #define LWIP_NETIF_TX_SINGLE_PBUF 0
- #define LWIP_NUM_NETIF_CLIENT_DATA 0
- #define LWIP_HAVE_LOOPIF (LWIP_NETIF_LOOPBACK && !LWIP_SINGLE_NETIF)
- #define LWIP_LOOPIF_MULTICAST 0
- #define LWIP_NETIF_LOOPBACK 0
- #define LWIP_LOOPBACK_MAX_PBUFS 0
- #define LWIP_NETIF_LOOPBACK_MULTITHREADING (!NO_SYS)
- #define TCPIP_THREAD_NAME "tcpip_thread"
- #define TCPIP_THREAD_STACKSIZE 0
- #define TCPIP_THREAD_PRIO 1
- #define TCPIP_MBOX_SIZE 0
- #define LWIP_TCPIP_THREAD_ALIVE()
- #define SLIPIF_THREAD_NAME "slipif_loop"
- #define SLIPIF_THREAD_STACKSIZE 0
- #define SLIPIF_THREAD_PRIO 1
- #define DEFAULT_THREAD_NAME "lwIP"
- #define DEFAULT_THREAD_STACKSIZE 0
- #define DEFAULT_THREAD_PRIO 1
- #define DEFAULT_RAW_RECVMBOX_SIZE 0
- #define DEFAULT_UDP_RECVMBOX_SIZE 0
- #define DEFAULT_TCP_RECVMBOX_SIZE 0
- #define DEFAULT_ACCEPTMBOX_SIZE 0
- #define LWIP_NETCONN 1
- #define LWIP_TCPIP_TIMEOUT 0
- #define LWIP_NETCONN_SEM_PER_THREAD 0
- #define LWIP_NETCONN_FULLDUPLEX 0

- #define LWIP_SOCKET 1
- #define LWIP_COMPAT_SOCKETS 1
- #define LWIP_POSIX_SOCKETS_IO_NAMES 1
- #define LWIP_SOCKET_OFFSET 0
- #define LWIP_SOCKET_EXTERNAL_HEADERS 0
- #define LWIP_TCP_KEEPALIVE 0
- #define LWIP_SO_SNDTIMEO 0
- #define LWIP_SO_RCVTIMEO 0
- #define LWIP_SO_SNDRCVTIMEO_NONSTANDARD 0
- #define LWIP_SO_RCVBUF 0
- #define LWIP_SO_LINGER 0
- #define RECV_BUFSIZE_DEFAULT INT_MAX
- #define LWIP_TCP_CLOSE_TIMEOUT_MS_DEFAULT 20000
- #define SO_REUSE 0
- #define SO_REUSE_RXTOALL 0
- #define LWIP_FIONREAD_LINUXMODE 0
- #define LWIP_SOCKET_SELECT 1
- #define LWIP_SOCKET_POLL 1
- #define LWIP_STATS 1
- #define LWIP_STATS_DISPLAY 0
- #define LINK_STATS 1
- #define ETHARP_STATS (LWIP_ARP)
- #define IP_STATS 1
- #define IPFRAG_STATS (IP_REASSEMBLY || IP_FRAG)
- #define ICMP_STATS 1
- #define IGMP_STATS (LWIP_IGMP)
- #define UDP_STATS (LWIP_UDP)
- #define TCP_STATS (LWIP_TCP)
- #define MEM_STATS ((MEM_CUSTOM_ALLOCATOR == 0) && (MEM_USE_POOLS == 0))
- #define MEMP_STATS (MEMP_MEM_MALLOC == 0)
- #define SYS_STATS (NO_SYS == 0)
- #define IP6_STATS (LWIP_IPV6)
- #define ICMP6_STATS (LWIP_IPV6 && LWIP_ICMP6)
- #define IP6_FRAG_STATS (LWIP_IPV6 && (LWIP_IPV6_FRAG || LWIP_IPV6_REASS))
- #define MLD6_STATS (LWIP_IPV6 && LWIP_IPV6_MLD)

- #define ND6_STATS (LWIP_IPV6)
- #define MIB2_STATS 0
- #define LWIP_CHECKSUM_CTRL_PER_NETIF 0
- #define CHECKSUM_GEN_IP 1
- #define CHECKSUM_GEN_UDP 1
- #define CHECKSUM_GEN_TCP 1
- #define CHECKSUM_GEN_ICMP 1
- #define CHECKSUM_GEN_ICMP6 1
- #define CHECKSUM_CHECK_IP 1
- #define CHECKSUM_CHECK_UDP 1
- #define CHECKSUM_CHECK_TCP 1
- #define CHECKSUM_CHECK_ICMP 1
- #define CHECKSUM_CHECK_ICMP6 1
- #define LWIP_CHECKSUM_ON_COPY 0
- #define LWIP_IPV6 0
- #define IPV6_REASS_MAXAGE 60
- #define LWIP_IPV6_SCOPES (LWIP_IPV6 && !LWIP_SINGLE_NETIF)
- #define LWIP_IPV6_SCOPES_DEBUG 0
- #define LWIP_IPV6_NUM_ADDRESSES 3
- #define LWIP_IPV6_FORWARD 0
- #define LWIP_IPV6_FRAG 1
- #define LWIP_IPV6_REASS LWIP_IPV6
- #define LWIP_IPV6_SEND_ROUTER_SOLICIT LWIP_IPV6
- #define LWIP_IPV6_AUTOCONFIG LWIP_IPV6
- #define LWIP_IPV6_ADDRESS_LIFETIMES LWIP_IPV6_AUTOCONFIG
- #define LWIP_IPV6_DUP_DETECT_ATTEMPTS 1
- #define LWIP_ICMP6 LWIP_IPV6
- #define LWIP_ICMP6_DATASIZE 0
- #define LWIP_ICMP6_HL 255
- #define LWIP_IPV6_MLD LWIP_IPV6
- #define MEMP_NUM_MLD6_GROUP 4
- #define LWIP_ND6_QUEUEING LWIP_IPV6
- #define MEMP_NUM_ND6_QUEUE 20
- #define LWIP_ND6_NUM_NEIGHBORS 10
- #define LWIP_ND6_NUM_DESTINATIONS 10

- #define LWIP_ND6_NUM_PREFIXES 5
- #define LWIP_ND6_NUM_ROUTERS 3
- #define LWIP_ND6_MAX_MULTICAST_SOLICIT 3
- #define LWIP_ND6_MAX_UNICAST_SOLICIT 3
- #define LWIP_ND6_MAX_ANYCAST_DELAY_TIME 1000
- #define LWIP_ND6_MAX_NEIGHBOR_ADVERTISEMENT 3
- #define LWIP_ND6_REACHABLE_TIME 30000
- #define LWIP_ND6_RETRANS_TIMER 1000
- #define LWIP_ND6_DELAY_FIRST_PROBE_TIME 5000
- #define LWIP_ND6_ALLOW_RA_UPDATES 1
- #define LWIP_ND6_TCP_REACHABILITY_HINTS 1
- #define LWIP_ND6_RDNSS_MAX_DNS_SERVERS 0
- #define LWIP_IPV6_DHCP6 0
- #define LWIP_IPV6_DHCP6_STATEFUL 0
- #define LWIP_IPV6_DHCP6_STATELESS LWIP_IPV6_DHCP6
- #define LWIP_DHCP6_GET_NTP_SRV 0
- #define LWIP_DHCP6_MAX_NTP_SERVERS 1
- #define LWIP_DHCP6_MAX_DNS_SERVERS DNS_MAX_SERVERS
- #define LWIP_HOOK_FILENAME "path/to/my/lwip_hooks.h"
- #define LWIP_HOOK_TCP_ISN(local_ip, local_port, remote_ip, remote_port)
- #define LWIP_HOOK_TCP_INPACKET_PCB(pcb, hdr, optlen, opt1len, opt2, p)
- #define LWIP_HOOK_TCP_OUT_TCPOPT_LENGTH(pcb, internal_len)
- #define LWIP_HOOK_TCP_OUT_ADD_TCPOPTS(p, hdr, pcb, opts)
- #define LWIP_HOOK_IP4_INPUT(pbuf, input_netif)
- #define LWIP_HOOK_IP4_ROUTE()
- #define LWIP_HOOK_IP4_ROUTE_SRC(src, dest)
- #define LWIP_HOOK_IP4_CANFORWARD(src, dest)
- #define LWIP_HOOK_ETHARP_GET_GW(netif, dest)
- #define LWIP_HOOK_IP6_INPUT(pbuf, input_netif)
- #define LWIP_HOOK_IP6_ROUTE(src, dest)
- #define LWIP_HOOK_ND6_GET_GW(netif, dest)
- #define LWIP_HOOK_VLAN_CHECK(netif, eth_hdr, vlan_hdr)
- #define LWIP_HOOK_VLAN_SET(netif, p, src, dst, eth_type)
- #define LWIP_HOOK_MEMP_AVAILABLE(memp_t_type)
- #define LWIP_HOOK_UNKNOWN_ETH_PROTOCOL(pbuf, netif)

- #define LWIP_HOOK_DHCP_APPEND_OPTIONS(netif, dhcp, state, msg, msg_type, options_len_ptr)
- #define LWIP_HOOK_DHCP_PARSE_OPTION(netif, dhcp, state, msg, msg_type, option, len, pbuf, offset)
- #define LWIP_HOOK_DHCP6_APPEND_OPTIONS(netif, dhcp6, state, msg, msg_type, options_len_ptr, max_len)
- #define LWIP_HOOK_SOCKETS_SETSOCKOPT(s, sock, level, optname, optval, optlen, err)
- #define LWIP_HOOK_SOCKETS_GETSOCKOPT(s, sock, level, optname, optval, optlen, err)
- #define LWIP_HOOK_NETCONN_EXTERNAL_RESOLVE(name, addr, addrtype, err)
- #define LWIP_DBG_MIN_LEVEL LWIP_DBG_LEVEL_ALL
- #define LWIP_DBG_TYPES_ON LWIP_DBG_ON
- #define ETHARP_DEBUG LWIP_DBG_OFF
- #define NETIF_DEBUG LWIP_DBG_OFF
- #define PBUF_DEBUG LWIP_DBG_OFF
- #define API_LIB_DEBUG LWIP_DBG_OFF
- #define API_MSG_DEBUG LWIP_DBG_OFF
- #define SOCKETS_DEBUG LWIP_DBG_OFF
- #define ICMP_DEBUG LWIP_DBG_OFF
- #define IGMP_DEBUG LWIP_DBG_OFF
- #define INET_DEBUG LWIP_DBG_OFF
- #define IP_DEBUG LWIP_DBG_OFF
- #define IP_REASS_DEBUG LWIP_DBG_OFF
- #define RAW_DEBUG LWIP_DBG_OFF
- #define MEM_DEBUG LWIP_DBG_OFF
- #define MEMP_DEBUG LWIP_DBG_OFF
- #define SYS_DEBUG LWIP_DBG_OFF
- #define TIMERS_DEBUG LWIP_DBG_OFF
- #define TCP_DEBUG LWIP_DBG_OFF
- #define TCP_INPUT_DEBUG LWIP_DBG_OFF
- #define TCP_FR_DEBUG LWIP_DBG_OFF
- #define TCP_RTO_DEBUG LWIP_DBG_OFF
- #define TCP_CWND_DEBUG LWIP_DBG_OFF
- #define TCP_WND_DEBUG LWIP_DBG_OFF
- #define TCP_OUTPUT_DEBUG LWIP_DBG_OFF
- #define TCP_RST_DEBUG LWIP_DBG_OFF
- #define TCP_QLEN_DEBUG LWIP_DBG_OFF
- #define UDP_DEBUG LWIP_DBG_OFF
- #define TCPIP_DEBUG LWIP_DBG_OFF

- #define SLIP_DEBUG LWIP_DBG_OFF
- #define DHCP_DEBUG LWIP_DBG_OFF
- #define AUTOIP_DEBUG LWIP_DBG_OFF
- #define ACD_DEBUG LWIP_DBG_OFF
- #define DNS_DEBUG LWIP_DBG_OFF
- #define IP6_DEBUG LWIP_DBG_OFF
- #define DHCP6_DEBUG LWIP_DBG_OFF
- #define LWIP_PERF 0

4.178.2 Detailed Description

lwIP Options Configuration

4.179 src/include/lwip/pbuf.h File Reference

```
#include "lwip/opt.h"#include "lwip/err.h"
```

4.179.1 Data Structures

- struct pbuf
- struct pbuf_rom
- struct pbuf_custom

4.179.2 Macros

- #define LWIP_SUPPORT_CUSTOM_PBUF ((IP_FRAG && !LWIP_NETIF_TX_SINGLE_PBUF) || (LWIP_IPV6 && LWIP_IPV6_FRAG))
- #define PBUF_NEEDS_COPY(p) ((p)->type_internal & PBUF_TYPE_FLAG_DATA_VOLATILE)
- #define PBUF_TYPE_FLAG_STRUCT_DATA_CONTIGUOUS 0x80
- #define PBUF_TYPE_FLAG_DATA_VOLATILE 0x40
- #define PBUF_TYPE_ALLOC_SRC_MASK 0x0F
- #define PBUF_ALLOC_FLAG_RX 0x0100
- #define PBUF_ALLOC_FLAG_DATA_CONTIGUOUS 0x0200
- #define PBUF_TYPE_ALLOC_SRC_MASK_APP_MIN 0x03
- #define PBUF_TYPE_ALLOC_SRC_MASK_APP_MAX PBUF_TYPE_ALLOC_SRC_MASK
- #define PBUF_FLAG_PUSH 0x01U
- #define PBUF_FLAG_IS_CUSTOM 0x02U
- #define PBUF_FLAG_MCASTLOOP 0x04U
- #define PBUF_FLAG_LLBCAST 0x08U

- #define PBUF_FLAG_LLMCAST 0x10U
- #define PBUF_FLAG_TCP_FIN 0x20U
- #define PBUF_POOL_FREE_OOSEQ 1

4.179.3 Typedefs

- typedef void(* pbuf_free_custom_fn) (struct pbuf *p)

4.179.4 Enumerations

- enum pbuf_layer { PBUF_TRANSPORT = 0 + (14 + 0) + 40 + 20 , PBUF_IP = 0 + (14 + 0) + 40 , PBUF_LINK = 0 + (14 + 0) , PBUF_RAW_TX = 0 , PBUF_RAW = 0 }
- enum pbuf_type { PBUF_RAM = (0x0200 | 0x80 | 0x00) , PBUF_ROM = 0x01 , PBUF_REF = (0x40 | 0x01) , PBUF_POOL = (0x0100 | 0x80 | 0x02) }

