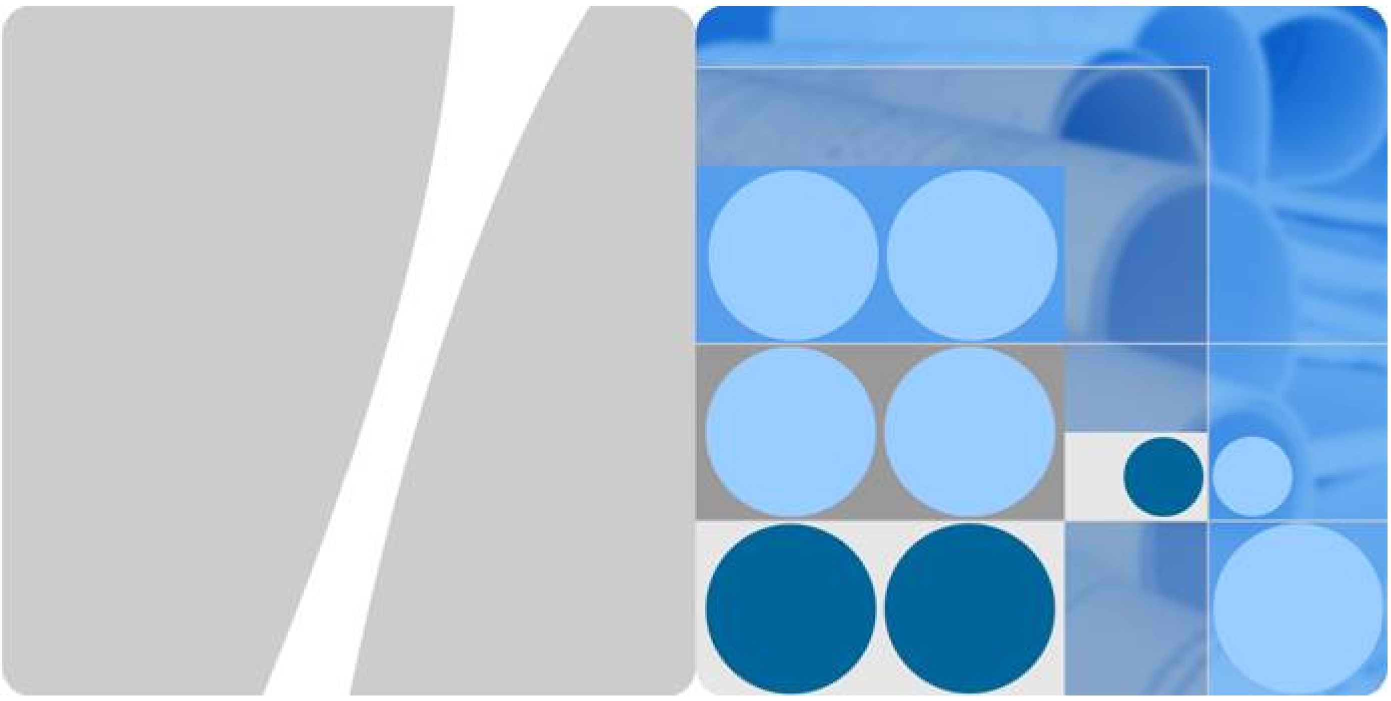


Huawei LiteOS LwIP Developer Guide

Issue 01

Date 2017-10-21

HUAWEI TECHNOLOGIES CO., LTD.

Huawei Technologies Co., Ltd.

Address:	Huawei Industrial Base Bantian, Longgang Shenzhen 518129 People's Republic of China
Website:	http://www.huawei.com
Email:	support@huawei.com

Contents

1 Huawei LiteOS lwIP Developer Guide

The Huawei lwIP Developer Guide provides information about the Huawei LiteOS lwIP. It includes programming information, sample codes, descriptions for functions, structures, callback functions, constants, and a brief introduction to the lwIP protocol.

This document is for Huawei LiteOS lwIP V100R001C20.

This guide is organized as follows:

1.1 About This Document

This chapter describes the scope, intended audience, and revision history of the document. It also provides the contact details for technical support.

1.1.1 Scope of the Document

The main focus of Huawei LiteOS lwIP development is to adapt lwIP to work with Huawei LiteOS and also to implement some additional features like DNS client, DHCP server and shell command.

1.1.2 Intended Audience

The intended audience of this document are:

- Developers developing applications on Huawei LiteOS.
- Module Leads
- Testers
- Test Leads
- System Engineers
- Test System Engineers
- Members of the Cooperating Teams
- Documentation Developers

These users will be primarily developing applications on top of the various supported features.

1.1.3 Revision History

This topic describes the revision history of the document.

Revision History	Date	Change Description	Author
VPP V100R001C00 (lwIP)	10-May-2016	● Initial Version	VPP IDG Team

Revision History	Date	Change Description	Author
VPP V100R001C10 (lwIP)	27-Jul-2016	● **[VPPTECH-26]:** BSD like socket interface (support multi-thread bidirectional data transfer). ● **[VPPTECH-161]:** lwIP provides API for getting the TCP/IP, UDP/IP connect information. Refer to the document Huawei LiteOS lwIP API Reference.chm ● **[VPPTECH-159]:** lwIP provides an API for configuring the pbuf RAM size. Refer to the document Huawei LiteOS lwIP API Reference.chm ● **[VPPTECH-160]**: lwIP provides API for setting the wifi driver status. Refer to the document Huawei LiteOS lwIP API Reference.chm ● lwIP provides a secure SSP callback registration API for registering mandatory user secure functions.	VPP IDG Team
VPP V100R001C20 (lwIP)	24-Oct-2016	● **[VPPTECH-238]:** lwIP provides PF_PACKET option on SOCK_RAW.	VPP IDG Team

1.1.4 Contact Us

This topic gives the detail information about the contact person.

How to contact us

You can contact us by sending an email to ashutosh.prakash@huawei.com.

Services

You may contact the above mentioned account for any queries relating to the usage of the application. We will reply to you within two working days. In order to help us answer your questions in time, Kindly submit your questions with as many details as possible, such as providing the information about the version, the OS adopted, and so on.This ,will help us to address your queries quickly and more effectively.

Defect

In case of any defects, kindly mail your feedback to the respective developer. For IDG related issues you can send mail to ayan.dutta@huawei.com.

Requirements

You are welcome to submit your requirements of lwIP algorithm to Huawei Customer Requirement Electronic Flow.

Technical Communication

Our lwIP team is willing to share the technical achievements with any other product developers and is hoping to borrow your valuable experiences as well. If necessary, contact us through this account.

Training

This mainly includes the knowledge for beginners and introduction to the subject on lwIP algorithm.

Others

Our lwIP team is always open to any opinions, suggestions and comments to meet your expectations.

1.1.5 Copyright

This topic gives information about the copyright.

statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

1.2 Introduction to Huawei LiteOS lwIP

This chapter provides a brief description about Huawei LiteOS lwIP.

1.2.1 Background

Over the last few years, the interest for connecting computers and computer supported devices to wireless networks has steadily increased. Computers are becoming more and more seamlessly integrated with everyday equipments, and prices are dropping. At the same time, wireless networking technologies such as Bluetooth and IEEE 802.11b/g (WiFi), have become common. This creates many new fascinating scenarios in areas such as healthcare, safety and security, transportation, and processing. Small devices such as sensors can be connected to an existing network infrastructure such as the global Internet, and monitored from anywhere.

Huawei is entering this IoT (Internet of Things) technology with connected sensors and monitoring devices. The main challenge in connected devices is to implement the light weight OS and protocol stacks, which helps in reduction of manufacturing cost. And also Huawei is going to migrate all existing conected device to light weight protocol stack. For example existing Huawei's product IP camera (IPC) uses Embedded Linux OS and BSD Linux TCP/IP stack. This requires huge memory (RAM and ROM) for its execution. Huawei wanted to find an alternate light weight OS and TCP/IP stack for IPC (without compromising performance), which can reduce the memory requirement by half and it saves 20 RMB per device from manufacturing perspective.

1.2.2 Purpose

Huawei was using Embedded Linux and BSD TCP/IP stack on the IP camera product. To migrate the IP camera product to a lighe weight OS and TCP/IP stack, the Huawei Euler team developed the Huawei LiteOS, and Huawei VPP team had to implement a light weight TCP/IP stack. VPP analyzed and concluded that the open source lwIP TCP/IP stack suits this requirement and ported the lwIP TCP/IP stacke to Huawei LiteOS for the IP camera product.

lwIP is a light-weight implementation of the TCP/IP protocol suite. The main goal of this TCP/IP implementation is to reduce the RAM usage with full feature of TCP. This light-weight IP stack is customized to run on top of Huawei LiteOS and is optimized to deliver a high throughput. lwIP also implements some additional features like DHCP server and network shell commands (for example, ping, arp, ifconfig).

1.2.3 Scope

The main focus of Huawei LiteOS lwIP development is to adapt lwIP to work with LiteOS. Some additional features like DNS client, DHCP server, and shell command support are implemented.

1.2.4 Third Party References

The following third party references are used for Huawei LiteOS lwIP development:

- SNTP is not part of lwIP TCP/IP stack. There is an implementation available for SNTP in lwIP contrib, that has been used in our Huawei LiteOS lwIP.

- For OS adaptation layer implementation, lwIP contrib has a Linux adapter code that has been used for LiteOS integration, because LiteOS also supports most of the POSIX-based system calls.

1.2.5 Standard Compliance

Huawei LiteOS lwIP implements the following RFCs:

- RFC 791 (IPv4 Standard)
- RFC 768 (UDP)
- RFC 793 (TCP)
- RFC 792 (ICMP)
- RFC 826 (ARP)
- RFC 2131 (DHCP)
- RFC 854 (Telnet)
- RFC 2018 (SACK)
- RFC 7323 (Window Scaling)
- RFC 6675 (SACK for TCP)
- RFC 3927(Autoip) Dynamic Configuration of IPv4 Link-Local Addresses
- RFC 2236 (IGMP) Internet Group Management Protocol, Version 2

1.3 Huawei LiteOS lwIP Features

This chapter provides a brief description about the supported features of Huawei LiteOS lwIP.

1.3.1 System Requirements

Huawei LiteOS lwIP requires the following:

- Lite-OS on HiSilicon IP Camera or compatible board.
- arm-hisiv300-linux-gcc or arm-hisiv500-linux-gcc for compiling Huawei LiteOS lwIP.

Huawei LiteOS lwIP supports Huawei LiteOS on the following platforms:

- hi3516a
- hi3518ev200
- hi3519_cortex-a7
- hi3519_cortex-a17

1.3.2 Supported Features

Huawei LiteOS lwIP supports the following features:

IPv4 (Internet Protocol version 4)

IPv4 is the fourth version in the development of Internet Protocol (IP). IPv4 is one of the core protocols of standards-based working methods in the Internet, and was the first version deployed for production in the ARPANET in 1983. IPv4 is a connectionless protocol for use on packet-switched networks. IPv4 operates on a best effort delivery model, in that it does not guarantee delivery, nor does it assure proper sequencing or avoidance of duplicate delivery.

ICMP (Internet Control Message Protocol)

ICMP is a core protocol of the Internet Protocol Suite. ICMP is used to send notification messages such as "host not reachable", and also to send ping messages such as "ECHO Request" and "ECHO Reply".

UDP (User Datagram Protocol)

UDP is a core protocol of the Internet Protocol Suite. UDP uses a simple connectionless transmission model with a minimal protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

TCP (Transmission Control Protocol)

TCP is a core protocol of the Internet Protocol Suite. TCP originated in the initial network implementation in which it complemented IP. Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network. Applications that do not require reliable data stream service may use UDP, which provides a connectionless datagram service that emphasizes reduced latency over reliability.

DNS (Domain names resolver) client

DNS is a hierarchical distributed naming system for computers, services, or any resource connected to the Internet or a private network. DNS client with minimal feature (support for a name resolution) based on RFC 1035 is supported in Huawei LiteOS lwIP.

DHCP (Dynamic Host Configuration Protocol) both client and server

DHCP is a standardized network protocol used on IP networks for dynamically distributing network configuration parameters, such as IP addresses for interfaces and services. With DHCP, computers request IP addresses and networking parameters automatically from a DHCP server, reducing the need for a network administrator or a user to configure these settings manually. Huawei LiteOS lwIP supports both client and server of DHCP protocol based on RFC 2131.

ARP (Address Resolution Protocol) for Ethernet

ARP is a telecommunication protocol used for resolution of network layer addresses into link layer addresses, a critical function in multiple-access networks.

Huawei LiteOS lwIP supports ARP protocol based on RFC 826 with a configurable ARP table.

PPPoE (Point to Point Protocol over Ethernet) client

PPPoE provides a standard for connecting multiple clients on an Ethernet local area network (LAN) network to a remote broadband access server (BAS). Ethernet is used to connect multiple PPPoE clients and form a LAN. Through the remote BAS, the clients can be connected to the Internet. Identity authentication and charging for each accessed client are achieved by using the PPP.

IGMP（Internet Group Management Protocol）

IGMP is used for communicating the information of the multicast group members between the host and the local router.

Huawei LiteOS lwIP supports IGMP v2 protocol based on RFC 2236.

Although the RFC 2236 requires IGMP v1 AND v2 capability we will only support v2 since now v1 is very old (August 1989)

Shell commands

Huawei LiteOS lwIP provides APIs for some of the commonly used networking shell commands such as ping, ifconfig, arp, ifup, ifdown, and dns. This module provides shell command handler function based on the LiteOS shell command module.

Socket API Types

Huawei LiteOS lwIP provides the following types of socket APIs.

- Low-level APIs for using without thread.
- Netconn APIs with thread
- BSD styled APIs which internally calls netconn APIs. Providing BSD styled APIs help the application for smooth migration from Linux TCP/IP stack to the lwIP TCP/IP stack.

SNTP (Simple Network Time Protocol)

SNTP is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks. Huawei LiteOS lwIP supports SNTP version 4 based on RFC 2030.

TCP Window Scaling Option

Huawei LiteOS lwIP supports Window Scaling option in TCP as per RFC 7323. Window Scaling option in TCP allows you to use a 30-bit window size value in a TCP connection, instead of 16-bit value. The window scale extension expands the definition of the TCP window to 30 bits and then uses an implicit scale factor to carry this 30-bit value in the 16-bit window field of the TCP header. The exponent of the scale factor is carried in a TCP option named Window Scale.

TCP SACK (Selective Acknowledgement) Option

This option is used to acknowledge out of sequence segments received by the stack so that you can skip these sacked segments for retransmission during loss recovery.

The following features are developed based on PDT requirements:

- **lwIP supports AutoIP and IGMP modules .**
- **Performance optimization changes with the DMA allocation change for zero copy.**
- **[BVT OR:201602239352]: lwIP supports Multi-threading for BSD styled sockets.**
- **[LiteOS OR:201606029490]: lwip provides API for setting the wifi driver status**. lwip provides API for setting the wifi driver status to lwip stack. When the wifi driver is busy, lwip stack stop to send message, when the wifi driver is ready, lwip stack continue to send the message. If the driver does not wake up from busy state before the expiry of DRIVER_WAKEUP_INTERVAL, then all the exisiting TCP connections using this netif driver are purged and removed. Hence, a blocking connect call will wait till the netif driver wakes up and a SYN retransmission occurs or till the TCP connection is purged due to the above timeout.

- **[LiteOS OR:201607265325]: lwip provides PF_PACKET option on SOCK_RAW**. LWIP supports SOCK_RAW for PF_PACKET family. Application can use this feature to create link layer level sockets. So, the lwip stack expects packets with link layer headers included in application for sending them out. SOCK_RAW packets are passed to and from the device driver without any changes in the packet data.

  By default all packets of the specified protocol type are passed to a packet socket. To only get packets from a specific interface, bind the socket to a specific interface. The sll_protocol and the sll_ifindex address fields are used for purposes of binding. When application sends packets it is enough to specify sll_family, sll_addr, sll_halen, sll_ifindex. The other fields should be 0. sll_hatype and sll_pkttype are set on received packets.

PF_PACKET supports the following APIs:

- lwip_socket
- lwip_bind
- lwip_recvfrom
- lwip_sendto
- lwip_close
- lwip_ioctl
- lwip_setsockopt
- lwip_getsockopt
- lwip_fnct

NOTE

Refer VPP2.0 V100R001C20 Huawei LiteOS lwIP API Reference for descriptions of these APIs.

The following options are supported for PF_PACKET socket s lwip_getsockopt() and lwip_setsockopt() APIs:

- SO_RCVTIMEO
- SO_RCVBUF
- SO_TYPE

The following socket options are supported for PF_PACKET socket under lwip_ioctl() API:

- FIONREAD
- FIONBIO
- SIOCGIFADDR
- SIOCSIFADDR
- SIOCGIFNETMASK,
- SIOCSIFNETMASK
- SIOCSIFHWADDR
- SIOCGIFHWADDR
- SIOCGIFFLAGS
- SIOCSIFFLAGS
- SIOCGIFNAME
- SIOCSIFNAME

- SIOCGIFINDEX
- SIOCGIFCONF

The following socket options are supported for PF_PACKET socket under the lwip_fcntl API:

- F_GETFL
- F_SETFL

1.3.3 Unsupported Features

The following features/protocols supported either partially or fully by lwIP will not be supported by Huawei LiteOS lwIP.

- IPv6 (Full Support not available in lwIP itself)
- PPPoS
- SNMP Agent (Only Private MIB Support present in lwIP).
- IP forwarding over multiple network interfaces.
- No Router based (Routing) functionalities will be supported. (Only End Device Functionalities will be supported).
- **PF_PACKET does not supports the following APIs :**
  - lwip_listen
  - lwip_accept
  - lwip_shutdown
  - lwip_getpeername
  - lwip_getsockname

  The unsupported APIs will return EOPNOTSUPP error code.

  NOTE

  Refer VPP2.0 V100R001C20 Huawei LiteOS lwIP API Reference.chm document for description of the above APIs.

  The following socket options can not be set for PF_PACKET sockets.

  The lwip_getsockopt() and lwip_setsockopt APIs do not support the following socket options:

  - SO_BROADCAST
  - SO_KEEPALIVE
  - SO_SNDTIMEO
  - SO_REUSEADDR
  - SO_REUSEPORT
  - SO_NO_CHECK
  - IP_TTL
  - IP_TOS
  - IP_HDRINCL
  - IP_MULTICAST_TTL
  - IP_MULTICAST_IF
  - IP_MULTICAST_LOOP

- IP_ADD_MEMBERSHIP
- IP_DROP_MEMBERSHIP
- TCP_NODELAY
- TCP_KEEPIDLE
- TCP_KEEPALIVE
- TCP_KEEPINTVL
- TCP_KEEPCNT
- UDPLITE_SEND_CSCOV
- UDPLITE_RECV_CSCOV
- SO_ACCEPTCONN
- SO_ERROR

The lwip_ioctl() does not support the following socket option:

- SIOCADDRT

The unsupported options set ENOPROTOOPT as error code.

1.4 Developing Applications

Light weight IP stack can run on Huawei LiteOS either using ethernet or WiFi. Application needs to configure the driver with lwIP.

1.4.1 Prerequisites

Following are the prerequisites of using Huawei LiteOS lwIP stack:

- The Huawei LiteOS lwIP stack is modified to run on Huawei LiteOS. So, it does not run on any other operating system at present.
  - Before using Huawei LiteOS lwIP, update your driver code with Huawei LiteOS lwIP related configurations. This is explained with a pseudo code in the "**Integration Steps**" section.
  - Register SSP secure functions before lwIP stack initialization. Please refer to pseudo code in "**Integration Steps**" section.

1.4.2 Dependencies

Dependencies of Huawei LiteOS lwIP are listed below:

- Huawei LiteOS lwIP depends on Huawei LiteOS system calls.
- Huawei LiteOS lwIP requires ethernet or WiFi driver module to send and receive data on physical layer.

1.4.3 Structure of Release Package

The release package contains the source code of Huawei LiteOS in the following structure:

lwIP/src - Contains all source file of Huawei LiteOS lwIP

lwIP/include - Contains all include file of Huawei LiteOS lwIP

lwIP/src/api - Contains the Netconn API, Socket API, and the tcpip thread

lwIP/src/core - Contains core implementation of DHCP, TCP, UDP, DNS, SNTP, and support code (for example, memory, netif)

lwIP/src/core/ipv4 - Contains IPv4 and ICMP implementation

lwIP/src/netif - Contains ARP, PPPoE, and ethernet driver abstraction

lwIP/src/arch - Contains OS abstraction

1.4.4 Using lwIP

Huawei LiteOS lwIP provides BSD style TCP/IP socket APIs, which can be used by an application for making socket connections. The main advantage of this stack is that the legacy application code which runs on BSD TCP/IP stack can be directly ported to this Huawei LiteOS lwIP TCP/IP stack. Huawei LiteOS lwIP supports the DHCP client feature, using which dynamic IP can be configured. Huawei LiteOS lwIP also supports the DNS client feature, using which, the application can resolve the domain name. Huawei LiteOS lwIP supports PPPoE clients, which can be connected to the Internet through a remote BAS.