4.179.5 Functions

- struct pbuf * pbuf_alloc (pbuf_layer l, u16_t length, pbuf_type type)
- struct pbuf * pbuf_alloc_reference (void *payload, u16_t length, pbuf_type type)
- struct pbuf * pbuf_alloced_custom (pbuf_layer l, u16_t length, pbuf_type type, struct pbuf_custom *p, void *payload_mem, u16_t payload_mem_len)
- void pbuf_realloc (struct pbuf *p, u16_t size)
- u8_t pbuf_header (struct pbuf *p, s16_t header_size)
- u8_t pbuf_header_force (struct pbuf *p, s16_t header_size)
- u8_t pbuf_add_header (struct pbuf *p, size_t header_size_increment)
- u8_t pbuf_add_header_force (struct pbuf *p, size_t header_size_increment)
- u8_t pbuf_remove_header (struct pbuf *p, size_t header_size)
- struct pbuf * pbuf_free_header (struct pbuf *q, u16_t size)
- void pbuf_ref (struct pbuf *p)
- u8_t pbuf_free (struct pbuf *p)
- u16_t pbuf_clen (const struct pbuf *p)
- void pbuf_cat (struct pbuf *head, struct pbuf *tail)
- void pbuf_chain (struct pbuf *head, struct pbuf *tail)
- struct pbuf * pbuf_dechain (struct pbuf *p)
- err_t pbuf_copy (struct pbuf *p_to, const struct pbuf *p_from)
- err_t pbuf_copy_partial_pbuf (struct pbuf *p_to, const struct pbuf *p_from, u16_t copy_len, u16_t offset)
- u16_t pbuf_copy_partial (const struct pbuf *p, void *dataptr, u16_t len, u16_t offset)
- void * pbuf_get_contiguous (const struct pbuf *p, void *buffer, size_t bufsize, u16_t len, u16_t offset)
- err_t pbuf_take (struct pbuf *buf, const void *dataptr, u16_t len)

- err_t pbuf_take_at (struct pbuf *buf, const void *dataptr, u16_t len, u16_t offset)
- struct pbuf * pbuf_skip (struct pbuf *in, u16_t in_offset, u16_t *out_offset)
- struct pbuf * pbuf_coalesce (struct pbuf *p, pbuf_layer layer)
- struct pbuf * pbuf_clone (pbuf_layer l, pbuf_type type, struct pbuf *p)
- u8_t pbuf_get_at (const struct pbuf *p, u16_t offset)
- int pbuf_try_get_at (const struct pbuf *p, u16_t offset)
- void pbuf_put_at (struct pbuf *p, u16_t offset, u8_t data)
- u16_t pbuf_memcmp (const struct pbuf *p, u16_t offset, const void *s2, u16_t n)
- u16_t pbuf_memfind (const struct pbuf *p, const void *mem, u16_t mem_len, u16_t start_offset)
- u16_t pbuf_strstr (const struct pbuf *p, const char *substr)

4.179.6 Detailed Description

pbuf API

4.179.7 Macro Definition Documentation

4.179.7.1 LWIP_SUPPORT_CUSTOM_PBUF

```
#define LWIP_SUPPORT_CUSTOM_PBUF   ((IP_FRAG && !LWIP_NETIF_TX_SINGLE_PBUF) || (LWIP_IPV6
&& LWIP_IPV6_FRAG))
```

LWIP_SUPPORT_CUSTOM_PBUF==1: Custom pbufs behave much like their pbuf type but they are allocated by external code (initialised by calling pbuf_alloced_custom()) and when pbuf_free gives up their last reference, they are freed by calling pbuf_custom->custom_free_function(). Currently, the pbuf_custom code is only needed for one specific configuration of IP_FRAG, unless required by external driver/application code.

4.179.7.2 PBUF_ALLOC_FLAG_DATA_CONTIGUOUS

```
#define PBUF_ALLOC_FLAG_DATA_CONTIGUOUS   0x0200
```

Indicates the application needs the pbuf payload to be in one piece

4.179.7.3 PBUF_ALLOC_FLAG_RX

```
#define PBUF_ALLOC_FLAG_RX   0x0100
```

Indicates this pbuf is used for RX (if not set, indicates use for TX). This information can be used to keep some spare RX buffers e.g. for receiving TCP ACKs to unblock a connection)

4.179.7.4 PBUF_FLAG_IS_CUSTOM

```
#define PBUF_FLAG_IS_CUSTOM   0x02U
```

indicates this is a custom pbuf: pbuf_free calls pbuf_custom->custom_free_function() when the last reference is released (plus custom PBUF_RAM cannot be trimmed)

4.179.7.5 PBUF_FLAG_LLBCAST

```
#define PBUF_FLAG_LLBCAST   0x08U
```

indicates this pbuf was received as link-level broadcast

4.179.7.6 PBUF_FLAG_LLMCAST

```
#define PBUF_FLAG_LLMCAST   0x10U
```

indicates this pbuf was received as link-level multicast

4.179.7.7 PBUF_FLAG_MCASTLOOP

```
#define PBUF_FLAG_MCASTLOOP   0x04U
```

indicates this pbuf is UDP multicast to be looped back

4.179.7.8 PBUF_FLAG_PUSH

```
#define PBUF_FLAG_PUSH   0x01U
```

indicates this packet's data should be immediately passed to the application

4.179.7.9 PBUF_FLAG_TCP_FIN

```
#define PBUF_FLAG_TCP_FIN   0x20U
```

indicates this pbuf includes a TCP FIN flag

4.179.7.10 PBUF_POOL_FREE_OOSEQ

```
#define PBUF_POOL_FREE_OOSEQ   1
```

Define this to 0 to prevent freeing ooseq pbufs when the PBUF_POOL is empty

4.179.7.11 PBUF_TYPE_ALLOC_SRC_MASK

```
#define PBUF_TYPE_ALLOC_SRC_MASK   0x0F
```

4 bits are reserved for 16 allocation sources (e.g. heap, pool1, pool2, etc) Internally, we use: 0=heap, 1=MEMP_PBUF, 2=MEMP_PBUF_POOL -> 13 types free

4.179.7.12 PBUF_TYPE_ALLOC_SRC_MASK_APP_MAX

```
#define PBUF_TYPE_ALLOC_SRC_MASK_APP_MAX   PBUF_TYPE_ALLOC_SRC_MASK
```

Last pbuf allocation type for applications

4.179.7.13 PBUF_TYPE_ALLOC_SRC_MASK_APP_MIN

```
#define PBUF_TYPE_ALLOC_SRC_MASK_APP_MIN   0x03
```

First pbuf allocation type for applications

4.179.7.14 PBUF_TYPE_FLAG_DATA_VOLATILE

```
#define PBUF_TYPE_FLAG_DATA_VOLATILE   0x40
```

Indicates the data stored in this pbuf can change. If this pbuf needs to be queued, it must be copied/duplicated.

4.179.7.15 PBUF_TYPE_FLAG_STRUCT_DATA_CONTIGUOUS

```
#define PBUF_TYPE_FLAG_STRUCT_DATA_CONTIGUOUS   0x80
```

Indicates that the payload directly follows the struct pbuf. This makes pbuf_header work in both directions.

4.179.8 Typedef Documentation

4.179.8.1 pbuf_free_custom_fn

```
typedef void(* pbuf_free_custom_fn) (struct pbuf *p)
```

Prototype for a function to free a custom pbuf

4.179.9 Function Documentation

4.179.9.1 pbuf_add_header()

```
u8_t pbuf_add_header (struct pbuf * p, size_t header_size_increment)
```

Adjusts the payload pointer to reveal headers in the payload.

Adjusts the ->payload pointer so that space for a header appears in the pbuf payload.

The ->payload, ->tot_len and ->len fields are adjusted.

Parameters

p	pbuf to change the header size.
header_size_increment	Number of bytes to increment header size which increases the size of the pbuf. New space is on the front. If header_size_increment is 0, this function does nothing and returns successful.

PBUF_ROM and PBUF_REF type buffers cannot have their sizes increased, so the call will fail. A check is made that the increase in header size does not move the payload pointer in front of the start of the buffer.

Returns non-zero on failure, zero on success.

4.179.9.2 pbuf_add_header_force()

```
u8_t pbuf_add_header_force (struct pbuf * p, size_t header_size_increment)
```

Same as pbuf_add_header but does not check if 'header_size > 0' is allowed. This is used internally only, to allow PBUF_REF for RX.

4.179.9.3 pbuf_clen()

```
u16_t pbuf_clen (const struct pbuf * p)
```

Count number of pbufs in a chain

Parameters

Returns the number of pbufs in a chain

p	first pbuf of chain

4.179.9.4 pbuf_dechain()

```
struct pbuf * pbuf_dechain (struct pbuf * p)
```

Dechains the first pbuf from its succeeding pbufs in the chain.

Makes p->tot_len field equal to p->len.

Parameters

p	pbuf to dechain

Returns remainder of the pbuf chain, or NULL if it was de-allocated.

Note
May not be called on a packet queue.

4.179.9.5 pbuf_free_header()

```
struct pbuf * pbuf_free_header (struct pbuf * q, u16_t size)
```

Similar to pbuf_header(-size) but de-refs header pbufs for (size >= p->len)

Parameters

q	pbufs to operate on
size	The number of bytes to remove from the beginning of the pbuf list. While size >= p->len, pbufs are freed. ATTENTION: this is the opposite direction as pbuf_header, but takes an u16_t not s16_t!

Returns the new head pbuf

4.179.9.6 pbuf_header()

```
u8_t pbuf_header (struct pbuf * p, s16_t header_size_increment)
```

Adjusts the payload pointer to hide or reveal headers in the payload.

Adjusts the ->payload pointer so that space for a header (dis)appears in the pbuf payload.

The ->payload, ->tot_len and ->len fields are adjusted.

Parameters

PBUF_ROM and PBUF_REF type buffers cannot have their sizes increased, so the call will fail. A check is made that the increase in header size does not move the payload pointer in front of the start of the buffer. **Returns** non-zero on failure, zero on success.

4.179.9.7 pbuf_header_force()

```
u8_t pbuf_header_force (struct pbuf * p, s16_t header_size_increment)
```

Same as pbuf_header but does not check if 'header_size > 0' is allowed. This is used internally only, to allow PBUF_REF for RX.

p	pbuf to change the header size.
header_size_increment	Number of bytes to increment header size which increases the size of the pbuf. New space is on the front. (Using a negative value decreases the header size.) If header_size_increment is 0, this function does nothing and returns successful.

4.179.9.8 pbuf_remove_header()

`u8_t pbuf_remove_header (struct pbuf * p, size_t header_size_decrement)`

Adjusts the payload pointer to hide headers in the payload.

Adjusts the ->payload pointer so that space for a header disappears in the pbuf payload.

The ->payload, ->tot_len and ->len fields are adjusted.

Parameters

p	pbuf to change the header size.
header_size_decrement	Number of bytes to decrement header size which decreases the size of the pbuf. If header_size_decrement is 0, this function does nothing and returns successful.

Returns non-zero on failure, zero on success.

4.179.9.9 pbuf_strstr()

`u16_t pbuf_strstr (const struct pbuf * p, const char * substr)`

Find occurrence of substr with length substr_len in pbuf p, start at offset start_offset WARNING: in contrast to strstr(), this one does not stop at the first \0 in the pbuf/source string!

Parameters

p	pbuf to search, maximum length is 0xFFFE since 0xFFFF is used as return value ’not found’
substr	string to search for in p, maximum length is 0xFFFE

Returns 0xFFFF if substr was not found in p or the index where it was found

4.180 src/include/lwip/priv/altcp_priv.h File Reference

```
#include "lwip/opt.h"#include "lwip/altcp.h"#include "lwip/ip_addr.h"
```

4.180.1 Functions

- struct altcp_pcb * altcp_alloc (void)
- void altcp_free (struct altcp_pcb *conn)

4.180.2 Detailed Description

Application layered TCP connection API (to be used from TCPIP thread) This interface mimics the tcp callback API to the application while preventing direct linking (much like virtual functions). This way, an application can make use of other application layer protocols on top of TCP without knowing the details (e.g. TLS, proxy connection).

4.180.3 Function Documentation

4.180.3.1 altcp_alloc()

```
struct altcp_pcb * altcp_alloc (void )
```

For altcp layer implementations only: allocate a new struct altcp_pcb from the pool and zero the memory

4.180.3.2 altcp_free()

```
void altcp_free (struct altcp_pcb * conn)
```

For altcp layer implementations only: return a struct altcp_pcb to the pool

4.181 src/include/lwip/priv/api_msg.h File Reference

```
#include "lwip/opt.h"#include "lwip/arch.h"#include "lwip/ip_addr.h"#include "lwip/err.h"# ←
    include "lwip/sys.h"#include "lwip/igmp.h"#include "lwip/api.h"#include "lwip/priv/ ←
    tcpip_priv.h"
```

4.181.1 Data Structures

- struct api_msg
- struct dns_api_msg

4.181.2 Functions

- void lwip_netconn_do_newconn (void *m)
- void lwip_netconn_do_delconn (void *m)
- void lwip_netconn_do_bind (void *m)
- void lwip_netconn_do_bind_if (void *m)
- void lwip_netconn_do_connect (void *m)
- void lwip_netconn_do_disconnect (void *m)
- void lwip_netconn_do_listen (void *m)
- void lwip_netconn_do_send (void *m)
- void lwip_netconn_do_recv (void *m)
- void lwip_netconn_do_accepted (void *m)
- void lwip_netconn_do_write (void *m)
- void lwip_netconn_do_getaddr (void *m)
- void lwip_netconn_do_close (void *m)
- void lwip_netconn_do_join_leave_group (void *m)
- void lwip_netconn_do_join_leave_group_netif (void *m)
- void lwip_netconn_do_gethostbyname (void *arg)
- struct netconn * netconn_alloc (enum netconn_type t, netconn_callback callback)
- void netconn_free (struct netconn *conn)

4.181.3 Detailed Description

netconn API lwIP internal implementations (do not use in application code)

4.181.4 Function Documentation

4.181.4.1 lwip_netconn_do_accepted()

```
void lwip_netconn_do_accepted (void * m)
```

Indicate that a TCP pcb has been accepted Called from netconn_accept

Parameters

m	the api_msg pointing to the connection

4.181.4.2 lwip_netconn_do_bind()

```
void lwip_netconn_do_bind (void * m)
```

Bind a pcb contained in a netconn Called from netconn_bind.

Parameters

m	the api_msg pointing to the connection and containing the IP address and port to bind to

4.181.4.3 lwip_netconn_do_bind_if()

```
void lwip_netconn_do_bind_if (void * m)
```

Bind a pcb contained in a netconn to an interface Called from netconn_bind_if.

Parameters

m	the api_msg pointing to the connection and containing the IP address and port to bind to

4.181.4.4 lwip_netconn_do_close()

```
void lwip_netconn_do_close (void * m)
```

Close or half-shutdown a TCP pcb contained in a netconn Called from netconn_close In contrast to closing sockets, the netconn is not deallocated.

Parameters

4.181.4.5 lwip_netconn_do_connect()

```
void lwip_netconn_do_connect (void * m)
```

Connect a pcb contained inside a netconn Called from netconn_connect.

Parameters

m	the api_msg pointing to the connection

m	the api_msg pointing to the connection and containing the IP address and port to connect to

4.181.4.6 lwip_netconn_do_delconn()

```
void lwip_netconn_do_delconn (void * m)
```

Delete the pcb inside a netconn. Called from netconn_delete.