1.4.5 Integration Steps

Huawei LiteOS lwIP stack can run on Huawei LiteOS either using ethernet or WiFi. Application needs to configure the driver with lwIP.

Pseudo code for driver (ethernet or WiFi) code abstraction layer for using lwIP is given below:

```
/* Global variable for lwIP Network interface should be declared */
struct netif g_netif;

/* user_memset mentioned below is the pseudocode for the */
/* MemSet function. It explains the prototype for the secure MemSet. */
/* User should implement this function based on their requirements */
int user_memset(void *pvDest,unsigned int ulDestMax, int Char, unsigned int
ulCount)
{
    memset(pvDest, Char, ulCount);
    return 0;
}

/* user_memcpy mentioned below is the pseudocode for the */
/* secure memcopy function. It explains the prototype for the secure memcopy. */
/* User should implement this function based on their requirements */
int  user_memcpy(void *pvDest,unsigned int ulDestMax, const void *Src, unsigned
int ulCount)
{
    memcpy(pvDest, Src, ulCount);
    return 0;
}

/* user_strncpy mentioned below is the pseudocode for the */
/* secure strncpy function. It explains the prototype for the secure strncpy. */
/* User should implement this function based on their requirements */
int  user_strncpy( char *pcDest, unsigned int ulDestMax, const char * pcSrc,
unsigned int ulCount )
{
    strncpy(pcDest, pcSrc, ulCount);
    return 0;
}

/* user_strncat mentioned below is the pseudocode for the */
/* secure strncat function. It explains the prototype for the secure strncat. */
/* User should implement this function based on their requirements */
int user_strncat( char *pcDest, unsigned int ulDestMax,const char *pcSrc,
unsigned int ulCount)
```

```
{
    strncat(pcDest, pcSrc, ulCount);
    return 0;
}

/* user_strcat mentioned below is the pseudocode for the */
/* secure strcat function. It explains the prototype for the secure strcat. */
/* User should implement this function based on their requirements */
int user_strcat( char *pcDest, unsigned int ulDestMax,const char *pcSrc)
{
    strcat(pcDest, pcSrc);
    return 0;
}

/* user_memmove mentioned below is the pseudocode for the */
/* secure memmove function. It explains the prototype for the secure memmove. */
/* User should implement this function based on their requirements */
int user_memmove(void *pcDest,unsigned int ulDestMax, const void *pcSrc, unsigned
int ulCount)
{
    memmove(pcDest, pcSrc, ulCount) ;
    return 0;
}

/* user_snprintf mentioned below is the pseudocode for the */
/* secure snprintf function. It explains the prototype for the secure snprintf.
*/
/* User should implement this function based on their requirements */
int user_snprintf(char* pcStrDest, unsigned int ulDestMax,
    unsigned int ulCount, const char* pszFormat, ...)
{
    int ret = 0;
    va_list arglist;

    va_start(arglist, pszFormat);
    ret = vsnprintf(pcStrDest, ulCount, pszFormat, arglist);
    va_end(arglist);
    return ret;
}

/* user_rand mentioned below is the pseudocode for the */
/* secure rand function. It explains the prototype for the secure rand. */
/* User should implement this function based on their requirements */
int user_rand(void)
{
    return rand();
}

/* user_driver_init_func mentioned below is the pseudocode for the */
/* driver init function. It explains the lwIP configuration which needs to be */
/* done along with driver init. User should implement this function based on
their driver */
void user_driver_init_func()
{
    ip_addr_t ipaddr, netmask, gw;

    /* After performing user driver init operation here */
    /* lwIP driver configuration needs to be done*/

#ifndef lwIP_WITH_DHCP_CLIENT
    IP4_ADDR(&amp;gw, 192, 168, 2, 1);
    IP4_ADDR(&amp;ipaddr, 192, 168, 2, 5);
    IP4_ADDR(&amp;netmask, 255, 255, 255, 0);
#endif

    g_netif.link_layer_type = ETHERNET_DRIVER_IF;
    g_netif.hwaddr_len = ETHARP_HWADDR_LEN;
    g_netif.drv_send = user_driver_send;
    memcpy(g_netif.hwaddr, driver_mac_address, ETHER_ADDR_LEN);
```

```
#ifndef lwIP_WITH_DHCP_CLIENT
    netifapi_netif_add(&amp;g_netif, &amp;ipaddr, &amp;netmask, &amp;gw);
#else
    netifapi_netif_add(&amp;g_netif, 0, 0, 0);
#endif
    /* lwIP configuratin ends */
}


/* user_driver_send mentioned below is the pseudocode for the */
/* driver send function. It explains the prototype for the driver send function.
*/
/* User should implement this function based on their driver */
void user_driver_send(struct netif *netif, struct pbuf *p)
{
    /* This will be the send function of the driver */
    /* It should send the data in pbuf p-&gt;payload of size p-&gt;tot_len */
}

/* user_driver_recv mentioned below is the pseudocode for the */
/* driver receive function. It explains how it should create pbuf */
/* and copy the incoming packets. User should implement this function */
/* based on their driver */
void user_driver_recv(char * data, int len)
{
    /* This should be the receive function of the user driver */
    /* Once it receives the data it should do the below */
    struct pbuf *p, *q;
    p = pbuf_alloc(PBUF_RAW, (len + ETH_PAD_SIZE), PBUF_RAM);
    if (p == NULL) {
        printf("user_driver_recv : pbuf_alloc failed\n");
        return;
    }
#if ETH_PAD_SIZE
    pbuf_header(p, -ETH_PAD_SIZE); /* drop the padding word */
#endif

    mempcy(p-&gt;payload, data, len);
#if ETH_PAD_SIZE
  pbuf_header(p, ETH_PAD_SIZE); /* reclaim the padding word */
#endif
    driverif_input(&amp;gnetif,p);
}

/* This is the init function to bring up lwIP and driver */
int init_func()
{
    STlwIPSecFuncSsp stlwIPSspCbk= {0};

    /*Register below user specific secure functions*/
    stlwIPSspCbk.pfMemset_s = user_memset;
    stlwIPSspCbk.pfMemcpy_s = user_memcpy;
    stlwIPSspCbk.pfStrNCpy_s = user_strncpy;
    stlwIPSspCbk.pfStrNCat_s = Stub_StrnCat;
    stlwIPSspCbk.pfStrCat_s = user_strcat;
    stlwIPSspCbk.pfMemMove_s = user_memmove;
    stlwIPSspCbk.pfSnprintf_s = user_snprintf;
    stlwIPSspCbk.pfRand = user_rand;

    /* Register Secure SSP callback functions*/
    if(lwIPRegSecSspCbk(&stlwIPSspCbk) != 0) {
        printf("Failed to register Secure callback functions \n");
        return -1;
    }

    /* Call lwIP tcpip_init before driver init*/
    tcpip_init(NULL, NULL);
    user_driver_init_func();
```

```
#ifndef lwIP_WITH_DHCP_CLIENT
    netifapi_dhcp_start(&amp;g_netif);
    do {
        msleep(20);
    } while (netifapi_dhcp_is_bound(&amp;g_netif) != ERR_OK);
    netifapi_netif_set_up(&amp;g_netif);
#endif
    printf("Network is up !!!\n");
}
```

1.4.6 Optimizing Huawei LiteOS lwIP

This contain the following topics:

1.4.6.1 Throughput Optimizations

The following optimizations are suggested for achieving better throughput in lwIP:

1. Configure the required number of UDP connections in MEMP_NUM_UDP_PCB. DHCP also creates one UDP connection, so consider this also while configuring this macro.
2. Configure the required number of TCP connections in MEMP_NUM_TCP_PCB.
3. Configure the required number of RAW connections in MEMP_NUM_RAW_PCB. The lwIP_ENABLE_LOS_SHELL_CMD module uses one RAW connection for the ping command.
4. Configure the required value in MEMP_NUM_NETCONN. This value is the total number of required UDP, TCP, and RAW connections.
5. If lwIP_ENABLE_LOS_SHELL_CMD is not used and RAW connection is not used separately also, then disable lwIP_RAW.
6. Similarly, UDP is not used at all then disable lwIP_UDP and if TCP is not used at all, then disable lwIP_TCP.
7. In driver receive function, if pbuf_alloc() on PBUF_RAM failed, then increase the MEM_SIZE.
8. Increase TCPIP_MBOX_SIZE and MEMP_NUM_TCPIP_MSG_INPKT values, if driverif_input() fails due to unavailability of space in TCPIP mbox for incoming packets. This can happen if the driver module is too fast in receiving upcoming packets.
9. Disable all debugging options and do not define lwIP_DEBUG.
10. If the word size of the architecture is 4, keep ETH_PAD_SIZE as 2.

1.4.6.2 Memory Optimizations

To achieve less footprint in lwIP, some of the optimizations are suggested below:

1. Use PBUF_POOL with pbuf_alloc() in driver receive function. And also configure required amount of pbuf in PBUF_POOL_SIZE. And also configure the PBUF_POOL_BUFSIZE based on the MTU.
2. Configure MEMP_NUM_TCP_PCB, MEMP_NUM_UDP_PCB, MEMP_NUM_RAW_PCB and MEMP_NUM_NETCONN, based on the required amount of TCP, UDP and RAW connections.
3. Reduce the TCPIP_MBOX_SIZE and MEMP_NUM_TCPIP_MSG_INPKT values, by considering the amount of traffic comes from peer.
4. Disable all debugging options and do not define lwIP_DEBUG.

5. Disable ETHARP_TRUST_IP_MAC. This updates the ARP table based on all incoming packets. If its disabled, only required entity's entry will get updated by doing specific ARP query.