Parameters

m	the api_msg pointing to the connection

4.181.4.7 lwip_netconn_do_disconnect()

```
void lwip_netconn_do_disconnect (void * m)
```

Disconnect a pcb contained inside a netconn Only used for UDP netconns. Called from netconn_disconnect.

Parameters

4.181.4.8 lwip_netconn_do_getaddr()

```
void lwip_netconn_do_getaddr (void * m)
```

Return a connection's local or remote address Called from netconn_getaddr

Parameters

4.181.4.9 lwip_netconn_do_gethostbyname()

```
void lwip_netconn_do_gethostbyname (void * arg)
```

Execute a DNS query Called from netconn_gethostbyname

Parameters

4.181.4.10 lwip_netconn_do_join_leave_group()

```
void lwip_netconn_do_join_leave_group (void * m)
```

Join multicast groups for UDP netconns. Called from netconn_join_leave_group

Parameters

4.181.4.11 lwip_netconn_do_join_leave_group_netif()

```
void lwip_netconn_do_join_leave_group_netif (void * m)
```

Join multicast groups for UDP netconns. Called from netconn_join_leave_group_netif

Parameters

m	the api_msg pointing to the connection to disconnect

m	the api_msg pointing to the connection

4.181.4.12 lwip_netconn_do_listen()

```
void lwip_netconn_do_listen (void * m)
```

Set a TCP pcb contained in a netconn into listen mode Called from netconn_listen.

Parameters

4.181.4.13 lwip_netconn_do_newconn()

```
void lwip_netconn_do_newconn (void * m)
```

Create a new pcb of a specific type inside a netconn. Called from netconn_new_with_proto_and_callback.

Parameters

4.181.4.14 lwip_netconn_do_recv()

```
void lwip_netconn_do_recv (void * m)
```

Indicate data has been received from a TCP pcb contained in a netconn Called from netconn_recv

Parameters

4.181.4.15 lwip_netconn_do_send()

```
void lwip_netconn_do_send (void * m)
```

Send some data on a RAW or UDP pcb contained in a netconn Called from netconn_send

Parameters

4.181.4.16 lwip_netconn_do_write()

```
void lwip_netconn_do_write (void * m)
```

Send some data on a TCP pcb contained in a netconn Called from netconn_write

Parameters

4.181.4.17 netconn_alloc()

```
struct netconn * netconn_alloc (enum netconn_type t, netconn_callback callback)
```

Create a new netconn (of a specific type) that has a callback function. The corresponding pcb is NOT created!

Parameters

See also enum netconn_type)

Parameters

Returns a newly allocated struct netconn or NULL on memory error

arg	the dns_api_msg pointing to the query

m	the api_msg pointing to the connection

m	the api_msg pointing to the connection

4.181.4.18 netconn_free()

```
void netconn_free (struct netconn * conn)
```

Delete a netconn and all its resources. The pcb is NOT freed (since we might not be in the right thread context do this).

Parameters

4.182 src/include/lwip/priv/mem_priv.h File Reference

```
#include "lwip/opt.h"#include "lwip/mem.h"
```

4.182.1 Detailed Description

lwIP internal memory implementations (do not use in application code)

4.183 src/include/lwip/priv/memp_priv.h File Reference

```
#include "lwip/opt.h"#include "lwip/mem.h"#include "lwip/priv/mem_priv.h"
```

4.183.1 Data Structures

- struct memp_desc

4.183.2 Functions

- void memp_init_pool (const struct memp_desc *desc)
- void * memp_malloc_pool (const struct memp_desc *desc)
- void memp_free_pool (const struct memp_desc *desc, void *mem)

4.183.3 Detailed Description

memory pools lwIP internal implementations (do not use in application code)

m	the api_msg pointing to the connection

m	the api_msg describing the connection type

m	the api_msg pointing to the connection

4.183.4 Function Documentation

4.183.4.1 memp_free_pool()

`void memp_free_pool (const struct memp_desc * desc, void * mem)`

Put a custom pool element back into its pool.

Parameters

4.183.4.2 memp_init_pool()

`void memp_init_pool (const struct memp_desc * desc)`

Initialize custom memory pool. Related functions: memp_malloc_pool, memp_free_pool

Parameters

4.183.4.3 memp_malloc_pool()

`void * memp_malloc_pool (const struct memp_desc * desc)`

Get an element from a custom pool.

Parameters

Returns a pointer to the allocated memory or a NULL pointer on error

4.184 src/include/lwip/priv/memp_std.h File Reference

4.184.1 Detailed Description

lwIP internal memory pools (do not use in application code) This file is deliberately included multiple times: once with empty definition of LWIP_MEMPOOL() to handle all includes and multiple times to build up various lists of mem pools.

4.185 src/include/lwip/priv/nd6_priv.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/ip6_addr.h"#include "lwip/netif.h ↩
    "
```

4.185.1 Data Structures

- struct nd6_q_entry
- struct nd6_neighbor_cache_entry

m	the api_msg pointing to the connection

m	the api_msg pointing to the connection

t	the type of 'connection' to create (

4.185.2 Detailed Description

Neighbor discovery and stateless address autoconfiguration for IPv6. Aims to be compliant with RFC 4861 (Neighbor discovery) and RFC 4862 (Address autoconfiguration).

4.186 src/include/lwip/priv/raw_priv.h File Reference

```
#include "lwip/opt.h"#include "lwip/raw.h"
```

4.186.1 Typedefs

- typedef enum raw_input_state raw_input_state_t

4.186.2 Enumerations

- enum raw_input_state

4.186.3 Functions

- raw_input_state_t raw_input (struct pbuf *p, struct netif *inp)
- void raw_netif_ip_addr_changed (const ip_addr_t *old_addr, const ip_addr_t *new_addr)

4.186.4 Detailed Description

raw API internal implementations (do not use in application code)

4.186.5 Typedef Documentation

4.186.5.1 raw_input_state_t

```
typedef enum raw_input_state raw_input_state_t
```

return codes for raw_input

4.186.6 Enumeration Type Documentation

4.186.6.1 raw_input_state

```
enum raw_input_state
```

return codes for raw_input

callback	a function to call on status changes (RX available, TX'ed)

conn	the netconn to free

desc	the pool where to put mem
mem	the memp element to free

4.186.7 Function Documentation

4.186.7.1 raw_input()

```
raw_input_state_t raw_input (struct pbuf * p, struct netif * inp)
```

Determine if in incoming IP packet is covered by a RAW PCB and if so, pass it to a user-provided receive callback function.

Given an incoming IP datagram (as a chain of pbufs) this function finds a corresponding RAW PCB and calls the corresponding receive callback function.

Parameters

Returns - 1 if the packet has been eaten by a RAW PCB receive callback function. The caller MAY NOT not reference the packet any longer, and MAY NOT call pbuf_free().

- 0 if packet is not eaten (pbuf is still referenced by the caller).

4.186.7.2 raw_netif_ip_addr_changed()

```
void raw_netif_ip_addr_changed (const ip_addr_t * old_addr, const ip_addr_t * new_addr)
```

This function is called from netif.c when address is changed

Parameters

4.187 src/include/lwip/priv/sockets_priv.h File Reference

```
#include "lwip/opt.h"#include "lwip/err.h"#include "lwip/sockets.h"#include "lwip/sys.h"
```

4.187.1 Data Structures

- struct lwip_sock
- struct lwip_select_cb

4.187.2 Macros

- #define SELWAIT_T u8_t

4.187.3 Detailed Description

Sockets API internal implementations (do not use in application code)

desc	pool to initialize

desc	the pool to get an element from

p	pbuf to be demultiplexed to a RAW PCB.
inp	network interface on which the datagram was received.

4.187.4 Macro Definition Documentation

4.187.4.1 SELWAIT_T

```
#define SELWAIT_T   u8_t
```

This is overridable for the rare case where more than 255 threads select on the same socket...

4.188 src/include/lwip/priv/tcp_priv.h File Reference

```
#include "lwip/opt.h"#include "lwip/tcp.h"#include "lwip/mem.h"#include "lwip/pbuf.h"# ←
    include "lwip/ip.h"#include "lwip/icmp.h"#include "lwip/err.h"#include "lwip/ip6.h"# ←
    include "lwip/ip6_addr.h"#include "lwip/prot/tcp.h"
```

4.188.1 Macros

- #define tcp_do_output_nagle(tpcb)
- #define TF_RESET   (u8_t)0x08U /* Connection was reset. */
- #define TCP_OVERSIZE_DBGCHECK   0
- #define TCP_CHECKSUM_ON_COPY   (LWIP_CHECKSUM_ON_COPY && CHECKSUM_GEN_TCP)
- #define TCP_BUILD_MSS_OPTION(mss)   lwip_htonl(0x02040000 | ((mss) & 0xFFFF))

4.188.2 Functions

- void tcp_init (void)
- void tcp_tmr (void)
- void tcp_slowtmr (void)
- void tcp_fasttmr (void)
- void tcp_txnow (void)
- void tcp_input (struct pbuf *p, struct netif *inp)
- struct tcp_pcb * tcp_alloc (u8_t prio)
- void tcp_free (struct tcp_pcb *pcb)
- void tcp_abandon (struct tcp_pcb *pcb, int reset)

old_addr	IP address of the netif before change
new_addr	IP address of the netif after change

- err_t tcp_send_empty_ack (struct tcp_pcb *pcb)
- err_t tcp_rexmit (struct tcp_pcb *pcb)
- err_t tcp_rexmit_rto_prepare (struct tcp_pcb *pcb)
- void tcp_rexmit_rto_commit (struct tcp_pcb *pcb)
- void tcp_rexmit_rto (struct tcp_pcb *pcb)
- void tcp_rexmit_fast (struct tcp_pcb *pcb)
- u32_t tcp_update_rcv_ann_wnd (struct tcp_pcb *pcb)
- err_t tcp_process_refused_data (struct tcp_pcb *pcb)
- void tcp_pcb_purge (struct tcp_pcb *pcb)
- void tcp_pcb_remove (struct tcp_pcb **pcblist, struct tcp_pcb *pcb)
- void tcp_segs_free (struct tcp_seg *seg)
- void tcp_seg_free (struct tcp_seg *seg)
- struct tcp_seg * tcp_seg_copy (struct tcp_seg *seg)
- err_t tcp_send_fin (struct tcp_pcb *pcb)
- err_t tcp_enqueue_flags (struct tcp_pcb *pcb, u8_t flags)
- void tcp_rst (const struct tcp_pcb *pcb, u32_t seqno, u32_t ackno, const ip_addr_t *local_ip, const ip_addr_t *remote_ip, u16_t local_port, u16_t remote_port)
- void tcp_rst_netif (struct netif *netif, u32_t seqno, u32_t ackno, const ip_addr_t *local_ip, const ip_addr_t *remote_ip, u16_t local_port, u16_t remote_port)
- u32_t tcp_next_iss (struct tcp_pcb *pcb)
- err_t tcp_keepalive (struct tcp_pcb *pcb)
- err_t tcp_split_unsent_seg (struct tcp_pcb *pcb, u16_t split)
- err_t tcp_zero_window_probe (struct tcp_pcb *pcb)
- u16_t tcp_eff_send_mss_netif (u16_t sendmss, struct netif *outif, const ip_addr_t *dest)
- err_t tcp_recv_null (void *arg, struct tcp_pcb *pcb, struct pbuf *p, err_t err)
- void tcp_timer_needed (void)
- void tcp_netif_ip_addr_changed (const ip_addr_t *old_addr, const ip_addr_t *new_addr)
- err_t tcp_ext_arg_invoke_callbacks_passive_open (struct tcp_pcb_listen *lpcb, struct tcp_pcb *cpcb)

4.188.3 Variables

- struct tcp_pcb * tcp_bound_pcbs
- union tcp_listen_pcbs_t tcp_listen_pcbs
- struct tcp_pcb * tcp_active_pcbs
- struct tcp_pcb * tcp_tw_pcbs
- struct tcp_pcb **const tcp_pcb_lists [4]

4.188.4 Detailed Description

TCP internal implementations (do not use in application code)

4.188.5 Macro Definition Documentation

4.188.5.1 TCP_BUILD_MSS_OPTION

```
#define TCP_BUILD_MSS_OPTION( mss)   lwip_htonl(0x02040000 | ((mss) & 0xFFFF))
```

This returns a TCP header option for MSS in an u32_t

4.188.5.2 TCP_CHECKSUM_ON_COPY

```
#define TCP_CHECKSUM_ON_COPY   (LWIP_CHECKSUM_ON_COPY && CHECKSUM_GEN_TCP)
```

Don't generate checksum on copy if CHECKSUM_GEN_TCP is disabled

4.188.5.3 tcp_do_output_nagle

`#define tcp_do_output_nagle( tpcb)` **Value:**

```
(((tpcb)->unacked == NULL) || \
((tpcb)->flags & (TF_NODELAY | TF_INFR)) || \
(((tpcb)->unsent != NULL) && (((tpcb)->unsent->next != NULL) || ←
     \
  ((tpcb)->unsent->len >= (tpcb)->mss))) || \
((tcp_sndbuf(tpcb) == 0) || (tcp_sndqueuelen(tpcb) >=  ←
    TCP_SND_QUEUELEN)) \
) ? 1 : 0)
```

This is the Nagle algorithm: try to combine user data to send as few TCP segments as possible. Only send if

- no previously transmitted data on the connection remains unacknowledged or
- the TF_NODELAY flag is set (nagle algorithm turned off for this pcb) or
- the only unsent segment is at least pcb->mss bytes long (or there is more than one unsent segment - with lwIP, this can happen although unsent->len < mss)
- or if we are in fast-retransmit (TF_INFR)

4.188.5.4 TCP_OVERSIZE_DBGCHECK

```
#define TCP_OVERSIZE_DBGCHECK   0
```

Enabled extra-check for TCP_OVERSIZE if LWIP_DEBUG is enabled

4.188.5.5 TF_RESET

```
#define TF_RESET   (u8_t)0x08U /* Connection was reset. */
```

Flags used on input processing, not on pcb->flags

4.188.6 Function Documentation

4.188.6.1 tcp_abandon()

```
void tcp_abandon (struct tcp_pcb * pcb, int reset)
```

Abandons a connection and optionally sends a RST to the remote host. Deletes the local protocol control block. This is done when a connection is killed because of shortage of memory.

Parameters

pcb	the tcp_pcb to abort
reset	boolean to indicate whether a reset should be sent

4.188.6.2 tcp_alloc()

```
struct tcp_pcb * tcp_alloc (u8_t prio)
```

Allocate a new tcp_pcb structure.

Parameters

prio	priority for the new pcb

Returns a new tcp_pcb that initially is in state CLOSED

4.188.6.3 tcp_eff_send_mss_netif()

```
u16_t tcp_eff_send_mss_netif (u16_t sendmss, struct netif * outif, const ip_addr_t * dest)
```

Calculates the effective send mss that can be used for a specific IP address by calculating the minimum of TCP_MSS and the mtu (if set) of the target netif (if not NULL).

4.188.6.4 tcp_enqueue_flags()