1.4.6.3 Customization

The new macros defined in Huawei LiteOS lwIP to suit our usage are listed below:

- Generally, all lwIP APIs are with the prefix of "lwIP_" (for example lwIP_connect, lwIP_bind and so on). Also, there is macro definition to replace BSD style API names (connect, bind etc) to lwIP style API names (lwIP_connect, lwIP_bind and so on). This helps to change BSD style API names to lwIP style API names in application code. But, if user wants to make lwIP to expose BSD style API (without lwIP_ prefix), then user need to enable lwIP_BSD_API macro. Also, this macro defines struct sockaddr similar to BSD.
- lwIP definition of the structure sockaddr_in is different from BSD. It has one additional member length. If user wants BSD style definition for structure sockaddr_in, then macro lwIP_BSD_SOCKADDR_IN needs to be defined.
- For enabling DHCP sever, user need to enable the macro lwIP_DHCPS. If lwIP_DHCPS is enabled, then lwIP_DHCP also should be enabled.
- Generally, lwIP creates socket file descriptor from 0. If user wants to change this file descriptor starting number, then it needs to be configured in macro lwIP_SOCKET_START_NUM.
- In sockets.h, lwIP defines SO_ macros (for example SO_DEBUG, SO_REUSEADDR and so on) as similar to BSD. But, if user wants to keep ARM BSD linux based macro values, then macro lwIP_ENABLE_ARM_BASED_SO_MACROS needs to be enabled.
- In lwIP, inet_addr is a macro which redirects to the lwIP specific function. But, if user wants inet_addr as an API, then lwIP_INET_ADDR_FUNC needs to be defined.
- In lwIP, inet_ntoa is a macro which redirects to the lwIP specific function. But, if user wants inet_ntoa as an API, then lwIP_INET_NTOA_FUNC needs to be defined.
- Generally, DHCP server sends the offer message as broadcast or unicast based on the flag set by client in its discover message. But, if user wants to always broadcast the offer message, then the macro lwIP_DHCPS_DISCOVER_BROADCAST needs to be enabled.

1.4.6.4 Huawei LiteOS lwIP Macros

This section lists the Huawei LiteOS lwIP macros. Configure these macros based on usage.

Default values for lwIP macros are defined in the opt.h and lwIPopts.h header files. Macros defined in lwIPopts.h overwrite the definition in opt.h.

Table 1-1 List of Macros

Macros	Description
LWIP_AUTOIP	This is macro To Enable/Disable AUTOIP Module

Macros	Description
MEM_SIZE	lwIP maintains a heap memory management (mem_malloc and mem_free), all dynamic memory allocation is handled by this module. There will not be any call made to system malloc() or free(). This macro is to define the size for the Heap memory management in lwIP.
MEM_LIBC_MALLOC	If this macro is enabled, then all dynamic memory allocation call will go to system malloc() and free(). And also it will disable the heap memory management module code in lwIP.
MEMP_MEM_MALLOC	lwIP maintains a pool memory management (memp_malloc and memp_free) for frequently used structures.
MEM_ALIGNMENT	Memory alignment in bytes needs to be configured based on the architecture. For example 4 should be configured if its 32 bit architecture.
MEMP_NUM_TCP_PCB	This macro is used to configure the required number of simultaneous TCP connections.
MEMP_NUM_UDP_PCB	This macro is used to configure the required number of simultaneous UDP connections. While configuring this user should consider the internal modules of lwIP like DNS, DHCP which creates UDP connection for its communication.
MEMP_NUM_RAW_PCB	This macro is used to configure the required number of simultaneous RAW connections.
MEMP_NUM_NETCONN	This macro is used to configure the total number of TCP, UDP and Raw connections. This should be sum of MEMP_NUM_TCP_PCB, MEMP_NUM_UDP_PCB and MEMP_NUM_RAW_PCB.
MEMP_NUM_TCP_PCB_LISTEN	This macro is used to configure the required number of simultaneous listening TCP connections.
MEMP_NUM_REASSDATA	This macro is used to configure the number of IP packets simultaneously queued for reassembly (whole packets, not fragments!)
MEMP_NUM_FRAG_PBUF	This macro is used to configure the number of IP fragments simultaneously sent (fragments, not whole packets!)
ARP_TABLE_SIZE	This macro is used to configure the ARP cache table size.

Macros	Description
ARP_QUEUEING	If this macro is enabled, then multiple outgoing packets are getting queued during ARP resolution. If this macro is not enabled, then it will only keep the recent packet which is sent by upper layer for each destination address.
MEMP_NUM_ARP_QUEUE	This macro is used to configure the number of simulateously queued outgoing packets (pbufs) that are waiting for an ARP response to resolve their destination address. This macro is applicable only if ARP_QUEUEING is enabled.
ETHARP_TRUST_IP_MAC	If this macro is enabled, then source IP address and source MAC address from all incoming packets get updated in ARP cache table. If this macro is disabled, then entry in ARP cache is updated only by its own ARP query.
ETH_PAD_SIZE	This macro is used to configure the 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 addresses in the IP header will not be aligned. For example, in a 32 bit architecture setting this to 2 can make IP header as 4 byte aligned and also it can speed up.
ETHARP_SUPPORT_STATIC_ENTRIES	This macro is used to enable the support for updating ARP cache table statically using the etharp_add_static_entry() and etharp_remove_static_entry() API from application.
IP_REASSEMBLY	This macro is used to enable the support for reassemble the fragmented IP packets.
IP_FRAG	This macro is used to enable the support for IP level fragmentation on all outgoing packets.
IP_REASS_MAXAGE	This macro is used to configure the timeout for fragmented incoming IP packets. lwIP maintains the fragmented packets for IP_REASS_MAXAGE seconds, if its not able to reassemble the packet within that time then it will drop those fragmented packets.
IP_REASS_MAX_PBUFS	This macro is used to configure the maximum amount of fragmentation allowed for an incoming IP packet.
IP_FRAG_USES_STATIC_BUF	If this macro is enabled, static buffer is used for IP fragmentation. Otherwise dynamic memory is allocated.

Macros	Description
IP_FRAG_MAX_MTU	Maximum MTU size needs to be configured in this macro.
IP_DEFAULT_TTL	Default value for Time-To-Live used by transport layers.
LWIP_RANDOMIZE_INITIAL_LOCAL_PORTS	This macro is used to enable the support for randomize the local port for the first local TCP/UDP pcb (default==0). This can prevent creating predictable port numbers after booting a device. For this LWIP_RAND macro also needs to be configured with a strong random number generator function. This is because lwIP depends on system for random genertor function.
LWIP_RAND	LWIP_RAND macro to be configured with a strong random number generator function. This is because lwIP depends on system for random generator function. Random numbers are used for generating random client ports in DNS and user TCP/UDP connections. And also its used for creating transaction IDs in DHCP and DNS and for the ISS value in TCP.
LWIP_ICMP	To enable ICMP module.
ICMP_TTL	To configure TTL value for ICMP messages.
LWIP_RAW	To enable RAW socket support in lwIP.
RAW_TTL	To configure TTL value for RAW socket messages.
LWIP_UDP	To enable the UDP connection support in lwIP.
UDP_TTL	To configure TTL value for UDP socket messages.
LWIP_TCP	To enable the TCP connection support in lwIP.
TCP_TTL	To configure the TTL value for TCP socket messages.
TCP_WND	To configure the TCP window size.
TCP_MAXRTX	To configure the maximum retransmission of TCP data segments.
TCP_SYNMAXRTX	To configure the maximum retransmission of TCP SYN segments.
TCP_FW1MAXRTX	To configure the maximum retransmission of TCP data segments in FIN_WAIT_1 state.

Macros	Description
TCP_QUEUE_OOSEQ	This is to enable support for queuing segments that arrive out of order. Define it to 0 if your device is low on memory.
TCP_MSS	To configure Maximum Segment Size for TCP connections.
TCP_SND_BUF	To configure Send buffer size of TCP.
TCP_SND_QUEUELEN	To configure send queue length of TCP.
TCP_OOSEQ_MAX_BYTES	To configure the maximum number of bytes queued on ooseq per pcb. Default is 0 (no limit). Only valid for TCP_QUEUE_OOSEQ==0.
TCP_OOSEQ_MAX_PBUFS	To configure the maximum number of pbufs queued on ooseq per pcb. Default is 0 (no limit). Only valid for TCP_QUEUE_OOSEQ==0.
TCP_LISTEN_BACKLOG	To enable the backlog support in TCP listen.
LWIP_DHCP	To enable DHCP client module.
LWIP_DHCPS	To enable DHCP server module.
LWIP_IGMP	To enable IGMP module.
LWIP_SNTP	To enable SNTP client module.
SNTP_MAX_REQUEST_RETRANSMIT	This macro is used to configure maximum SNTP client retries to SNTP server, default is set to 3 that is retransmission will happen thrice after first original transmission.
LWIP_DNS	To enable DNS client module.
DNS_TABLE_SIZE	To configure the size of the DNS cache table size.
DNS_MAX_NAME_LENGTH	To configure the supported maximum length for domain name. This is configured to 255 as per DNS RFC. Its recommeded to not modify this.
DNS_MAX_SERVERS	To configure the number of DNS servers.
DNS_MAX_LABLE_LENGTH	To configure the maximum length of each sub domain name in a domain name. This is configured to 63 as per DNS RFC. Its recommended to not modify this.
DNS_MAX_IPADDR	To configure the maximum IP address that DNS client can cache.
PBUF_LINK_HLEN	To configure the number of bytes that should be allocated for link level header. This should include the actual length plus ETH_PAD_SIZE.

Macros	Description
LWIP_NETIF_API	To enable the thread safe netif module (netiapi_*).
TCPIP_THREAD_NAME	To configure a name for TCP IP thread.
DEFAULT_RAW_RECVMBOX_SIZE	Each raw connection maintains a queue for incoming packets, to buffer the incoming packets till a receive call is made from application layer. This macro is used to configure the size of the receive message box queue.
DEFAULT_UDP_RECVMBOX_SIZE	Each UDP connection maintains a queue for incoming packets, to buffer the incoming packets till a receive call is made from application layer. This macro is used to configure the size of the receive message box queue.
DEFAULT_TCP_RECVMBOX_SIZE	Each TCP connection maintains a queue for incoming packets, to buffer the incoming packets till a receive call is made from application layer. This macro is used to configure the size of the receive message box queue.
DEFAULT_ACCEPTMBOX_SIZE	This macro is used to configure the size of the message box queue to maintain incoming TCP connections.
TCPIP_MBOX_SIZE	This macro is used to configure the message box queue of the TCP IP thread. This queue maintains all the operation request from application thread and driver thread.
LWIP_TCPIP_TIMEOUT	This macro enables the feature for running a user defined timer handler function on lwIP tcpip thread.
LWIP_SOCKET_START_NUM	This macro is used to configure the start number for the socket file descriptor created by lwIP.
LWIP_BSD_API	This macro helps to expose the socket API similar to linux API name.
LWIP_BSD_SOCKADDR_IN	This exposes the BDS style socketaddr_in structure.
LWIP_ENABLE_ARM_BASED_SO_ MACROS	This macro defines the ARM values for all SO_* macros. By default BSD values are defined in all SO_* macros.
LWIP_COMPAT_SOCKETS	This creates a linux BSD style name macro for all socket APIs.
LWIP_POSIX_SOCKETS_IO_NAME S	This creates a POSIX style macro for read, write, fcntl and close.