```
err_t tcp_enqueue_flags (struct tcp_pcb * pcb, u8_t flags)
```

Enqueue SYN or FIN for transmission.

Called by tcp_connect, tcp_listen_input, and tcp_close (via tcp_send_fin)

Parameters

pcb	Protocol control block for the TCP connection.
flags	TCP header flags to set in the outgoing segment.

4.188.6.5 tcp_ext_arg_invoke_callbacks_passive_open()

```
err_t tcp_ext_arg_invoke_callbacks_passive_open (struct tcp_pcb_listen * lpcb, struct tcp_
* cpcb)
```

This function calls the "passive_open" callback for all ext_args if a connection is in the process of being accepted. This is called just after the SYN is received and before a SYN/ACK is sent, to allow to modify the very first segment sent even on passive open. Naturally, the "accepted" callback of the pcb has not been called yet!

4.188.6.6 tcp_fasttmr()

```
void tcp_fasttmr (void )
```

Is called every TCP_FAST_INTERVAL (250 ms) and process data previously "refused" by upper layer (application) and sends delayed ACKs or pending FINs.

Automatically called from tcp_tmr().

4.188.6.7 tcp_free()

```
void tcp_free (struct tcp_pcb * pcb)
```

Free a tcp pcb

4.188.6.8 tcp_init()

```
void tcp_init (void )
```

Initialize this module.

4.188.6.9 tcp_input()

```
void tcp_input (struct pbuf * p, struct netif * inp)
```

The initial input processing of TCP. It verifies the TCP header, demultiplexes the segment between the PCBs and passes it on to tcp_process(), which implements the TCP finite state machine. This function is called by the IP layer (in ip_input()).

Parameters

p	received TCP segment to process (p->payload pointing to the TCP header)
inp	network interface on which this segment was received

4.188.6.10 tcp_keepalive()

```
err_t tcp_keepalive (struct tcp_pcb * pcb)
```

Send keepalive packets to keep a connection active although no data is sent over it.

Called by tcp_slowtmr()

Parameters

pcb	the tcp_pcb for which to send a keepalive packet

4.188.6.11 tcp_netif_ip_addr_changed()

```
void tcp_netif_ip_addr_changed (const ip_addr_t * old_addr, const ip_addr_t * new_addr)
```

This function is called from netif.c when address is changed or netif is removed

Parameters

old_addr	IP address of the netif before change
new_addr	IP address of the netif after change or NULL if netif has been removed

4.188.6.12 tcp_next_iss()

```
u32_t tcp_next_iss (struct tcp_pcb * pcb)
```

Calculates a new initial sequence number for new connections.

Returns u32_t pseudo random sequence number

4.188.6.13 tcp_pcb_purge()

```
void tcp_pcb_purge (struct tcp_pcb * pcb)
```

Purges a TCP PCB. Removes any buffered data and frees the buffer memory (pcb->ooseq, pcb->unsent and pcb->unacked are freed).

Parameters

pcb	tcp_pcb to purge. The pcb itself is not deallocated!

4.188.6.14 tcp_pcb_remove()

```
void tcp_pcb_remove (struct tcp_pcb ** pcblist, struct tcp_pcb * pcb)
```

Purges the PCB and removes it from a PCB list. Any delayed ACKs are sent first.

Parameters

pcblist	PCB list to purge.
pcb	tcp_pcb to purge. The pcb itself is NOT deallocated!

4.188.6.15 tcp_process_refused_data()

```
err_t tcp_process_refused_data (struct tcp_pcb * pcb)
```

Pass pcb->refused_data to the recv callback

4.188.6.16 tcp_recv_null()

```
err_t tcp_recv_null (void * arg, struct tcp_pcb * pcb, struct pbuf * p, err_t err)
```

Default receive callback that is called if the user didn't register a recv callback for the pcb.

4.188.6.17 tcp_rexmit()

```
err_t tcp_rexmit (struct tcp_pcb * pcb)
```

Requeue the first unacked segment for retransmission

Called by tcp_receive() for fast retransmit.

Parameters

pcb	the tcp_pcb for which to retransmit the first unacked segment

4.188.6.18 tcp_rexmit_fast()

```
void tcp_rexmit_fast (struct tcp_pcb * pcb)
```

Handle retransmission after three dupacks received

Parameters

pcb	the tcp_pcb for which to retransmit the first unacked segment

4.188.6.19 tcp_rexmit_rto()

```
void tcp_rexmit_rto (struct tcp_pcb * pcb)
```

Requeue all unacked segments for retransmission

Called by tcp_process() only, tcp_slowtmr() needs to do some things between "prepare" and "commit".

Parameters

pcb	the tcp_pcb for which to re-enqueue all unacked segments

4.188.6.20 tcp_rexmit_rto_commit()

```
void tcp_rexmit_rto_commit (struct tcp_pcb * pcb)
```

Requeue all unacked segments for retransmission

Called by tcp_slowtmr() for slow retransmission.

Parameters

4.188.6.21 tcp_rexmit_rto_prepare()

```
err_t tcp_rexmit_rto_prepare (struct tcp_pcb * pcb)
```

Requeue all unacked segments for retransmission

Called by tcp_slowtmr() for slow retransmission.

Parameters

4.188.6.22 tcp_rst()

```
void tcp_rst (const struct tcp_pcb * pcb, u32_t seqno, u32_t ackno, const ip_addr_t * loca
const ip_addr_t * remote_ip, u16_t local_port, u16_t remote_port)
```

Send a TCP RESET packet (empty segment with RST flag set) to abort a connection.

Called by tcp_abort() (to abort a local connection), tcp_closen() (if not all data has been received by the application), tcp_timewait_input(
(if a SYN is received) and tcp_process() (received segment in the wrong state).

Since a RST segment is in most cases not sent for an active connection, tcp_rst() has a number of arguments that are taken from a tcp_pcb for most other segment output functions.

Parameters

pcb	the tcp_pcb for which to re-enqueue all unacked segments

pcb	the tcp_pcb for which to re-enqueue all unacked segments

4.188.6.23 tcp_rst_netif()

```
void tcp_rst_netif (struct netif * netif, u32_t seqno, u32_t ackno, const ip_addr_t * loca
const ip_addr_t * remote_ip, u16_t local_port, u16_t remote_port)
```

Send a TCP RESET packet (empty segment with RST flag set) to show that there is no matching local connection for a received segment.

Called by tcp_input() (if no matching local pcb was found) and tcp_listen_input() (if incoming segment has ACK flag set).

Since a RST segment is in most cases not sent for an active connection, tcp_rst() has a number of arguments that are taken from a tcp_pcb for most other segment output functions.

Parameters

4.188.6.24 tcp_seg_copy()

```
struct tcp_seg * tcp_seg_copy (struct tcp_seg * seg)
```

Returns a copy of the given TCP segment. The pbuf and data are not copied, only the pointers

Parameters

Returns a copy of seg

4.188.6.25 tcp_seg_free()

```
void tcp_seg_free (struct tcp_seg * seg)
```

Frees a TCP segment (tcp_seg structure).

Parameters

4.188.6.26 tcp_segs_free()

```
void tcp_segs_free (struct tcp_seg * seg)
```

Deallocates a list of TCP segments (tcp_seg structures).

Parameters

4.188.6.27 tcp_send_empty_ack()

```
err_t tcp_send_empty_ack (struct tcp_pcb * pcb)
```

Send an ACK without data.

Parameters

pcb	TCP pcb (may be NULL if no pcb is available)
seqno	the sequence number to use for the outgoing segment
ackno	the acknowledge number to use for the outgoing segment
local_ip	the local IP address to send the segment from
remote_ip	the remote IP address to send the segment to
local_port	the local TCP port to send the segment from
remote_port	the remote TCP port to send the segment to

netif	the netif on which to send the RST (since we have no pcb)
seqno	the sequence number to use for the outgoing segment
ackno	the acknowledge number to use for the outgoing segment
local_ip	the local IP address to send the segment from
remote_ip	the remote IP address to send the segment to
local_port	the local TCP port to send the segment from
remote_port	the remote TCP port to send the segment to

seg	the old tcp_seg

4.188.6.28 tcp_send_fin()

```
err_t tcp_send_fin (struct tcp_pcb * pcb)
```

Called by tcp_close() to send a segment including FIN flag but not data. This FIN may be added to an existing segment or a new, otherwise empty segment is enqueued.

Parameters

Returns ERR_OK if sent, another err_t otherwise

4.188.6.29 tcp_slowtmr()

```
void tcp_slowtmr (void )
```

Called every 500 ms and implements the retransmission timer and the timer that removes PCBs that have been in TIME-WAIT for enough time. It also increments various timers such as the inactivity timer in each PCB.

Automatically called from tcp_tmr().

4.188.6.30 tcp_split_unsent_seg()

```
err_t tcp_split_unsent_seg (struct tcp_pcb * pcb, u16_t split)
```

Split segment on the head of the unsent queue. If return is not ERR_OK, existing head remains intact

The split is accomplished by creating a new TCP segment and pbuf which holds the remainder payload after the split. The original pbuf is trimmed to new length. This allows splitting of read-only pbufs

Parameters

4.188.6.31 tcp_timer_needed()

```
void tcp_timer_needed (void )
```

External function (implemented in timers.c), called when TCP detects that a timer is needed (i.e. active- or time-wait-pcb found).

Called from TCP_REG when registering a new PCB: the reason is to have the TCP timer only running when there are active (or time-wait) PCBs.

4.188.6.32 tcp_tmr()

```
void tcp_tmr (void )
```

Called periodically to dispatch TCP timers.

seg	single tcp_seg to free

seg	tcp_seg list of TCP segments to free

pcb	Protocol control block for the TCP connection to send the ACK

4.188.6.33 tcp_txnow()

```
void tcp_txnow (void )
```

Call tcp_output for all active pcbs that have TF_NAGLEMEMERR set

4.188.6.34 tcp_update_rcv_ann_wnd()

```
u32_t tcp_update_rcv_ann_wnd (struct tcp_pcb * pcb)
```

Update the state that tracks the available window space to advertise.

Returns how much extra window would be advertised if we sent an update now.

4.188.6.35 tcp_zero_window_probe()

```
err_t tcp_zero_window_probe (struct tcp_pcb * pcb)
```

Send persist timer zero-window probes to keep a connection active when a window update is lost.

Called by tcp_slowtmr()

Parameters

4.188.7 Variable Documentation

4.188.7.1 tcp_active_pcbs

```
struct tcp_pcb* tcp_active_pcbs[extern]
```

List of all TCP PCBs that are in a state in which they accept or send data.

4.188.7.2 tcp_bound_pcbs

```
struct tcp_pcb* tcp_bound_pcbs[extern]
```

List of all TCP PCBs bound but not yet (connected || listening)

4.188.7.3 tcp_listen_pcbs

```
union tcp_listen_pcbs_t tcp_listen_pcbs[extern]
```

List of all TCP PCBs in LISTEN state

4.188.7.4 tcp_pcb_lists

```
struct tcp_pcb** const tcp_pcb_lists[4][extern]
```

An array with all (non-temporary) PCB lists, mainly used for smaller code size

pcb	the tcp_pcb over which to send a segment

pcb	the tcp_pcb for which to split the unsent head
split	the amount of payload to remain in the head

pcb	the tcp_pcb for which to send a zero-window probe packet

4.188.7.5 tcp_tw_pcbs

```
struct tcp_pcb* tcp_tw_pcbs[extern]
```

List of all TCP PCBs in TIME-WAIT state

4.189 src/include/lwip/priv/tcpip_priv.h File Reference

```
#include "lwip/opt.h"#include "lwip/tcpip.h"#include "lwip/sys.h"#include "lwip/timeouts.h"
```

4.189.1 Functions

- err_t tcpip_send_msg_wait_sem (tcpip_callback_fn fn, void *apimsg, sys_sem_t *sem)
- err_t tcpip_api_call (tcpip_api_call_fn fn, struct tcpip_api_call_data *call)

4.189.2 Detailed Description

TCPIP API internal implementations (do not use in application code)

4.189.3 Function Documentation

4.189.3.1 tcpip_api_call()

```
err_t tcpip_api_call (tcpip_api_call_fn fn, struct tcpip_api_call_data * call)
```

Synchronously calls function in TCPIP thread and waits for its completion. It is recommended to use LWIP_TCPIP_CORE_LOCKING (preferred) or LWIP_NETCONN_SEM_PER_THREAD. If not, a semaphore is created and destroyed on every call which is usually an expensive/slow operation.

Parameters

fn	Function to call
call	Call parameters

Returns Return value from tcpip_api_call_fn

4.189.3.2 tcpip_send_msg_wait_sem()

```
err_t tcpip_send_msg_wait_sem (tcpip_callback_fn fn, void * apimsg, sys_sem_t * sem)
```

Sends a message to TCPIP thread to call a function. Caller thread blocks on on a provided semaphore, which is NOT automatically signalled by TCPIP thread, this has to be done by the user. It is recommended to use LWIP_TCPIP_CORE_LOCKING since this is the way with least runtime overhead.

Parameters

Returns ERR_OK if the function was called, another err_t if not

fn	function to be called from TCPIP thread
apimsg	argument to API function
sem	semaphore to wait on

4.190 src/include/lwip/prot/ethernet.h File Reference

```
#include "lwip/arch.h"#include "lwip/prot/ieee.h"#include "arch/bpstruct.h"#include "arch/ ↩
    epstruct.h"
```

4.190.1 Data Structures

- struct eth_addr
- struct eth_hdr
- struct eth_vlan_hdr

4.190.2 Macros

- #define ETH_ADDR(b0, b1, b2, b3, b4, b5)   {{b0, b1, b2, b3, b4, b5}}
- #define LL_IP4_MULTICAST_ADDR_0   0x01
- #define LL_IP6_MULTICAST_ADDR_0   0x33

4.190.3 Detailed Description

Ethernet protocol definitions

4.190.4 Macro Definition Documentation

4.190.4.1 ETH_ADDR

```
#define ETH_ADDR( b0, b1, b2, b3, b4, b5)   {{b0, b1, b2, b3, b4, b5}}
```

Initialize a struct eth_addr with its 6 bytes (takes care of correct braces)

4.190.4.2 LL_IP4_MULTICAST_ADDR_0

```
#define LL_IP4_MULTICAST_ADDR_0   0x01
```

The 24-bit IANA IPv4-multicast OUI is 01-00-5e:

4.190.4.3 LL_IP6_MULTICAST_ADDR_0

```
#define LL_IP6_MULTICAST_ADDR_0   0x33
```

IPv6 multicast uses this prefix

4.191 src/include/netif/ethernet.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/netif.h"#include "lwip/prot/ ↩
    ethernet.h"
```

4.191.1 Macros

- #define LWIP_ARP_FILTER_NETIF 0

4.191.2 Functions

- err_t ethernet_input (struct pbuf *p, struct netif *netif)
- err_t ethernet_output (struct netif *netif, struct pbuf *p, const struct eth_addr *src, const struct eth_addr *dst, u16_t eth_type)

4.191.3 Detailed Description

Ethernet input function - handles INCOMING ethernet level traffic To be used in most low-level netif implementations

4.191.4 Macro Definition Documentation

4.191.4.1 LWIP_ARP_FILTER_NETIF

```
#define LWIP_ARP_FILTER_NETIF   0
```

Define this to 1 and define LWIP_ARP_FILTER_NETIF_FN(pbuf, netif, type) to a filter function that returns the correct netif when using multiple netifs on one hardware interface where the netif's low-level receive routine cannot decide for the correct netif (e.g. when mapping multiple IP addresses to one hardware interface).