Macros	Description
LWIP_STATS	This measures the statistics of all ongoing connections.
LWIP_SACK	This macro is used to enabled /disable SACK fucntionality in LWIP stack. This macro needs to be enabled for enabling both sender and receiver SACK functionality.
LWIP_SACK_DATA_SEG_PIGGYBACK	This macro is used to enable sending of SACK Options if any in the segments with ACK flag set and carrying data too. If this macro is disabled, then SACK Options will be sent only in empty ACK segments and not in segments carrying data as well as ACK in case of bidirectional data transfer
LWIP_WND_SCALE	This macro is used to enable/disable the Window Scaling functionality in LWIP stack.
TCP_WND_MIN	If window scaling is enabled then this minimum receive window also should be configured. So that if peer is not supporting window scaling option, then this minimum receive window will be considered .This value should not be greater than 0XFFFF and this value should not be more than TCP_WND.
MEM_PBUF_RAM_SIZE_LIMIT	This is to enable/disable limiting of RAM memory size for pbuf allocations for PBUF_RAM type. This limits the operating system memory to be allocated for PBUF_RAM and not for allocation of internal buf memory. This is applicable only if MEM_LIBC_MALLOC is enabled
LWIP_PBUF_STATS	This is used to enable/disable the debug prints for PBUF_RAM memory allocation statistics when MEM_PBUF_RAM_SIZE_LIMIT is enabled.
PBUF_RAM_SIZE_MIN	The minimum RAM memory required to be set through the API pbuf_ram_size_set.
DRIVER_STATUS_CHECK	This macro is for enabling the driver send buffer status intimation to the stack using netifapi_stop_queue and netifapi_wake_queue APIs.

Macros	Description
DRIVER_WAKEUP_INTERVAL	When the netif driver status is changed to Busy and does not change back to Ready state before the expiry of this timer, then all TCP connections linked to this netif are flushed. Value is 120000 msec by default.
PBUF_LINK_CHKSUM_LEN	This macro gives the length of the link layer checksum which will be filled by ethernet driver.
LWIP_DEV_DEBUG	This macro is for developer debugging only. This must not be enabled in customer environment.
LWIP_PPPOE	This macro is used to enable or disable the PPPoE module.
PPP_SUPPORT	This macro controls whether to support PPP.
PPPOE_SUPPORT	This macro controls whether to support PPPoE
CHAP_SUPPORT	This macro controls whether to support Challenge-Handshake Authentication Protocol (CHAP) in PPP.

1.4.7 Sample Codes

Some of the sample code of lwIP are listed below, which explains the usage of lwIP. It has a pseudo code which explains the changes which need to be done on the driver (ethernet or WiFi) module. This sample code can be compilable with lwIP and Huawei LiteOS on Hisilicon board cross compiler.

1.4.7.1 Sample Code for UDP

```
#include "lwIP/sockets.h"
#include "lwIP/err.h"
#include "lwIP/ip.h"
#include "lwIP/tcpip.h"
#include "lwIP/netif.h"
#include "lwIP/dhcp.h"

#define STACK_IP "192.168.2.5"
#define STACK_PORT 2277
#define PEER_PORT 3377
#define PEER_IP "192.168.2.2"

#define MSG "Hi, I am lwIP"

#define BUF_SIZE (1024 * 8)
u8_t g_buf[BUF_SIZE+1] = {0}

/* Global variable for lwIP Network interface */
struct netif g_netif;

/* user_driver_init_func mentioned below is the pseudocode for the */
/* driver init function. Its explains the lwIP configuration which needs to be */
/* done along with driver init. User should implement this function based on
their driver */
```

```
void user_driver_init_func()
{
    ip_addr_t ipaddr, netmask, gw;

    /* After performing user driver init operation */
    /* lwIP driver configuration needs to be done*/

    /* lwIP configuration starts */
    IP4_ADDR(&gw, 192, 168, 2, 1);
    IP4_ADDR(&ipaddr, 192, 168, 2, 5);
    IP4_ADDR(&netmask, 255, 255, 255, 0);

    g_netif.link_layer_type = ETHERNET_DRIVER_IF;
    g_netif.hwaddr_len = ETHARP_HWADDR_LEN;
    g_netif.drv_send = user_driver_send;
    memcpy(g_netif.hwaddr, driver_mac_address, ETHER_ADDR_LEN);

    netifapi_netif_add(&g_netif, &ipaddr, &netmask, &gw);
    netifapi_netif_set_default(&g_netif);
    /* lwIP configuratin ends */
}

/* user_driver_send mentioned below is the pseudocode for the */
/* driver send function. It explains the prototype for the driver send function.
*/
/* User should implement this function based on their driver */

void user_driver_send(struct netif *netif, struct pbuf *p)
{
    /* This will be the send function of the driver */
    /* It should send the data in pbuf p->payload of size p->tot_len */
}

/* user_driver_recv mentioned below is the pseudocode for the */
/* driver receive function. It explains how it should create pbuf */
/* and copy the incoming packets. User should implement this function */
/* based on their driver */

void user_driver_recv(char * data, int len)
{
    /* This should be the receive function of the user driver */
    /* Once it receives the data it should do the below */
    struct pbuf *p, *q;

    p = pbuf_alloc(PBUF_RAW, (len + ETH_PAD_SIZE), PBUF_RAM);

    if (p == NULL) {
        printf("user_driver_recv : pbuf_alloc failed\n");
        return;
    }

#if ETH_PAD_SIZE
    pbuf_header(p, -ETH_PAD_SIZE); /* drop the padding word */
#endif

    mempcy(p->payload, data, len);

#if ETH_PAD_SIZE
  pbuf_header(p, ETH_PAD_SIZE); /* reclaim the padding word */
#endif

    driverif_input(&gnetif,p);
}

int sample_init_func()
{
    /* Call lwIP tcpip_init before driver init*/
    tcpip_init(NULL, NULL);
    user_driver_init_func();
}

int sample_udp()
{
    s32_t sfd;
    struct sockaddr_in srv_addr = {0};
    struct sockaddr_in cln_addr = {0};
    socklen_t cln_addr_len = sizeof(cln_addr);
    s32_t ret = 0, i = 0;

    /* socket creation */
    printf("going to call lwIP_socket\n");
    sfd = lwIP_socket(AF_INET,SOCK_DGRAM,0);
    if (sfd == -1)
    {
        printf("lwIP_socket failed, return is %d\n", sfd);
        goto FAILURE;
    }

    printf("lwIP_socket succeeded\n");

    srv_addr.sin_family = AF_INET;
    srv_addr.sin_addr.s_addr=inet_addr(STACK_IP);
    srv_addr.sin_port=htons(STACK_PORT);

    printf("going to call lwIP_bind\n");
    ret = lwIP_bind(sfd,(struct sockaddr *)&srv_addr,sizeof(srv_addr));
    if (ret != 0)
    {
        printf("lwIP_bind failed, return is %d\n", ret);
        goto FAILURE;
    }

    printf("lwIP_bind succeeded\n");
    /* socket creation */

    /* send */
    cln_addr.sin_family = AF_INET;
    cln_addr.sin_addr.s_addr=inet_addr(PEER_IP);
    cln_addr.sin_port=htons(PEER_PORT);

    printf("calling lwIP_sendto...\n");
    memset(g_buf, 0, BUF_SIZE);
    strcpy(g_buf, MSG);

    ret = lwIP_sendto(sfd, g_buf, strlen(MSG), 0, (struct sockaddr *)&cln_addr,
(socklen_t)sizeof(cln_addr));
    if (ret <= 0)
    {
        printf("lwIP_sendto failed, return is %d\n", ret);
        goto FAILURE;
    }
    printf("lwIP_sendto succeeded, return is %d\n", ret);
    /* send */

    /* recv */
    printf("going to call lwIP_recvfrom\n");
    memset(g_buf, 0, BUF_SIZE);
    ret = lwIP_recvfrom(sfd, g_buf, sizeof(g_buf), 0, (struct sockaddr
*)&cln_addr, &cln_addr_len);
    if (ret <= 0)
    {
        printf("lwIP_recvfrom failed, return is %d\n", ret);
        goto FAILURE;
    }

    printf("lwIP_recvfrom succeeded, return is %d\n", ret);
    printf("received msg is : %s\n", g_buf);
    printf("client ip %x, port %d\n", cln_addr.sin_addr.s_addr, 
cln_addr.sin_port);
    /* recv */

    lwIP_close(sfd);
    return 0;

FAILURE:
    printf("failed, errno is %d\n", errno);
    lwIP_close(sfd);
    return -1;
}


int main()
{
    int ret;
    ret = sample_init_func();
    if (ret != 0)
    {
        printf("init failed\n");
        exit(0);
    }

    ret = sample_udp()
    if (ret != 0)
    {
        printf("Sample Test case failed\n");
        exit(0);
    }

    return 0;
}
```