4.192 src/include/lwip/prot/iana.h File Reference

4.192.1 Enumerations

- enum lwip_iana_hwtype { LWIP_IANA_HWTYPE_ETHERNET = 1 }
- enum lwip_iana_port_number { LWIP_IANA_PORT_SMTP = 25 , LWIP_IANA_PORT_DHCP_SERVER = 67 , LWIP_IANA_PORT = 68 , LWIP_IANA_PORT_TFTP = 69 , LWIP_IANA_PORT_HTTP = 80 , LWIP_IANA_PORT_SNTP = 123 , LWIP_IANA_PORT_ = 137 , LWIP_IANA_PORT_SNMP = 161 , LWIP_IANA_PORT_SNMP_TRAP = 162 , LWIP_IANA_PORT_HTTPS = 443 , LWIP_IANA_PORT_SMTPS = 465 , LWIP_IANA_PORT_MQTT = 1883 , LWIP_IANA_PORT_MDNS = 5353 , LWIP_IANA_PORT_SECURE_MQTT = 8883 }

4.192.2 Detailed Description

IANA assigned numbers (RFC 1700 and successors)

4.193 src/include/lwip/prot/ieee.h File Reference

4.193.1 Enumerations

- enum lwip_ieee_eth_type { ETHTYPE_IP = 0x0800U , ETHTYPE_ARP = 0x0806U , ETHTYPE_WOL = 0x0842U , ETHTYPE_RA = 0x8035U , ETHTYPE_VLAN = 0x8100U , ETHTYPE_IPV6 = 0x86DDU , ETHTYPE_PPPOEDISC = 0x8863U , ETHTYPE_PPP = 0x8864U , ETHTYPE_JUMBO = 0x8870U , ETHTYPE_PROFINET = 0x8892U , ETHTYPE_ETHERCAT = 0x88A4U , ETHTYPE_LLDP = 0x88CCU , ETHTYPE_SERCOS = 0x88CDU , ETHTYPE_MRP = 0x88E3U , ETHTYPE_PTP = 0x88F7U , ETHTYPE_QINQ = 0x9100U }

4.193.2 Detailed Description

IEEE assigned numbers

4.194 src/include/lwip/prot/tcp.h File Reference

```
#include "lwip/arch.h"#include "arch/bpstruct.h"#include "arch/epstruct.h"
```

4.194.1 Detailed Description

TCP protocol definitions

4.195 src/include/lwip/tcp.h File Reference

```
#include "lwip/opt.h"#include "lwip/tcpbase.h"#include "lwip/mem.h"#include "lwip/pbuf.h"# ↩
    include "lwip/ip.h"#include "lwip/icmp.h"#include "lwip/err.h"#include "lwip/ip6.h"# ↩
    include "lwip/ip6_addr.h"
```

4.195.1 Data Structures

- struct tcp_ext_arg_callbacks
- struct tcp_pcb_listen
- struct tcp_pcb

4.195.2 Macros

- #define TCP_PCB_COMMON(type)

4.195.3 Typedefs

- typedef err_t(* tcp_accept_fn) (void *arg, struct tcp_pcb *newpcb, err_t err)
- typedef err_t(* tcp_recv_fn) (void *arg, struct tcp_pcb *tpcb, struct pbuf *p, err_t err)
- typedef err_t(* tcp_sent_fn) (void *arg, struct tcp_pcb *tpcb, u16_t len)
- typedef err_t(* tcp_poll_fn) (void *arg, struct tcp_pcb *tpcb)

- typedef void(* tcp_err_fn) (void *arg, err_t err)
- typedef err_t(* tcp_connected_fn) (void *arg, struct tcp_pcb *tpcb, err_t err)
- typedef void(* tcp_extarg_callback_pcb_destroyed_fn) (u8_t id, void *data)
- typedef err_t(* tcp_extarg_callback_passive_open_fn) (u8_t id, struct tcp_pcb_listen *lpcb, struct tcp_pcb *cpcb)

4.195.4 Functions

- struct tcp_pcb * tcp_new (void)
- struct tcp_pcb * tcp_new_ip_type (u8_t type)
- void tcp_arg (struct tcp_pcb *pcb, void *arg)
- void tcp_recv (struct tcp_pcb *pcb, tcp_recv_fn recv)
- void tcp_sent (struct tcp_pcb *pcb, tcp_sent_fn sent)
- void tcp_err (struct tcp_pcb *pcb, tcp_err_fn err)
- void tcp_accept (struct tcp_pcb *pcb, tcp_accept_fn accept)
- void tcp_poll (struct tcp_pcb *pcb, tcp_poll_fn poll, u8_t interval)
- void tcp_backlog_delayed (struct tcp_pcb *pcb)
- void tcp_backlog_accepted (struct tcp_pcb *pcb)
- void tcp_recved (struct tcp_pcb *pcb, u16_t len)
- err_t tcp_bind (struct tcp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port)
- void tcp_bind_netif (struct tcp_pcb *pcb, const struct netif *netif)
- err_t tcp_connect (struct tcp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port, tcp_connected_fn connected)
- struct tcp_pcb * tcp_listen_with_backlog_and_err (struct tcp_pcb *pcb, u8_t backlog, err_t *err)
- struct tcp_pcb * tcp_listen_with_backlog (struct tcp_pcb *pcb, u8_t backlog)
- void tcp_abort (struct tcp_pcb *pcb)
- err_t tcp_close (struct tcp_pcb *pcb)
- err_t tcp_shutdown (struct tcp_pcb *pcb, int shut_rx, int shut_tx)
- err_t tcp_write (struct tcp_pcb *pcb, const void *dataptr, u16_t len, u8_t apiflags)
- void tcp_setprio (struct tcp_pcb *pcb, u8_t prio)
- err_t tcp_output (struct tcp_pcb *pcb)
- u8_t tcp_ext_arg_alloc_id (void)
- void tcp_ext_arg_set_callbacks (struct tcp_pcb *pcb, u8_t id, const struct tcp_ext_arg_callbacks *const callbacks)
- void tcp_ext_arg_set (struct tcp_pcb *pcb, u8_t id, void *arg)
- void * tcp_ext_arg_get (const struct tcp_pcb *pcb, u8_t id)

4.195.5 Detailed Description

TCP API (to be used from TCPIP thread) See also TCP

4.195.6 Macro Definition Documentation

4.195.6.1 TCP_PCB_COMMON

`#define TCP_PCB_COMMON( type)` **Value:**

```
type *next; /* for the linked list */ \
void *callback_arg; \
TCP_PCB_EXTARGS \
enum tcp_state state; /* TCP state */ \
u8_t prio; \
/* ports are in host byte order */ \
u16_t local_port
```

members common to struct tcp_pcb and struct tcp_listen_pcb

4.195.7 Typedef Documentation

4.195.7.1 tcp_accept_fn

`typedef err_t(* tcp_accept_fn) (void *arg, struct tcp_pcb *newpcb, err_t err)`

Function prototype for tcp accept callback functions. Called when a new connection can be accepted on a listening pcb.

Parameters

arg	Additional argument to pass to the callback function (

See also tcp_arg())

Parameters

newpcb	The new connection pcb
err	An error code if there has been an error accepting. Only return ERR_ABRT if you have called tcp_abort from within the callback function!

4.195.7.2 tcp_connected_fn

`typedef err_t(* tcp_connected_fn) (void *arg, struct tcp_pcb *tpcb, err_t err)`

Function prototype for tcp connected callback functions. Called when a pcb is connected to the remote side after initiating a connection attempt by calling tcp_connect().

Parameters

arg	Additional argument to pass to the callback function (

See also tcp_arg())

Parameters

Note
When a connection attempt fails, the error callback is currently called!

tpcb	The connection pcb which is connected
err	An unused error code, always ERR_OK currently ;-)

4.195.7.3 tcp_err_fn

```
typedef void(* tcp_err_fn) (void *arg, err_t err)
```

Function prototype for tcp error callback functions. Called when the pcb receives a RST or is unexpectedly closed for any other reason.

Note
The corresponding pcb is already freed when this callback is called!

Parameters

arg	Additional argument to pass to the callback function (

See also tcp_arg())

Parameters

err	Error code to indicate why the pcb has been closed ERR_ABRT: aborted through tcp_abort or by a TCP timer ERR_RST: the connection was reset by the remote host

4.195.7.4 tcp_extarg_callback_passive_open_fn

```
typedef err_t(* tcp_extarg_callback_passive_open_fn) (u8_t id, struct tcp_pcb_listen *lpcb
struct tcp_pcb *cpcb)
```

Function prototype to transition arguments from a listening pcb to an accepted pcb

Parameters

Returns ERR_OK if OK, any error if connection should be dropped

4.195.7.5 tcp_extarg_callback_pcb_destroyed_fn

```
typedef void(* tcp_extarg_callback_pcb_destroyed_fn) (u8_t id, void *data)
```

Function prototype for deallocation of arguments. Called *just before* the pcb is freed, so don't expect to be able to do anything with this pcb!

Parameters

4.195.7.6 tcp_poll_fn

```
typedef err_t(* tcp_poll_fn) (void *arg, struct tcp_pcb *tpcb)
```

Function prototype for tcp poll callback functions. Called periodically as specified by **See also** tcp_poll.

Parameters

See also tcp_arg())

Parameters

Returns ERR_OK: try to send some data by calling tcp_output Only return ERR_ABRT if you have called tcp_abort from within the callback function!

id	ext arg id (allocated via tcp_ext_arg_alloc_id)
lpcb	the listening pcb accepting a connection
cpcb	the newly allocated connection pcb

id	ext arg id (allocated via tcp_ext_arg_alloc_id)
data	pointer to the data (set via tcp_ext_arg_set before)

4.195.7.7 tcp_recv_fn

```
typedef err_t(* tcp_recv_fn) (void *arg, struct tcp_pcb *tpcb, struct pbuf *p, err_t err)
```

Function prototype for tcp receive callback functions. Called when data has been received.

Parameters

See also tcp_arg())

Parameters

4.195.7.8 tcp_sent_fn

```
typedef err_t(* tcp_sent_fn) (void *arg, struct tcp_pcb *tpcb, u16_t len)
```

Function prototype for tcp sent callback functions. Called when sent data has been acknowledged by the remote side. Use it to free corresponding resources. This also means that the pcb has now space available to send new data.

Parameters

See also tcp_arg())

Parameters

Returns ERR_OK: try to send some data by calling tcp_output Only return ERR_ABRT if you have called tcp_abort from within the callback function!

4.195.8 Function Documentation

4.195.8.1 tcp_setprio()

```
void tcp_setprio (struct tcp_pcb * pcb, u8_t prio)
```

Sets the priority of a connection.

Parameters

4.196 src/include/lwip/prot/udp.h File Reference

```
#include "lwip/arch.h"#include "arch/bpstruct.h"#include "arch/epstruct.h"
```

4.196.1 Detailed Description

UDP protocol definitions

arg	Additional argument to pass to the callback function (

tpcb	tcp pcb

arg	Additional argument to pass to the callback function (

4.197 src/include/lwip/udp.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/netif.h"#include "lwip/ip_addr.h ↩
    "#include "lwip/ip.h"#include "lwip/ip6_addr.h"#include "lwip/prot/udp.h"
```

4.197.1 Data Structures

- struct udp_pcb

4.197.2 Typedefs

- typedef void(* udp_recv_fn) (void *arg, struct udp_pcb *pcb, struct pbuf *p, const ip_addr_t *addr, u16_t port)

4.197.3 Functions

- struct udp_pcb * udp_new (void)
- struct udp_pcb * udp_new_ip_type (u8_t type)
- void udp_remove (struct udp_pcb *pcb)
- err_t udp_bind (struct udp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port)
- void udp_bind_netif (struct udp_pcb *pcb, const struct netif *netif)
- err_t udp_connect (struct udp_pcb *pcb, const ip_addr_t *ipaddr, u16_t port)
- void udp_disconnect (struct udp_pcb *pcb)
- void udp_recv (struct udp_pcb *pcb, udp_recv_fn recv, void *recv_arg)
- err_t udp_sendto_if (struct udp_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, u16_t dst_port, struct netif *netif)
- err_t udp_sendto_if_src (struct udp_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, u16_t dst_port, struct netif *netif, const ip_addr_t *src_ip)
- err_t udp_sendto (struct udp_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, u16_t dst_port)
- err_t udp_send (struct udp_pcb *pcb, struct pbuf *p)
- void udp_input (struct pbuf *p, struct netif *inp)
- void udp_init (void)
- void udp_netif_ip_addr_changed (const ip_addr_t *old_addr, const ip_addr_t *new_addr)

tpcb	The connection pcb which received data
p	The received data (or NULL when the connection has been closed!)
err	An error code if there has been an error receiving Only return ERR_ABRT if you have called tcp_abort from within the callback function!

arg	Additional argument to pass to the callback function (

tpcb	The connection pcb for which data has been acknowledged
len	The amount of bytes acknowledged

4.197.4 Detailed Description

UDP API (to be used from TCPIP thread) See also UDP

4.197.5 Typedef Documentation

4.197.5.1 udp_recv_fn

```
typedef void(* udp_recv_fn) (void *arg, struct udp_pcb *pcb, struct pbuf *p, const ip_addr
*addr, u16_t port)
```

Function prototype for udp pcb receive callback functions addr and port are in same byte order as in the pcb The callback is responsible for freeing the pbuf if it's not used any more.

ATTENTION: Be aware that 'addr' might point into the pbuf 'p' so freeing this pbuf can make 'addr' invalid, too.

Parameters

4.197.6 Function Documentation

4.197.6.1 udp_init()

```
void udp_init (void )
```

Initialize this module.

4.197.6.2 udp_input()

```
void udp_input (struct pbuf * p, struct netif * inp)
```

Process an incoming UDP datagram.

Given an incoming UDP datagram (as a chain of pbufs) this function finds a corresponding UDP PCB and hands over the pbuf to the pcbs recv function. If no pcb is found or the datagram is incorrect, the pbuf is freed.

Parameters

4.197.6.3 udp_netif_ip_addr_changed()

```
void udp_netif_ip_addr_changed (const ip_addr_t * old_addr, const ip_addr_t * new_addr)
```

This function is called from netif.c when address is changed

Parameters

pcb	the tcp_pcb to manipulate
prio	new priority

arg	user supplied argument (udp_pcb.recv_arg)
pcb	the udp_pcb which received data
p	the packet buffer that was received
addr	the remote IP address from which the packet was received
port	the remote port from which the packet was received

p	pbuf to be demultiplexed to a UDP PCB (p->payload pointing to the UDP header)
inp	network interface on which the datagram was received.

4.198 src/include/lwip/raw.h File Reference