1.4.7.2 Sample Code for TCP Client

```
#include "lwIP/sockets.h"
#include "lwIP/err.h"
#include "lwIP/ip.h"
#include "lwIP/tcpip.h"
#include "lwIP/netif.h"
#include "lwIP/dhcp.h"

#define STACK_IP "192.168.2.5"
#define STACK_PORT 2277
#define PEER_PORT 3377
#define PEER_IP "192.168.2.2"

#define MSG "Hi, I am lwIP"

#define BUF_SIZE (1024 * 8)
u8_t g_buf[BUF_SIZE+1] = {0}

/* Global variable for lwIP Network interface */
struct netif g_netif;

/* user_driver_init_func mentioned below is the pseudocode for the */
/* driver init function. Its explains the lwIP configuration which needs to be */
/* done along with driver init. User should implement this function based on 
their driver */

void user_driver_init_func()
{
    ip_addr_t ipaddr, netmask, gw;

    /* After performing user driver init operation */
    /* lwIP driver configuration needs to be done*/

    /* lwIP configuration starts */
    IP4_ADDR(&gw, 192, 168, 2, 1);
    IP4_ADDR(&ipaddr, 192, 168, 2, 5);
    IP4_ADDR(&netmask, 255, 255, 255, 0);

    g_netif.link_layer_type = ETHERNET_DRIVER_IF;
    g_netif.hwaddr_len = ETHARP_HWADDR_LEN;
    g_netif.drv_send = user_driver_send;
    memcpy(g_netif.hwaddr, driver_mac_address, ETHER_ADDR_LEN);

    netifapi_netif_add(&g_netif, &ipaddr, &netmask, &gw);
    netifapi_netif_set_default(&g_netif);
    /* lwIP configuratin ends */
}

/* user_driver_send mentioned below is the pseudocode for the */
/* driver send function. It explains the prototype for the driver send function.
*/
/* User should implement this function based on their driver */

void user_driver_send(struct netif *netif, struct pbuf *p)
{
    /* This will be the send function of the driver */
    /* It should send the data in pbuf p->payload of size p->tot_len */
}

/* user_driver_recv mentioned below is the pseudocode for the */
/* driver receive function. It explains how it should create pbuf */
/* and copy the incoming packets. User should implement this function */
/* based on their driver */

void user_driver_recv(char * data, int len)
{
    /* This should be the receive function of the user driver */
    /* Once it receives the data it should do the below */
    struct pbuf *p, *q;

    p = pbuf_alloc(PBUF_RAW, (len + ETH_PAD_SIZE), PBUF_RAM);

    if (p == NULL) {
        printf("user_driver_recv : pbuf_alloc failed\n");
        return;
    }

#if ETH_PAD_SIZE
    pbuf_header(p, -ETH_PAD_SIZE); /* drop the padding word */
#endif

    mempcy(p->payload, data, len);

#if ETH_PAD_SIZE
  pbuf_header(p, ETH_PAD_SIZE); /* reclaim the padding word */
#endif

    driverif_input(&gnetif,p);
}

int sample_init_func()
{
    /* Call lwIP tcpip_init before driver init*/
    tcpip_init(NULL, NULL);

    user_driver_init_func();
}

int sample_tcp_client()
{
    s32_t sfd = -1;
    struct sockaddr_in srv_addr = {0};
    struct sockaddr_in cln_addr = {0};
    socklen_t cln_addr_len = sizeof(cln_addr);
    s32_t ret = 0, i = 0;

    /* tcp client connection */
    printf("going to call lwIP_socket\n");
    sfd = lwIP_socket(AF_INET,SOCK_STREAM,0);
    if (sfd == -1)
    {
        printf("lwIP_socket failed, return is %d\n", sfd);
        goto FAILURE;
    }

    printf("lwIP_socket succeeded, sfd %d\n", sfd);

    srv_addr.sin_family = AF_INET;
    srv_addr.sin_addr.s_addr=inet_addr(PEER_IP);
    srv_addr.sin_port=htons(PEER_PORT);

    printf("going to call lwIP_connect\n");
    ret = lwIP_connect(sfd, (struct sockaddr *)&srv_addr, sizeof(srv_addr));
    if (ret != 0)
    {
        printf("lwIP_connect failed, return is %d\n", ret);
        goto FAILURE;
    }
    printf("lwIP_connec succeeded, return is %d\n", ret);
    /* tcp client connection */

    /* send */
    memset(g_buf, 0, BUF_SIZE);
    strcpy(g_buf, MSG);

    printf("calling send...\n");
    ret = lwIP_send(sfd, g_buf, sizeof(MSG), 0);

    if (ret <= 0)
    {
        printf("lwIP_send failed, return is %d, i is %d\n", ret, i);
        goto FAILURE;
    }

    printf("send finished ret is %d\n", ret);
    /* send */

    /* recv */
    memset(g_buf, 0, BUF_SIZE);
    printf("going to call recv\n");
    ret = lwIP_recv(sfd, g_buf, sizeof(g_buf), 0);
    if (ret <= 0)
    {
        printf("lwIP_recv failed, return is %d\n", ret);
        goto FAILURE;
    }

    printf("lwIP_recv succeeded, return is %d\n", ret);
    printf("received msg is : %s\n", g_buf);
    /* recv */

    lwIP_close(sfd);
    return 0;

FAILURE:
    lwIP_close(sfd);
    printf("errno is %d\n", errno);
    return -1;
}

int main()
{
    int ret;
    ret = sample_init_func();
    if (ret != 0)
    {
        printf("init failed\n");
        exit(0);
    }

    ret = sample_tcp_client()
    if (ret != 0)
    {
        printf("Sample Test case failed\n");
        exit(0);
    }

    return 0;
}
```

1.4.7.3 Sample Code for TCP Server

```
#include "lwIP/sockets.h"
#include "lwIP/err.h"
#include "lwIP/ip.h"
#include "lwIP/tcpip.h"
#include "lwIP/netif.h"
#include "lwIP/dhcp.h"

#define STACK_IP "192.168.2.5"
#define STACK_PORT 2277
#define PEER_PORT 3377
#define PEER_IP "192.168.2.2"

#define MSG "Hi, I am lwIP"

#define BUF_SIZE (1024 * 8)
u8_t g_buf[BUF_SIZE+1] = {0}

/* Global variable for lwIP Network interface */
struct netif g_netif;

/* user_driver_init_func mentioned below is the pseudocode for the */
/* driver init function. Its explains the lwIP configuration which needs to be */
/* done along with driver init. User should implement this function based on
their driver */

void user_driver_init_func()
{
    ip_addr_t ipaddr, netmask, gw;

    /* After performing user driver init operation */
    /* lwIP driver configuration needs to be done*/

    /* lwIP configuration starts */
    IP4_ADDR(&gw, 192, 168, 2, 1);
    IP4_ADDR(&ipaddr, 192, 168, 2, 5);
    IP4_ADDR(&netmask, 255, 255, 255, 0);

    g_netif.link_layer_type = ETHERNET_DRIVER_IF;
    g_netif.hwaddr_len = ETHARP_HWADDR_LEN;
    g_netif.drv_send = user_driver_send;
    memcpy(g_netif.hwaddr, driver_mac_address, ETHER_ADDR_LEN);

    netifapi_netif_add(&g_netif, &ipaddr, &netmask, &gw);
    netifapi_netif_set_default(&g_netif);
    /* lwIP configuratin ends */
}

/* user_driver_send mentioned below is the pseudocode for the */
/* driver send function. It explains the prototype for the driver send function.
```

```
*/
/* User should implement this function based on their driver */

void user_driver_send(struct netif *netif, struct pbuf *p)
{
    /* This will be the send function of the driver */
    /* It should send the data in pbuf p->payload of size p->tot_len */
}

/* user_driver_recv mentioned below is the pseudocode for the */
/* driver receive function. It explains how it should create pbuf */
/* and copy the incoming packets. User should implement this function */
/* based on their driver */

void user_driver_recv(char * data, int len)
{
    /* This should be the receive function of the user driver */
    /* Once it receives the data it should do the below */
    struct pbuf *p, *q;

    p = pbuf_alloc(PBUF_RAW, (len + ETH_PAD_SIZE), PBUF_RAM);

    if (p == NULL) {
        printf("user_driver_recv : pbuf_alloc failed\n");
        return;
    }

#if ETH_PAD_SIZE
    pbuf_header(p, -ETH_PAD_SIZE); /* drop the padding word */
#endif

    mempcy(p->payload, data, len);

#if ETH_PAD_SIZE
  pbuf_header(p, ETH_PAD_SIZE); /* reclaim the padding word */
#endif

    driverif_input(&gnetif,p);
}


int sample_init_func()
{
    /* Call lwIP tcpip_init before driver init*/
    tcpip_init(NULL, NULL);

    user_driver_init_func();
}

int sample_tcp_server()
{
    s32_t sfd = -1, lsfd = -1;
    struct sockaddr_in srv_addr = {0};
    struct sockaddr_in cln_addr = {0};
    socklen_t cln_addr_len = sizeof(cln_addr);
    s32_t ret = 0, i = 0;

    /* tcp server */
    printf("going to call lwIP_socket\n");
    lsfd = lwIP_socket(AF_INET,SOCK_STREAM,0);
    if (lsfd == -1)
    {
        printf("lwIP_socket failed, return is %d\n", lsfd);
        goto FAILURE;
    }

    printf("lwIP_socket succeeded\n");

    srv_addr.sin_family = AF_INET;
    srv_addr.sin_addr.s_addr=inet_addr(STACK_IP);
    srv_addr.sin_port=htons(STACK_PORT);

    ret = lwIP_bind(lsfd, (struct sockaddr *)&srv_addr,sizeof(srv_addr));
    if (ret != 0)
    {
        printf("lwIP_bind failed, return is %d\n", ret);
        goto FAILURE;
    }

    ret =  lwIP_listen(lsfd, 0);
    if (ret != 0)
    {
        printf("lwIP_listen failed, return is %d\n", ret);
        goto FAILURE;
    }

    printf("lwIP_listen succeeded, return is %d\n", ret);

    printf("going to call lwIP_accept\n");
    sfd = lwIP_accept(lsfd, (struct sockaddr *)&cln_addr, &cln_addr_len);
    if (sfd < 0)
    {
        printf("lwIP_accept failed, return is %d\n", sfd);
    }
    printf("lwIP_accept succeeded, return is %d\n", sfd);
    /* tcp server */

    /* send */
    memset(g_buf, 0, BUF_SIZE);
    strcpy(g_buf, MSG);

    printf("calling send...\n");
    ret = lwIP_send(sfd, g_buf, sizeof(MSG), 0);

    if (ret <= 0)
    {
        printf("lwIP_send failed, return is %d, i is %d\n", ret, i);
        goto FAILURE;
    }

    printf("send finished ret is %d\n", ret);
    /* send */

    /* recv */
    memset(g_buf, 0, BUF_SIZE);
    printf("going to call recv\n");
    ret = lwIP_recv(sfd, g_buf, sizeof(g_buf), 0);
    if (ret <= 0)
    {
        printf("lwIP_recv failed, return is %d\n", ret);
        goto FAILURE;
    }

    printf("lwIP_recv succeeded, return is %d\n", ret);
    printf("received msg is : %s\n", g_buf);
    /* recv */

    lwIP_close(sfd);
    lwIP_close(lsfd);
    return 0;

FAILURE:
    lwIP_close(sfd);
    lwIP_close(lsfd);
    printf("errno is %d\n", errno);
    return -1;
}
```

```
int main()
{
    int ret;
    ret = sample_init_func();
    if (ret != 0)
    {
        printf("init failed\n");
        exit(0);
    }

    ret = sample_tcp_server()
    if (ret != 0)
    {
        printf("Sample Test case failed\n");
        exit(0);
    }

    return 0;
}
```

1.4.7.4 Sample Code for DNS

```
#include "lwIP/opt.h"
#include "lwIP/sockets.h"
#include "lwIP/netdb.h"
#include "lwIP/err.h"
#include "lwIP/inet.h"

void dns_call_with_unsafe_api()
{
    struct hostent *result;
    int i = 0;
    ip_addr_t *addr;
    char addrString[20] = {0};
    char *hostname;
    char *dns_server_ip;
    ip_addr_t dns_server_ipaddr;

    hostname = "www.huawei.com";

    dns_server_ip = "192.168.0.2";
    inet_aton(dns_server_ip, &dns_server_ipaddr);
    lwIP_dns_setserver(0, &dns_server_ipaddr);

    result = lwIP_gethostbyname(hostname);

    if (result)
    {
        while (1)
        {
            addr = *(((ip_addr_t **)result->h_addr_list) + i);
            if (addr == NULL)
            {
                break;
            }

            inet_ntoa_r(*addr, addrString, 20);
            printf("dns call for %s, returns %s\n", hostname, addrString);
            i++;
        }
    }
    else
    {
        printf("dns call failed\n");
    }
}

void dns_call_with_safe_api()
{
    int i = 0;
    ip_addr_t *addr;
    char addrString[20] = {0};
    char *buf;
    int buflen;
    char *hostname;
    char *dns_server_ip;
    ip_addr_t dns_server_ipaddr;
    struct hostent ret;
    struct hostent *result = NULL;
    int h_errnop;

    hostname = "www.huawei.com";

    dns_server_ip = "192.168.0.2";
    inet_aton(dns_server_ip, &dns_server_ipaddr);
    lwIP_dns_setserver(0, &dns_server_ipaddr);

    buflen = sizeof(struct gethostbyname_r_helper) + strlen(hostname) +
MEM_ALIGNMENT;
    buf = malloc(buflen);

    lwIP_gethostbyname_r(hostname, &ret, buf, buflen, &result, &h_errnop);

    if (result)
    {
        while (1)
        {
            addr = *(((ip_addr_t **)result->h_addr_list) + i);
            if (addr == NULL)
            {
                break;
            }

            inet_ntoa_r(*addr, addrString, 20);
            printf("dns call for %s, returns %s\n", hostname, addrString);
            i++;
        }
    }
    else
    {
        printf("dns call failed\n");
    }
    free(buf);
}

/* To get the dns server address configured in lwIP */
/* Application can call lwIP_dns_getserver() API and then decide whether to
change it */
/* If application needs to change it then, it needs */
void display_dns_server_address()
{
    int i;
    ip_addr_t addr;
    int ret;
    char addrString[20] = {0};

    for (i = 0; i < DNS_MAX_SERVERS; i++)
    {
        ret = lwIP_dns_getserver(i, &addr);
        if (ret != ERR_OK) {
            printf("lwIP_dns_getserver failed\n");
            return;
        }

        memset(addrString, 0, sizeof(addrString));
        inet_ntoa_r(addr, addrString, 20);
        printf("dns server address configured at index %d, is %s\n", i,
addrString);
```

```
    }

    return;
}

int main()
{
    /* after doing lwIP init, driver init and netifapi_netif_add */
    display_dns_server_address();
    dns_call_with_unsafe_api();
    dns_call_with_safe_api();

    return 0;
}
```

1.4.7.5 Sample Code for SNTP

```
#include "lwIP/opt.h"
#include "lwIP/sntp.h"

/*
 * Compile time configuration for SNTP
 * 1) Configure SNTP server address
*/

int start_sntp()
{
    int ret;
    /* If NULL is passed to lwIP_sntp_start, then sntp server configured
     * in macro SNTP_SERVER_ADDRESS will be used by this API. If application
     * wants to do this with different SNTP server means, it can be passed as
argument
     * to below API instead of NULL. */
    ret = lwIP_sntp_start(NULL);

    if (ret == ERR_OK) {
        printf("sntp started successfully\n");
    } else {
        printf("sntp start failed\n");
    }

    /* After doing this it will preiodically sends SNTP request message
     * for every 1 hour (SNTP_UPDATE_DELAY) and keeps updating system time
     * by calling the function configured in macro SNTP_SET_SYSTEM_TIME(t) */

    /* after startding SNTP, application code can continue its other task
     * once application decides to stop the application need to call
     * the API lwIP_sntp_stop() */
}

int main()
{
    /* after doing lwIP init, driver init and netifapi_netif_add */
    start_sntp();
    return 0;
}
```

1.4.7.6 Sample Code for DHCP Client

```
#include "lwIP/opt.h"
#include "lwIP/netifapi.h"
#include "lwIP/inet.h"
#include "lwIP/netif.h"

struct netif g_netif;

int dhcp_client_start(struct netif *pnetif)
{
    int ret;
    char addrString[20] = {0};

    /* Calling netifapi_dhcp_start() will start initiating DHCP configuration
     * process by sending DHCP messages */
    ret = netifapi_dhcp_start(pnetif);

    if (ret == ERR_OK) {
        printf("dhcp client started successfully\n");
    } else {
        printf("dhcp client start failed\n");
    }

    /* After doing this it will get the IP and update to netif, once it finishes
     * the process with DHCP server. Application need to call
netifapi_dhcp_is_bound()
     * API to check whether DHCP process is finished or not */

    do {
        sleep(1); /* sleep for sometime, like 1 sec */
        ret = netifapi_dhcp_is_bound(pnetif);
    } while(ret != ERR_OK);

    memset(addrString, 0, sizeof(addrString));
    inet_ntoa_r(pnetif->ip_addr, addrString, 20);
    printf("ipaddr %s\n", addrString);
    memset(addrString, 0, sizeof(addrString));
    inet_ntoa_r(pnetif->netmask, addrString, 20);
    printf("netmask %s\n", addrString);
    memset(addrString, 0, sizeof(addrString));
    inet_ntoa_r(pnetif->gw, addrString, 20);
    printf("gw %s\n", addrString);
}

int main()
{
    /* after doing lwIP init, driver init and netifapi_netif_add */
    dhcp_client_start(&g_netif);

    /* Later if application wants to stop the DHCP client then it should
     * call netifapi_dhcp_stop() and netifapi_dhcp_cleanup() */
    /* netifapi_dhcp_stop(&g_netif); */
    /* netifapi_dhcp_cleanup(&g_netif); */
    return 0;
}
```

1.4.7.7 Sample Code for DHCP Server

```
#include "lwIP/opt.h"
#include "lwIP/netifapi.h"
#include "lwIP/inet.h"
#include "lwIP/netif.h"

struct netif g_netif;

int dhcp_server_start(struct netif *pnetif)
{
    int ret;
    char addrString[20] = {0};

    /* Calling netifapi_dhcps_start() will start DHCP server */
    ret = netifapi_dhcps_start(pnetif);

    if (ret == ERR_OK) {
        printf("dhcp server started successfully\n");
    } else {
        printf("dhcp server start failed\n");
    }
}

int main()
{
    /* after doing lwIP init, driver init and netifapi_netif_add */
    dhcp_server_start(&g_netif);

    /* Later if application wants to stop the DHCP client then it should
     * call netifapi_dhcps_stop()*/
    /*netifapi_dhcps_stop(&g_netif);*/
    return 0;
}
```

1.4.7.8 Sample Code for PPPoE Client

```
#if LWIP_PPPOE
#include "netif/ppp.h"

extern struct netif *pnetif_hi3516cv300;

u32_t osShellPPPoE(int argc, char **argv)
{
  if (0 == tcpip_init_finish)
  {
    PRINTK("%s: tcpip_init have not been called\n", __FUNCTION__);
    return LOS_NOK;
  }

  ret = lwip_pppoe_start(pnetif_hi3516cv300, "username", "123456");
  if(ret < 0)
  {
    PRINTK("pppoe : start pppoe failed: %d\n", ret);
    lwip_pppoe_stop(netif);
    return LOS_NOK;
  }
  return LOS_OK;}
}
#endif /* LWIP_PPPOE */
```

1.4.8 Limitations

Following limitations need to be considered before using Huawei LiteOS lwIP:

- Huawei LiteOS lwIP runs only on top of Huawei LiteOS, because the OS adaptation layer written on Huawei LiteOS lwIP is tightly coupled with Huawei LiteOS system interfaces.
- Huawei LiteOS lwIP can simultaneously run with two network interface (using ethernet and WiFi) but DHCP client cannot be used with both. Because DHCP client binds the UDP socket on DHCP client port, Huawei LiteOS lwIP does not allow multiple network interfaces (struct netif of ethernet and WiFi) to bind on same port.
- DNS client supports only A type resource record in response. While parsing the multiple answer records in DNS response message, if it encounters any malformed answer record then it stops parsing and returns success if it has any successfully parsed record or else it returns failure.
- lwIP provides the following types of socket APIs:
  - BSD style APIs

    These APIs are thread safe.
  - Netconn APIs

    These APIs are thread safe.

  - Low level APIs.

    Low level APIs are not thread safe and its not recommended to use this API. If you use low level APIs, the application thread and driver thread needs to take care of locking the core TCPIP thread functionality.

- Multithread usage of lwIP has the following limitations:
  - The lwIP core is not thread safe. If application wants to use Huawei LiteOS lwIP in a multithread environment, it should use "upper" API layers (netconn or sockets). If application uses low level API, it should protect the lwIP core.
  - Socket file descriptor created using Huawei LiteOS lwIP should not be used in multiple application threads. It should be used only in one application thread.
  - netif_xxx and dhcp_xxx are not thread safe APIs. So application should use the thread safe APIs available in netifapi module as netifapi_netif_xxx and netifapi_dhcp_xxx.
- Application must not create more than 254 network interfaces because the interface index maintained for network interfaces varies from 1 to 254.