```
#include "lwip/opt.h"#include "lwip/pbuf.h"#include "lwip/def.h"#include "lwip/ip.h"# ↩
    include "lwip/ip_addr.h"#include "lwip/ip6_addr.h"
```

4.198.1 Data Structures

- struct raw_pcb

4.198.2 Typedefs

- typedef u8_t(* raw_recv_fn) (void *arg, struct raw_pcb *pcb, struct pbuf *p, const ip_addr_t *addr)

4.198.3 Functions

- struct raw_pcb * raw_new (u8_t proto)
- struct raw_pcb * raw_new_ip_type (u8_t type, u8_t proto)
- void raw_remove (struct raw_pcb *pcb)
- err_t raw_bind (struct raw_pcb *pcb, const ip_addr_t *ipaddr)
- void raw_bind_netif (struct raw_pcb *pcb, const struct netif *netif)
- err_t raw_connect (struct raw_pcb *pcb, const ip_addr_t *ipaddr)
- void raw_disconnect (struct raw_pcb *pcb)
- err_t raw_sendto (struct raw_pcb *pcb, struct pbuf *p, const ip_addr_t *ipaddr)
- err_t raw_sendto_if_src (struct raw_pcb *pcb, struct pbuf *p, const ip_addr_t *dst_ip, struct netif *netif, const ip_addr_t *src_ip)
- err_t raw_send (struct raw_pcb *pcb, struct pbuf *p)
- void raw_recv (struct raw_pcb *pcb, raw_recv_fn recv, void *recv_arg)

4.198.4 Detailed Description

raw API (to be used from TCPIP thread) See also RAW

old_addr	IP address of the netif before change
new_addr	IP address of the netif after change

4.198.5 Typedef Documentation

4.198.5.1 raw_recv_fn

```
typedef u8_t(* raw_recv_fn) (void *arg, struct raw_pcb *pcb, struct pbuf *p, const ip_addr
*addr)
```

Function prototype for raw pcb receive callback functions.

Parameters

arg	user supplied argument (raw_pcb.recv_arg)
pcb	the raw_pcb which received data
p	the packet buffer that was received
addr	the remote IP address from which the packet was received

Returns 1 if the packet was 'eaten' (aka. deleted), 0 if the packet lives on If returning 1, the callback is responsible for freeing the pbuf if it's not used any more.

4.199 src/include/lwip/sockets.h File Reference

```
#include "lwip/opt.h"#include "lwip/ip_addr.h"#include "lwip/netif.h"#include "lwip/err.h"# ↩
    include "lwip/inet.h"#include "lwip/errno.h"#include <string.h>
```

4.199.1 Macros

- #define LWIP_TIMEVAL_PRIVATE 1

4.199.2 Functions

- void lwip_socket_thread_init (void)
- void lwip_socket_thread_cleanup (void)
- int lwip_shutdown (int s, int how)
- int lwip_listen (int s, int backlog)
- int lwip_fcntl (int s, int cmd, int val)

4.199.3 Detailed Description

Socket API (to be used from non-TCPIP threads)

4.199.4 Macro Definition Documentation

4.199.4.1 LWIP_TIMEVAL_PRIVATE

```
#define LWIP_TIMEVAL_PRIVATE   1
```

LWIP_TIMEVAL_PRIVATE: if you want to use the struct timeval provided by your system, set this to 0 and include <sys/time.h> in cc.h

4.199.5 Function Documentation

4.199.5.1 lwip_fcntl()

```
int lwip_fcntl (int s, int cmd, int val)
```

A minimal implementation of fcntl. Currently only the commands F_GETFL and F_SETFL are implemented. The flag O_NONBLOCK and access modes are supported for F_GETFL, only the flag O_NONBLOCK is implemented for F_SETFL.

4.199.5.2 lwip_listen()

```
int lwip_listen (int s, int backlog)
```

Set a socket into listen mode. The socket may not have been used for another connection previously.

Parameters

s	the socket to set to listening mode
backlog	(ATTENTION: needs TCP_LISTEN_BACKLOG=1)

Returns 0 on success, non-zero on failure

4.199.5.3 lwip_shutdown()

```
int lwip_shutdown (int s, int how)
```

Close one end of a full-duplex connection.

4.199.5.4 lwip_socket_thread_cleanup()

```
void lwip_socket_thread_cleanup (void )
```

LWIP_NETCONN_SEM_PER_THREAD==1: destroy thread-local semaphore

4.199.5.5 lwip_socket_thread_init()

```
void lwip_socket_thread_init (void )
```

LWIP_NETCONN_SEM_PER_THREAD==1: initialize thread-local semaphore

4.200 src/include/lwip/stats.h File Reference

```
#include "lwip/opt.h"#include "lwip/mem.h"#include "lwip/memp.h"
```

4.200.1 Data Structures

- struct stats_proto
- struct stats_igmp
- struct stats_mem
- struct stats_syselem

- struct stats_sys
- struct stats_mib2
- struct stats_mib2_netif_ctrs
- struct stats_

4.200.2 Functions

- void stats_init (void)

4.200.3 Variables

- struct stats_ lwip_stats

4.200.4 Detailed Description

Statistics API (to be used from TCPIP thread)

4.200.5 Function Documentation

4.200.5.1 stats_init()

```
void stats_init (void )
```

Init statistics

4.200.6 Variable Documentation

4.200.6.1 lwip_stats

```
struct stats_ lwip_stats[extern]
```

Global variable containing lwIP internal statistics. Add this to your debugger's watchlist.

4.201 src/include/lwip/sys.h File Reference

```
#include "lwip/opt.h"#include "lwip/err.h"#include "arch/sys_arch.h"
```

4.201.1 Macros

- #define SYS_ARCH_TIMEOUT 0xffffffffUL
- #define SYS_MBOX_EMPTY SYS_ARCH_TIMEOUT
- #define LWIP_COMPAT_MUTEX 0
- #define sys_sem_wait(sem) sys_arch_sem_wait(sem, 0)
- #define sys_sem_valid_val(sem) sys_sem_valid(&(sem))
- #define sys_sem_set_invalid_val(sem) sys_sem_set_invalid(&(sem))

- #define sys_mbox_tryfetch(mbox, msg)   sys_arch_mbox_tryfetch(mbox, msg)
- #define sys_mbox_valid_val(mbox)   sys_mbox_valid(&(mbox))
- #define sys_mbox_set_invalid_val(mbox)   sys_mbox_set_invalid(&(mbox))
- #define LWIP_MARK_TCPIP_THREAD()
- #define SYS_ARCH_DECL_PROTECT(lev)   sys_prot_t lev
- #define SYS_ARCH_PROTECT(lev)   lev = sys_arch_protect()
- #define SYS_ARCH_UNPROTECT(lev)   sys_arch_unprotect(lev)

4.201.2 Typedefs

- typedef void(* lwip_thread_fn) (void *arg)

4.201.3 Functions

- err_t sys_mutex_new (sys_mutex_t *mutex)
- void sys_mutex_lock (sys_mutex_t *mutex)
- void sys_mutex_unlock (sys_mutex_t *mutex)
- void sys_mutex_free (sys_mutex_t *mutex)
- int sys_mutex_valid (sys_mutex_t *mutex)
- void sys_mutex_set_invalid (sys_mutex_t *mutex)
- err_t sys_sem_new (sys_sem_t *sem, u8_t count)
- void sys_sem_signal (sys_sem_t *sem)
- u32_t sys_arch_sem_wait (sys_sem_t *sem, u32_t timeout)
- void sys_sem_free (sys_sem_t *sem)
- int sys_sem_valid (sys_sem_t *sem)
- void sys_sem_set_invalid (sys_sem_t *sem)
- void sys_msleep (u32_t ms)
- err_t sys_mbox_new (sys_mbox_t *mbox, int size)
- void sys_mbox_post (sys_mbox_t *mbox, void *msg)
- err_t sys_mbox_trypost (sys_mbox_t *mbox, void *msg)
- err_t sys_mbox_trypost_fromisr (sys_mbox_t *mbox, void *msg)
- u32_t sys_arch_mbox_fetch (sys_mbox_t *mbox, void **msg, u32_t timeout)
- u32_t sys_arch_mbox_tryfetch (sys_mbox_t *mbox, void **msg)
- void sys_mbox_free (sys_mbox_t *mbox)
- int sys_mbox_valid (sys_mbox_t *mbox)
- void sys_mbox_set_invalid (sys_mbox_t *mbox)
- sys_thread_t sys_thread_new (const char *name, lwip_thread_fn thread, void *arg, int stacksize, int prio)
- void sys_init (void)
- u32_t sys_jiffies (void)
- u32_t sys_now (void)

4.201.4 Detailed Description

OS abstraction layer

4.201.5 Macro Definition Documentation

4.201.5.1 LWIP_COMPAT_MUTEX

```
#define LWIP_COMPAT_MUTEX   0
```

Define LWIP_COMPAT_MUTEX if the port has no mutexes and binary semaphores should be used instead

4.201.5.2 SYS_ARCH_TIMEOUT

```
#define SYS_ARCH_TIMEOUT   0xffffffffUL
```

Return code for timeouts from sys_arch_mbox_fetch and sys_arch_sem_wait

4.201.5.3 SYS_MBOX_EMPTY

```
#define SYS_MBOX_EMPTY   SYS_ARCH_TIMEOUT
```

sys_mbox_tryfetch() returns SYS_MBOX_EMPTY if appropriate. For now we use the same magic value, but we allow this to change in future.

4.201.5.4 sys_mbox_set_invalid_val

```
#define sys_mbox_set_invalid_val( mbox)   sys_mbox_set_invalid(&(mbox))
```

Same as sys_mbox_set_invalid() but taking a value, not a pointer

4.201.5.5 sys_mbox_tryfetch

```
#define sys_mbox_tryfetch( mbox, msg)   sys_arch_mbox_tryfetch(mbox, msg)
```

For now, we map straight to sys_arch implementation.

4.201.5.6 sys_mbox_valid_val

```
#define sys_mbox_valid_val( mbox)   sys_mbox_valid(&(mbox))
```

Same as sys_mbox_valid() but taking a value, not a pointer

4.201.5.7 sys_sem_set_invalid_val

```
#define sys_sem_set_invalid_val( sem)   sys_sem_set_invalid(&(sem))
```

Same as sys_sem_set_invalid() but taking a value, not a pointer

4.201.5.8 sys_sem_valid_val

```
#define sys_sem_valid_val( sem)   sys_sem_valid(&(sem))
```

Same as sys_sem_valid() but taking a value, not a pointer

4.201.5.9 sys_sem_wait

```
#define sys_sem_wait( sem)   sys_arch_sem_wait(sem, 0)
```

Wait for a semaphore - forever/no timeout

4.201.6 Typedef Documentation

4.201.6.1 lwip_thread_fn

```
typedef void(* lwip_thread_fn) (void *arg)
```

Function prototype for thread functions

4.201.7 Function Documentation

4.201.7.1 sys_jiffies()

```
u32_t sys_jiffies (void )
```

Ticks/jiffies since power up.

4.202 src/include/lwip/tcpbase.h File Reference

```
#include "lwip/opt.h"
```

4.202.1 Detailed Description

Base TCP API definitions shared by TCP and ALTCP See also TCP

4.203 src/include/lwip/tcpip.h File Reference

```
#include "lwip/opt.h"#include "lwip/err.h"#include "lwip/timeouts.h"#include "lwip/netif.h"
```

4.203.1 Macros

- #define LOCK_TCPIP_CORE()   sys_mutex_lock(&lock_tcpip_core)
- #define UNLOCK_TCPIP_CORE()   sys_mutex_unlock(&lock_tcpip_core)
- #define tcpip_callback_with_block(function, ctx, block)   ((block != 0)? tcpip_callback(function, ctx) : tcpip_try_callback(function, ctx))

4.203.2 Typedefs

- typedef void(* tcpip_init_done_fn) (void *arg)
- typedef void(* tcpip_callback_fn) (void *ctx)

4.203.3 Functions

- void tcpip_init (tcpip_init_done_fn tcpip_init_done, void *arg)
- err_t tcpip_inpkt (struct pbuf *p, struct netif *inp, netif_input_fn input_fn)
- err_t tcpip_input (struct pbuf *p, struct netif *inp)
- err_t tcpip_try_callback (tcpip_callback_fn function, void *ctx)
- err_t tcpip_callback (tcpip_callback_fn function, void *ctx)
- err_t tcpip_callback_wait (tcpip_callback_fn function, void *ctx)
- struct tcpip_callback_msg * tcpip_callbackmsg_new (tcpip_callback_fn function, void *ctx)
- void tcpip_callbackmsg_delete (struct tcpip_callback_msg *msg)
- err_t tcpip_callbackmsg_trycallback (struct tcpip_callback_msg *msg)
- err_t tcpip_callbackmsg_trycallback_fromisr (struct tcpip_callback_msg *msg)
- err_t pbuf_free_callback (struct pbuf *p)
- err_t mem_free_callback (void *m)

4.203.4 Variables

- sys_mutex_t lock_tcpip_core

4.203.5 Detailed Description

Functions to sync with TCPIP thread

4.203.6 Macro Definition Documentation

4.203.6.1 LOCK_TCPIP_CORE

```
#define LOCK_TCPIP_CORE( )    sys_mutex_lock(&lock_tcpip_core)
```

Lock lwIP core mutex (needs LWIP_TCPIP_CORE_LOCKING 1)

4.203.6.2 UNLOCK_TCPIP_CORE

```
#define UNLOCK_TCPIP_CORE( )    sys_mutex_unlock(&lock_tcpip_core)
```

Unlock lwIP core mutex (needs LWIP_TCPIP_CORE_LOCKING 1)

4.203.7 Typedef Documentation

4.203.7.1 tcpip_callback_fn

```
typedef void(* tcpip_callback_fn) (void *ctx)
```

Function prototype for functions passed to tcpip_callback()

4.203.7.2 tcpip_init_done_fn

```
typedef void(* tcpip_init_done_fn) (void *arg)
```

Function prototype for the init_done function passed to tcpip_init

4.203.8 Function Documentation

4.203.8.1 mem_free_callback()

```
err_t mem_free_callback (void * m)
```

A simple wrapper function that allows you to free heap memory from interrupt context.

Parameters

m	the heap memory to free

Returns ERR_OK if callback could be enqueued, an err_t if not

4.203.8.2 pbuf_free_callback()

```
err_t pbuf_free_callback (struct pbuf * p)
```

A simple wrapper function that allows you to free a pbuf from interrupt context.

Parameters

p	The pbuf (chain) to be dereferenced.

Returns ERR_OK if callback could be enqueued, an err_t if not

4.203.8.3 tcpip_callback_wait()

```
err_t tcpip_callback_wait (tcpip_callback_fn function, void * ctx)
```

Sends a message to TCPIP thread to call a function. Caller thread blocks until the function returns. It is recommended to use LWIP_TCPIP_CORE_LOCKING (preferred) or LWIP_NETCONN_SEM_PER_THREAD. If not, a semaphore is created and destroyed on every call which is usually an expensive/slow operation.

Parameters

function	the function to call
ctx	parameter passed to f

Returns ERR_OK if the function was called, another err_t if not

4.203.8.4 tcpip_inpkt()

```
err_t tcpip_inpkt (struct pbuf * p, struct netif * inp, netif_input_fn input_fn)
```

Pass a received packet to tcpip_thread for input processing

Parameters

p	the received packet
inp	the network interface on which the packet was received
input_fn	input function to call

4.203.9 Variable Documentation

4.203.9.1 lock_tcpip_core

```
sys_mutex_t lock_tcpip_core[extern]
```

The global semaphore to lock the stack.

4.204 src/include/lwip/timeouts.h File Reference

```
#include "lwip/opt.h"#include "lwip/err.h"#include "lwip/sys.h"
```

4.204.1 Data Structures

- struct lwip_cyclic_timer

4.204.2 Macros

- #define SYS_TIMEOUTS_SLEEPTIME_INFINITE   0xFFFFFFFF

4.204.3 Typedefs

- typedef void(* lwip_cyclic_timer_handler) (void)
- typedef void(* sys_timeout_handler) (void *arg)

4.204.4 Functions

- void sys_timeouts_init (void)
- void sys_timeout (u32_t msecs, sys_timeout_handler handler, void *arg)
- void sys_untimeout (sys_timeout_handler handler, void *arg)
- void sys_restart_timeouts (void)
- void sys_check_timeouts (void)
- u32_t sys_timeouts_sleeptime (void)

4.204.5 Variables

- const struct lwip_cyclic_timer lwip_cyclic_timers []
- const int lwip_num_cyclic_timers

4.204.6 Detailed Description

Timer implementations

4.204.7 Macro Definition Documentation

4.204.7.1 SYS_TIMEOUTS_SLEEPTIME_INFINITE

```
#define SYS_TIMEOUTS_SLEEPTIME_INFINITE   0xFFFFFFFF
```

Returned by sys_timeouts_sleeptime() to indicate there is no timer, so we can sleep forever.

4.204.8 Typedef Documentation

4.204.8.1 lwip_cyclic_timer_handler

```
typedef void(* lwip_cyclic_timer_handler) (void)
```

Function prototype for a stack-internal timer function that has to be called at a defined interval

4.204.8.2 sys_timeout_handler

```
typedef void(* sys_timeout_handler) (void *arg)
```

Function prototype for a timeout callback function. Register such a function using sys_timeout().

Parameters

arg	Additional argument to pass to the function - set up by sys_timeout()

4.204.9 Function Documentation

4.204.9.1 sys_restart_timeouts()

```
void sys_restart_timeouts (void )
```

Rebase the timeout times to the current time. This is necessary if sys_check_timeouts() hasn't been called for a long time (e.g. while saving energy) to prevent all timer functions of that period being called.

4.204.9.2 sys_timeout()

```
void sys_timeout (u32_t msecs, sys_timeout_handler handler, void * arg)
```

Create a one-shot timer (aka timeout). Timeouts are processed in the following cases:

- while waiting for a message using sys_timeouts_mbox_fetch()
- by calling sys_check_timeouts() (NO_SYS==1 only)

Parameters

4.204.9.3 sys_timeouts_init()

```
void sys_timeouts_init (void )
```

Initialize this module

msecs	time in milliseconds after that the timer should expire
handler	callback function to call when msecs have elapsed
arg	argument to pass to the callback function

4.204.9.4 sys_timeouts_sleeptime()

```
u32_t sys_timeouts_sleeptime (void )
```

Return the time left before the next timeout is due. If no timeouts are enqueued, returns 0xffffffff

4.204.9.5 sys_untimeout()

```
void sys_untimeout (sys_timeout_handler handler, void * arg)
```

Go through timeout list (for this task only) and remove the first matching entry (subsequent entries remain untouched), even though the timeout has not triggered yet.

Parameters

handler	callback function that would be called by the timeout
arg	callback argument that would be passed to handler

4.204.10 Variable Documentation

4.204.10.1 lwip_cyclic_timers

```
const struct lwip_cyclic_timer lwip_cyclic_timers[][extern]
```

This array contains all stack-internal cyclic timers. To get the number of timers, use lwip_num_cyclic_timers

This array contains all stack-internal cyclic timers. To get the number of timers, use LWIP_ARRAYSIZE()

4.204.10.2 lwip_num_cyclic_timers

```
const int lwip_num_cyclic_timers[extern]
```

Array size of lwip_cyclic_timers[]

4.205 src/include/netif/bridgeif.h File Reference

```
#include "netif/bridgeif_opts.h"#include "lwip/err.h"#include "lwip/prot/ethernet.h"# ↩
    include "lwip/tcpip.h"
```

4.205.1 Data Structures

- struct bridgeif_initdata_s

4.205.2 Macros

- #define BRIDGEIF_INITDATA1(max_ports, max_fdb_dynamic_entries, max_fdb_static_entries, ethaddr)   {ethaddr, max_ports, max_fdb_dynamic_entries, max_fdb_static_entries}
- #define BRIDGEIF_INITDATA2(max_ports, max_fdb_dynamic_entries, max_fdb_static_entries, e0, e1, e2, e3, e4, e5)   {{e0, e1, e2, e3, e4, e5}, max_ports, max_fdb_dynamic_entries, max_fdb_static_entries}

4.205.3 Typedefs

- typedef struct bridgeif_initdata_s bridgeif_initdata_t

4.205.4 Functions

- err_t bridgeif_init (struct netif *netif)
- err_t bridgeif_add_port (struct netif *bridgeif, struct netif *portif)
- err_t bridgeif_fdb_add (struct netif *bridgeif, const struct eth_addr *addr, bridgeif_portmask_t ports)
- err_t bridgeif_fdb_remove (struct netif *bridgeif, const struct eth_addr *addr)
- void bridgeif_fdb_update_src (void *fdb_ptr, struct eth_addr *src_addr, u8_t port_idx)
- bridgeif_portmask_t bridgeif_fdb_get_dst_ports (void *fdb_ptr, struct eth_addr *dst_addr)
- void * bridgeif_fdb_init (u16_t max_fdb_entries)

4.205.5 Detailed Description

lwIP netif implementing an IEEE 802.1D MAC Bridge

4.206 src/include/netif/bridgeif_opts.h File Reference

```
#include "lwip/opt.h"
```

4.206.1 Macros

- #define BRIDGEIF_PORT_NETIFS_OUTPUT_DIRECT NO_SYS
- #define BRIDGEIF_MAX_PORTS 7
- #define BRIDGEIF_DEBUG LWIP_DBG_OFF
- #define BRIDGEIF_FDB_DEBUG LWIP_DBG_OFF
- #define BRIDGEIF_FW_DEBUG LWIP_DBG_OFF

4.206.2 Detailed Description

lwIP netif implementing an IEEE 802.1D MAC Bridge

4.207 src/include/netif/ieee802154.h File Reference

```
#include "lwip/opt.h"#include "arch/bpstruct.h"#include "arch/epstruct.h"
```

4.207.1 Data Structures

- struct ieee_802154_hdr

4.207.2 Detailed Description

Definitions for IEEE 802.15.4 MAC frames

4.208 src/include/netif/lowpan6.h File Reference

```
#include "netif/lowpan6_opts.h"#include "netif/lowpan6_common.h"#include "lwip/pbuf.h"# ↩
    include "lwip/ip.h"#include "lwip/ip_addr.h"#include "lwip/netif.h"
```

4.208.1 Macros

- #define LOWPAN6_TMR_INTERVAL 1000

4.208.2 Functions

- void lowpan6_tmr (void)
- err_t lowpan6_set_context (u8_t idx, const ip6_addr_t *context)
- err_t lowpan6_set_short_addr (u8_t addr_high, u8_t addr_low)
- err_t lowpan6_output (struct netif *netif, struct pbuf *q, const ip6_addr_t *ip6addr)
- err_t lowpan6_input (struct pbuf *p, struct netif *netif)
- err_t lowpan6_set_pan_id (u16_t pan_id)
- u16_t lowpan6_calc_crc (const void *buf, u16_t len)
- err_t tcpip_6lowpan_input (struct pbuf *p, struct netif *inp)

4.208.3 Detailed Description

6LowPAN output for IPv6. Uses ND tables for link-layer addressing. Fragments packets to 6LowPAN units.

4.208.4 Macro Definition Documentation

4.208.4.1 LOWPAN6_TMR_INTERVAL

```
#define LOWPAN6_TMR_INTERVAL   1000
```

1 second period for reassembly

4.208.5 Function Documentation

4.208.5.1 lowpan6_calc_crc()

```
u16_t lowpan6_calc_crc (const void * buf, u16_t len)
```

Calculate the 16-bit CRC as required by IEEE 802.15.4

4.208.5.2 lowpan6_tmr()

```
void lowpan6_tmr (void )
```

Periodic timer for 6LowPAN functions:

- Remove incomplete/old packets

4.209 src/include/netif/lowpan6_ble.h File Reference

```
#include "netif/lowpan6_opts.h"#include "netif/lowpan6_common.h"#include "lwip/pbuf.h"# ↩
    include "lwip/ip.h"#include "lwip/ip_addr.h"#include "lwip/netif.h"
```

4.209.1 Functions

- err_t rfc7668_output (struct netif *netif, struct pbuf *q, const ip6_addr_t *ip6addr)
- err_t rfc7668_input (struct pbuf *p, struct netif *netif)
- err_t rfc7668_set_local_addr_eui64 (struct netif *netif, const u8_t *local_addr, size_t local_addr_len)
- err_t rfc7668_set_local_addr_mac48 (struct netif *netif, const u8_t *local_addr, size_t local_addr_len, int is_public_addr)
- err_t rfc7668_set_peer_addr_eui64 (struct netif *netif, const u8_t *peer_addr, size_t peer_addr_len)
- err_t rfc7668_set_peer_addr_mac48 (struct netif *netif, const u8_t *peer_addr, size_t peer_addr_len, int is_public_addr)
- err_t rfc7668_set_context (u8_t index, const ip6_addr_t *context)
- err_t rfc7668_if_init (struct netif *netif)
- err_t tcpip_rfc7668_input (struct pbuf *p, struct netif *inp)
- void ble_addr_to_eui64 (u8_t *dst, const u8_t *src, int public_addr)
- void eui64_to_ble_addr (u8_t *dst, const u8_t *src)

4.209.2 Detailed Description

6LowPAN over BLE for IPv6 (RFC7668).

4.209.3 Function Documentation

4.209.3.1 rfc7668_set_local_addr_eui64()

```
err_t rfc7668_set_local_addr_eui64 (struct netif * netif, const u8_t * local_addr, size_t
local_addr_len)
```

Set the local address used for stateful compression. This expects an address of 8 bytes.

4.209.3.2 rfc7668_set_local_addr_mac48()

```
err_t rfc7668_set_local_addr_mac48 (struct netif * netif, const u8_t * local_addr, size_t
local_addr_len, int is_public_addr)
```

Set the local address used for stateful compression. This expects an address of 6 bytes.

4.209.3.3 rfc7668_set_peer_addr_eui64()

```
err_t rfc7668_set_peer_addr_eui64 (struct netif * netif, const u8_t * peer_addr, size_t
peer_addr_len)
```

Set the peer address used for stateful compression. This expects an address of 8 bytes.

4.209.3.4 rfc7668_set_peer_addr_mac48()

```
err_t rfc7668_set_peer_addr_mac48 (struct netif * netif, const u8_t * peer_addr, size_t
peer_addr_len, int is_public_addr)
```

Set the peer address used for stateful compression. This expects an address of 6 bytes.

4.209.3.5 tcpip_rfc7668_input()

```
err_t tcpip_rfc7668_input (struct pbuf * p, struct netif * inp)
```

Pass a received packet to tcpip_thread for input processing

Parameters

p	the received packet, p->payload pointing to the IEEE 802.15.4 header.
inp	the network interface on which the packet was received

Returns see tcpip_inpkt, same return values

4.210 src/include/netif/lowpan6_common.h File Reference

```
#include "netif/lowpan6_opts.h"#include "lwip/pbuf.h"#include "lwip/ip.h"#include "lwip/ ↩
    ip6_addr.h"#include "lwip/netif.h"
```

4.210.1 Data Structures

- struct lowpan6_link_addr

4.210.2 Detailed Description

Common 6LowPAN routines for IPv6. Uses ND tables for link-layer addressing. Fragments packets to 6LowPAN units.

4.211 src/include/netif/lowpan6_opts.h File Reference

```
#include "lwip/opt.h"
```

4.211.1 Macros

- #define LWIP_6LOWPAN_NUM_CONTEXTS 10
- #define LWIP_6LOWPAN_INFER_SHORT_ADDRESS 1
- #define LWIP_6LOWPAN_IPHC 1
- #define LWIP_6LOWPAN_802154_HW_CRC 0
- #define LWIP_6LOWPAN_CALC_CRC(buf, len) lowpan6_calc_crc(buf, len)
- #define LWIP_LOWPAN6_DEBUG LWIP_DBG_OFF
- #define LWIP_LOWPAN6_802154_DEBUG LWIP_DBG_OFF
- #define LWIP_LOWPAN6_IP_COMPRESSED_DEBUG LWIP_DBG_OFF
- #define LWIP_LOWPAN6_DECOMPRESSION_DEBUG LWIP_DBG_OFF
- #define LWIP_RFC7668_IP_UNCOMPRESSED_DEBUG LWIP_DBG_OFF
- #define LWIP_RFC7668_LINUX_WORKAROUND_PUBLIC_ADDRESS 1

4.211.2 Detailed Description

6LowPAN options list

4.211.3 Macro Definition Documentation

4.211.3.1 LWIP_6LOWPAN_802154_HW_CRC

```
#define LWIP_6LOWPAN_802154_HW_CRC   0
```

Set this to 1 if your IEEE 802.15.4 interface can calculate and check the CRC in hardware. This means TX packets get 2 zero bytes added on transmission which are to be filled with the CRC.

4.211.3.2 LWIP_6LOWPAN_CALC_CRC

```
#define LWIP_6LOWPAN_CALC_CRC( buf, len)   lowpan6_calc_crc(buf, len)
```

If LWIP_6LOWPAN_802154_HW_CRC==0, this can override the default slow implementation of the CRC used for 6LoWPAN over IEEE 802.15.4 (which uses a shift register).

4.211.3.3 LWIP_6LOWPAN_INFER_SHORT_ADDRESS

```
#define LWIP_6LOWPAN_INFER_SHORT_ADDRESS   1
```

LWIP_6LOWPAN_INFER_SHORT_ADDRESS: set this to 0 to disable creating short addresses for matching addresses (debug only)

4.211.3.4 LWIP_6LOWPAN_IPHC

```
#define LWIP_6LOWPAN_IPHC   1
```

LWIP_6LOWPAN_IPHC: set this to 0 to disable IP header compression as per RFC 6282 (which is mandatory for BLE)

4.211.3.5 LWIP_6LOWPAN_NUM_CONTEXTS

```
#define LWIP_6LOWPAN_NUM_CONTEXTS   10
```

LWIP_6LOWPAN_NUM_CONTEXTS: define the number of compression contexts per netif type

4.211.3.6 LWIP_LOWPAN6_802154_DEBUG

```
#define LWIP_LOWPAN6_802154_DEBUG   LWIP_DBG_OFF
```

Debug level for 6LoWPAN over IEEE 802.15.4

4.211.3.7 LWIP_LOWPAN6_DEBUG

```
#define LWIP_LOWPAN6_DEBUG   LWIP_DBG_OFF
```

Debug level for 6LoWPAN in general

4.211.3.8 LWIP_LOWPAN6_DECOMPRESSION_DEBUG

```
#define LWIP_LOWPAN6_DECOMPRESSION_DEBUG   LWIP_DBG_OFF
```

LWIP_LOWPAN6_DECOMPRESSION_DEBUG: enable decompression debug output

4.211.3.9 LWIP_LOWPAN6_IP_COMPRESSED_DEBUG

```
#define LWIP_LOWPAN6_IP_COMPRESSED_DEBUG   LWIP_DBG_OFF
```

LWIP_LOWPAN6_IP_COMPRESSED_DEBUG: enable compressed IP frame output debugging

4.211.3.10 LWIP_RFC7668_IP_UNCOMPRESSED_DEBUG

```
#define LWIP_RFC7668_IP_UNCOMPRESSED_DEBUG   LWIP_DBG_OFF
```

LWIP_RFC7668_IP_UNCOMPRESSED_DEBUG: enable decompressed IP frame output debugging

4.211.3.11 LWIP_RFC7668_LINUX_WORKAROUND_PUBLIC_ADDRESS

```
#define LWIP_RFC7668_LINUX_WORKAROUND_PUBLIC_ADDRESS   1
```

LWIP_RFC7668_LINUX_WORKAROUND_PUBLIC_ADDRESS: Currently, the linux kernel driver for 6lowpan sets/clears a bit in the address, depending on the BD address (either public or not). Might not be RFC7668 conform, so you may select to do that (=1) or not (=0)

4.212 src/include/netif/ppp/pppol2tp.h File Reference

```
#include "netif/ppp/ppp_opts.h"
```

4.212.1 Detailed Description

Network Point to Point Protocol over Layer 2 Tunneling Protocol header file.

4.213 src/include/netif/ppp/pppos.h File Reference

```
#include "netif/ppp/ppp_opts.h"
```

4.213.1 Detailed Description

Network Point to Point Protocol over Serial header file.

4.214 src/include/netif/slipif.h File Reference

```
#include "lwip/opt.h"#include "lwip/netif.h"
```

4.214.1 Macros

- #define SLIP_USE_RX_THREAD !NO_SYS
- #define SLIP_RX_QUEUE SLIP_RX_FROM_ISR

4.214.2 Functions

- err_t slipif_init (struct netif *netif)
- void slipif_poll (struct netif *netif)
- void slipif_process_rxqueue (struct netif *netif)
- void slipif_received_byte (struct netif *netif, u8_t data)
- void slipif_received_bytes (struct netif *netif, u8_t *data, u8_t len)

4.214.3 Detailed Description

SLIP netif API

4.214.4 Macro Definition Documentation

4.214.4.1 SLIP_RX_QUEUE

```
#define SLIP_RX_QUEUE   SLIP_RX_FROM_ISR
```

Set this to 1 to enable functions to pass in RX bytes from ISR context. If enabled, slipif_received_byte[s]() process incoming bytes and put assembled packets on a queue, which is fed into lwIP from slipif_poll(). If disabled, slipif_poll() polls the serial line (using sio_tryread()). Set this to 1 (default for SLIP_RX_FROM_ISR) to queue incoming packets received by slipif_received_byte[s]() as long as PBUF_POOL pbufs are available. If disabled, packets will be dropped if more than one packet is received.

4.214.4.2 SLIP_USE_RX_THREAD

```
#define SLIP_USE_RX_THREAD   !NO_SYS
```

Set this to 1 to start a thread that blocks reading on the serial line (using sio_read()).

4.215 src/include/netif/zepif.h File Reference

```
#include "lwip/opt.h"#include "netif/lowpan6.h"#include "lwip/netif.h"
```

4.215.1 Data Structures

- struct zepif_init

4.215.2 Functions

- err_t zepif_init (struct netif *netif)

4.215.3 Detailed Description

A netif implementing the ZigBee Eencapsulation Protocol (ZEP). This is used to tunnel 6LowPAN over UDP.

4.216 src/netif/bridgeif.c File Reference

```
#include "netif/bridgeif.h"#include "lwip/netif.h"#include "lwip/sys.h"#include "lwip/ ↩
    etharp.h"#include "lwip/ethip6.h"#include "lwip/snmp.h"#include "lwip/timeouts.h"# ↩
    include <string.h>
```

4.216.1 Functions

- err_t bridgeif_fdb_add (struct netif *bridgeif, const struct eth_addr *addr, bridgeif_portmask_t ports)
- err_t bridgeif_fdb_remove (struct netif *bridgeif, const struct eth_addr *addr)
- err_t bridgeif_init (struct netif *netif)
- err_t bridgeif_add_port (struct netif *bridgeif, struct netif *portif)

4.216.2 Detailed Description

lwIP netif implementing an IEEE 802.1D MAC Bridge

4.217 src/netif/bridgeif_fdb.c File Reference

```
#include "netif/bridgeif.h"#include "lwip/sys.h"#include "lwip/mem.h"#include "lwip/ ↩
    timeouts.h"#include <string.h>
```

4.217.1 Functions

- void bridgeif_fdb_update_src (void *fdb_ptr, struct eth_addr *src_addr, u8_t port_idx)
- bridgeif_portmask_t bridgeif_fdb_get_dst_ports (void *fdb_ptr, struct eth_addr *dst_addr)
- void * bridgeif_fdb_init (u16_t max_fdb_entries)

4.217.2 Detailed Description

lwIP netif implementing an FDB for IEEE 802.1D MAC Bridge

4.218 src/netif/ethernet.c File Reference

```
#include "lwip/opt.h"#include "netif/ethernet.h"#include "lwip/def.h"#include "lwip/stats.h ↩
    "#include "lwip/etharp.h"#include "lwip/ip.h"#include "lwip/snmp.h"#include <string.h># ↩
    include "netif/ppp/ppp_opts.h"#include "path/to/my/lwip_hooks.h"
```

4.218.1 Functions

- err_t ethernet_input (struct pbuf *p, struct netif *netif)
- err_t ethernet_output (struct netif *netif, struct pbuf *p, const struct eth_addr *src, const struct eth_addr *dst, u16_t eth_type)

4.218.2 Detailed Description

Ethernet common functions

4.219 src/netif/lowpan6.c File Reference

```
#include "netif/lowpan6.h"#include "lwip/ip.h"#include "lwip/pbuf.h"#include "lwip/ip_addr. ↩
    h"#include "lwip/netif.h"#include "lwip/nd6.h"#include "lwip/mem.h"#include "lwip/udp.h ↩
    "#include "lwip/tcpip.h"#include "lwip/snmp.h"#include "netif/ieee802154.h"#include < ↩
    string.h>
```

4.219.1 Data Structures

- struct lowpan6_reass_helper
- struct lowpan6_ieee802154_data

4.219.2 Functions

- u16_t lowpan6_calc_crc (const void *buf, u16_t len)
- void lowpan6_tmr (void)
- err_t lowpan6_set_context (u8_t idx, const ip6_addr_t *context)
- err_t lowpan6_set_short_addr (u8_t addr_high, u8_t addr_low)
- err_t lowpan6_output (struct netif *netif, struct pbuf *q, const ip6_addr_t *ip6addr)
- err_t lowpan6_input (struct pbuf *p, struct netif *netif)
- err_t lowpan6_set_pan_id (u16_t pan_id)
- err_t tcpip_6lowpan_input (struct pbuf *p, struct netif *inp)

4.219.3 Detailed Description

6LowPAN output for IPv6. Uses ND tables for link-layer addressing. Fragments packets to 6LowPAN units.

This implementation aims to conform to IEEE 802.15.4(-2015), RFC 4944 and RFC 6282.

4.219.4 Function Documentation

4.219.4.1 lowpan6_calc_crc()

`u16_t lowpan6_calc_crc (const void * buf, u16_t len)`

Calculate the 16-bit CRC as required by IEEE 802.15.4

4.219.4.2 lowpan6_tmr()

`void lowpan6_tmr (void )`

Periodic timer for 6LowPAN functions:

- Remove incomplete/old packets

4.220 src/netif/lowpan6_ble.c File Reference

```
#include "netif/lowpan6_ble.h"#include "lwip/ip.h"#include "lwip/pbuf.h"#include "lwip/ ↩
    ip_addr.h"#include "lwip/netif.h"#include "lwip/nd6.h"#include "lwip/mem.h"#include " ↩
    lwip/udp.h"#include "lwip/tcpip.h"#include "lwip/snmp.h"#include <string.h>
```

4.220.1 Functions

- void ble_addr_to_eui64 (u8_t *dst, const u8_t *src, int public_addr)
- void eui64_to_ble_addr (u8_t *dst, const u8_t *src)
- err_t rfc7668_set_local_addr_eui64 (struct netif *netif, const u8_t *local_addr, size_t local_addr_len)
- err_t rfc7668_set_local_addr_mac48 (struct netif *netif, const u8_t *local_addr, size_t local_addr_len, int is_public_addr)
- err_t rfc7668_set_peer_addr_eui64 (struct netif *netif, const u8_t *peer_addr, size_t peer_addr_len)
- err_t rfc7668_set_peer_addr_mac48 (struct netif *netif, const u8_t *peer_addr, size_t peer_addr_len, int is_public_addr)
- err_t rfc7668_set_context (u8_t idx, const ip6_addr_t *context)
- err_t rfc7668_output (struct netif *netif, struct pbuf *q, const ip6_addr_t *ip6addr)
- err_t rfc7668_input (struct pbuf *p, struct netif *netif)
- err_t rfc7668_if_init (struct netif *netif)
- err_t tcpip_rfc7668_input (struct pbuf *p, struct netif *inp)

4.220.2 Detailed Description

6LowPAN over BLE output for IPv6 (RFC7668).

4.220.3 Function Documentation

4.220.3.1 rfc7668_set_local_addr_eui64()

```
err_t rfc7668_set_local_addr_eui64 (struct netif * netif, const u8_t * local_addr, size_t
local_addr_len)
```

Set the local address used for stateful compression. This expects an address of 8 bytes.

4.220.3.2 rfc7668_set_local_addr_mac48()

```
err_t rfc7668_set_local_addr_mac48 (struct netif * netif, const u8_t * local_addr, size_t
local_addr_len, int is_public_addr)
```

Set the local address used for stateful compression. This expects an address of 6 bytes.

4.220.3.3 rfc7668_set_peer_addr_eui64()

```
err_t rfc7668_set_peer_addr_eui64 (struct netif * netif, const u8_t * peer_addr, size_t
peer_addr_len)
```

Set the peer address used for stateful compression. This expects an address of 8 bytes.

4.220.3.4 rfc7668_set_peer_addr_mac48()

```
err_t rfc7668_set_peer_addr_mac48 (struct netif * netif, const u8_t * peer_addr, size_t
peer_addr_len, int is_public_addr)
```

Set the peer address used for stateful compression. This expects an address of 6 bytes.

4.220.3.5 tcpip_rfc7668_input()

```
err_t tcpip_rfc7668_input (struct pbuf * p, struct netif * inp)
```

Pass a received packet to tcpip_thread for input processing

Parameters

p	the received packet, p->payload pointing to the IEEE 802.15.4 header.
inp	the network interface on which the packet was received

Returns see tcpip_inpkt, same return values

4.221 src/netif/lowpan6_common.c File Reference

```
#include "netif/lowpan6_common.h"#include "lwip/ip.h"#include "lwip/pbuf.h"#include "lwip/ ↩
    ip_addr.h"#include "lwip/netif.h"#include "lwip/udp.h"#include <string.h>
```

4.221.1 Detailed Description

Common 6LowPAN routines for IPv6. Uses ND tables for link-layer addressing. Fragments packets to 6LowPAN units.

This implementation aims to conform to IEEE 802.15.4(-2015), RFC 4944 and RFC 6282.

4.222 src/netif/ppp/pppapi.c File Reference

```
#include "netif/ppp/ppp_opts.h"
```

4.222.1 Detailed Description

Point To Point Protocol Sequential API module

4.223 src/netif/ppp/pppol2tp.c File Reference

```
#include "netif/ppp/ppp_opts.h"
```

4.223.1 Detailed Description

Network Point to Point Protocol over Layer 2 Tunneling Protocol program file.

4.224 src/netif/ppp/pppos.c File Reference

```
#include "netif/ppp/ppp_opts.h"
```

4.224.1 Detailed Description

Network Point to Point Protocol over Serial file.

4.225 src/netif/slipif.c File Reference

```
#include "netif/slipif.h"#include "lwip/opt.h"#include "lwip/def.h"#include "lwip/pbuf.h"# ↩
    include "lwip/stats.h"#include "lwip/snmp.h"#include "lwip/sys.h"#include "lwip/sio.h"
```

4.225.1 Macros

- #define SLIP_MAX_SIZE 1500
- #define SLIP_SIO_SPEED(sio_fd) 0

4.225.2 Functions

- err_t slipif_init (struct netif *netif)
- void slipif_poll (struct netif *netif)
- void slipif_process_rxqueue (struct netif *netif)
- void slipif_received_byte (struct netif *netif, u8_t data)
- void slipif_received_bytes (struct netif *netif, u8_t *data, u8_t len)

4.225.3 Detailed Description

SLIP Interface

4.225.4 Macro Definition Documentation

4.225.4.1 SLIP_MAX_SIZE

```
#define SLIP_MAX_SIZE   1500
```

Maximum packet size that is received by this netif

4.225.4.2 SLIP_SIO_SPEED

```
#define SLIP_SIO_SPEED( sio_fd)   0
```

Define this to the interface speed for SNMP (sio_fd is the sio_fd_t returned by sio_open). The default value of zero means 'unknown'.

4.226 src/netif/zepif.c File Reference

```
#include "netif/zepif.h"#include "netif/lowpan6.h"#include "lwip/udp.h"#include "lwip/ ↩
    timeouts.h"#include <string.h>#include "arch/bpstruct.h"#include "arch/epstruct.h"
```

4.226.1 Macros

- #define ZEPIF_LOOPBACK 0

4.226.2 Functions

- err_t zepif_init (struct netif *netif)

4.226.3 Macro Definition Documentation

4.226.3.1 ZEPIF_LOOPBACK

```
#define ZEPIF_LOOPBACK   0
```

Define this to 1 to loop back TX packets for testing