1.4.9 Design Specification and Constraints

Below listed constraints needs to be considered before using this lwIP:

1. Huawei LiteOS lwIP runs only on top of Huawei LiteOS, because the OS adaptation layer written on lwIP is tightly coupled with Huawei LiteOS system interfaces.
2. Huawei LiteOS lwIP can simultaneously run with two network interfaces (using ethernet and WiFi) but the DHCP client cannot be used with both. Because DHCP client binds the UDP socket on DHCP client port, lwIP does not allow multiple network interfaces (struct netif of ethernet and WiFi) to bind on same port.

1.5 Huawei LiteOS lwIP-BSD Compatibility

The following table explains the BSD compatibility details of lwIP socket APIs :

Table 1-2

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
Linux Compatibility APIs	int lwIP_accept(int s, struct sockaddr *addr, socklen_t *addrlen);	int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);	None	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_bind(int s, const struct sockaddr *name, socklen_t namelen);	int bind(int socket, const struct sockaddr *address, socklen_t address_len);	None	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_shutdown(int s, int how);	int shutdown(int sockfd, int how);	None	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name. The "SHUT_RD" macro should not have been defined previously	Only "SHUT_RDWR" is supported for "how" parameter in this API. Closing one end of the full-duplex connection is not supported in lwIP.

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_connect(int s, const struct sockaddr *name, socklen_t namelen);	int connect(int socket, const struct sockaddr *address, socklen_t address_len);	None	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_getsockname (int s, struct sockaddr *name, socklen_t *namelen);	int getsockname(int sockfd, struct sockaddr *addr, socklen_t *addrlen);	None	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_getpeername (int s, struct sockaddr *name, socklen_t *namelen);	int getpeername(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);	None	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen)	int setsockopt(int socket, int level, int option_name, const void *option_value, socklen_t option_len);	1. For "level" parameter, only SOL_SOCKET, IPPROTO_IP, IPPROTO_TCP are supported. 2. Under SOL_SOCKET the options supported are: SO_BROADCAST, SO_KEEPALIVE, SO_SNDTIMEO, SO_RCVTIMEO, SO_RCVBUF, SO_REUSEADDR, SO_REUSEPORT,SO_NO_CHECK. For SO_SNDTIMEO, SO_RCVTIMEO, SO_RCVBUF, the macros lwIP_SO_SNDTIMEO, lwIP_SO_RCVTIMEO and lwIP_SO_RCVBUF should	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
			have been defined at compile time. For SO_REUSEADDR, SO_REUSEPORT, the macro SO_REUSE should have been defined at compile time. 3. Under IPPROTO_IP the options supported are: IP_TTL, IP_TOS. 4. Under IPPROTO_TCP the options supported are: TCP_NODELAY, TCP_KEEPALIVE, TCP_KEEPIDLE, TCP_KEEPINTVL, TCP_KEEPCNT. For TCP_KEEPIDLE, TCP_KEEPINTVL, TCP_KEEPCNT, the macro lwIP_TCP		

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
			_KEEPALIVE should have been defined at compile time. **NOTE** Options not mentioned above are not supported.		

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_getsockopt (int s, int level, int optname, void *optval, socklen_t *optlen);	int getsockopt(int socket, int level, int option_name, void *restrict option_value, socklen_t *restrict option_len);	1. For "level" parameter, only SOL_SOCKET, IPPROTO_IP, IPPROTO_TCP are supported. 2. Under SOL_SOCKET the options supported are: SO_ACCEPTCONN, SO_BROADCAST, SO_ERROR, SO_KEEPALIVE, SO_SNDTIMEO, SO_RCVTIMEO, SO_RCVBUF, SO_REUSEADDR, SO_REUSEPORT, SO_TYPE, SO_NO_CHECK. For SO_SNDTIMEO, SO_RCVTIMEO, SO_RCVBUF, the macros lwIP_SO_SNDTIMEO, lwIP_SO_	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
			RCVTIMEO and lwIP_SO_RCVBUF should have been defined at compile time. For SO_REUSEADDR, SO_REUSEPORT, the macro SO_REUSE should have been defined at compile time. 3. Under IPPROTO_IP the options supported are: IP_TTL, IP_TOS. 4. Under IPPROTO_TCP the options supported are: TCP_NODELAY, TCP_KEEPALIVE, TCP_KEEPIDLE, TCP_KEEPINTVL, TCP_KEEPCNT, TCP_QUEUE_SEQ. For		

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
			TCP_KEEPIDLE, TCP_KEEPINTVL, TCP_KEEPCNT, the macro lwIP_TCP_KEEPALIVE should have been defined at compile time. **NOTE** Options not mentioned above are not supported.		

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_listen(int s, int backlog);	int listen(int socket, int backlog);	For the "backlog" parameter to be used, the macro "TCP_LISTEN_BACKLOG" should have been enabled at compile time.	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_recv(int s, void *mem, size_t len, int flags);	ssize_t recv(int socket, void *buffer, size_t length, int flags);	For the "flags" parameter, only "MSG_DONTWAIT" & "MSG_PEEK" are supported. Other options are not supported. Return value type is different between lwIP (int) and Linux API (ssize_t). Note: Some linux flavours do seem to support "int" as the return value type. But, most common seems to ssize_t based on Posix Standard.	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_recvfrom(int s, void *mem, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);	ssize_t recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr * restrict from, socklen_t * restrict fromlen);	For the "flags" parameter, only "MSG_DONTWAIT" & "MSG_PEEK" are supported. Other options are not supported. Return value type is different between lwIP (int) and Linux API (ssize_t).	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_send(int s, const void *dataptr, size_t size, int flags);	ssize_t send(int sockfd, const void *buf, size_t len, int flags);	For the "flags" parameter, only "MSG_MORE" & "MSG_DONTWAIT" are supported. Other options are not supported. Return value type is different between lwIP (int) and Linux API(ssize_t) **NOTE** Some linux flavours do seem to support "int" as the return value type. But, most common seems to ssize_t based on Posix Standard.	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_sendto(int s, const void *dataptr, size_t size, int flags, const struct sockaddr *to, socklen_t tolen);	ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);	For the "flags" parameter, only "MSG_MORE" & "MSG_DONTWAIT" are supported. Other options are not supported. Return value type is different between lwIP (int) and Linux API(ssize_t)	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_socket(int domain, int type, int protocol);	int socket(int domain, int type, int protocol);	In lwIP: 1. For AF_INET socket, type SOCK_RAW| SOCK_DGRAM| SOCK_STREAM is supported. 2. For AF_PACKET socket, only type SOCK_RAW is supported	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. If the macro "lwIP_BSD_API" is defined at compile time of lwIP, then lwIP will expose API similar to linux function name.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, struct timeval *timeout);	int select(int nfds, fd_set *restrict readfds, fd_set *restrict writefds, fd_set *restrict errorfds, struct timeval *restrict timeout);	None	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	struct hostent* lwIP_gethostbyname(const char *name)	struct hostent *gethostbyname(const char *name);	Resolves the name to address. But, doesn't provide the list of aliases in the struct hostent as a part of result.	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. Macro "lwIP_DNS_API_DECLARE_STRUCTS" should be defined at compile time.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_gethostbyname_r(const char *name, struct hostent *ret, char *buf, size_t buflen, struct hostent **result, int *h_errnop)	int gethostbyname_r(const char *name, struct hostent *ret, char *buf, size_t buflen, struct hostent **result, int *h_errnop);	Behaves same as linux function except for the fact that it doesn't provide the list of aliases in the struct hostent as a part of result.	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile. Macro "lwIP_DNS_API_DECLARE_STRUCTS" should be defined at compile time.	
	int getifaddrs(struct ifaddrs **ifap);	int getifaddrs(struct ifaddrs **ifap);	If a_data of struct ifaddrs is unused member in lwIP.	lwIP initializationneeds to be completed before calling this API.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	void freeifaddrs(struct ifaddrs *ifa);	void freeifaddrs(struct ifaddrs *ifa);	None	Only the buffer allocated by getifaddrs needs to be passed to this API.	
Posix Compatibility APIs	int lwIP_read(int s, void *mem, size_t len);	ssize_t read(int fildes, void *buf, size_t nbyte);	Return type is different. Supports only Sockets and not all the file descriptors	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_write(int s, const void *dataptr, size_t size);	ssize_t write(int fildes, const void *buf, size_t nbyte);	Return type is different. Supports only Sockets and not all the file descriptors	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_close(int s);	int close(int fildes);	Supports only Sockets and not all file descriptors	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_fcntl(int s, int cmd, int val);	int fcntl(int fildes, int cmd, ...);	1. Function prototype doesn't support variable arguments. 2. Only F_GETFL & F_SETFL commands are supported. For F_SETFL, only O_NONBLOCK is supported for val.	If the macro "lwIP_COMPAT_SOCKETS" is defined at compile time, then it will create a macro similar to that of linux function name. This will help the application code written using linux function names to compile.	

Category	lwIP API	Linux API	Difference	Pre-Requisites	Remarks
	int lwIP_ioctl(int s, long cmd, void *argp);	int ioctl(int fd, unsigned long request, ...);	Linux API supports variable argument support. But lwIP API supports only one void * as 3rd argument. And also it supports only the below listed options. ● SIOCADDRT ● SIOCGIFADDR ● SIOCSIFADDR ● SIOCGIFNETMASK ● SIOCSIFNETMASK ● SIOCSIFHWADDR ● SIOCGIFHWADDR ● SIOCGIFFLAGS ● SIOCSIFFLAGS ● SIOCGIFNAME ● SIOCSIFNAME ● SIOCGIFCONF ● SIOCGIFINDEX ● FIONBIO	None	

1.6 Network Security Redline Description

This chapter provides the network security declaration and risk rectification.

1.6.1 Network Security Risk Rectification

Security Redline Tool used to scan and fix issues:

All issues related with CodeDEX are fixed

Security of Management Channel:

NA

Security of Operation System:

NA

Protocol and Interface against Attack:

lwIP is tested with Codenomicon and the below listed defects are fixed:

[DTS2015060309169]: Core dump on receiving fragmented IP packet with length field set to less than 20.

[DTS2015020203052]: Core dump on receiving IP packet with length field set to less than 20.

[DTS2015020503372]: Core dump on receiving TCP packet with data offset field set to greater than 5.

[DTS2016041406653]: Invalid memory read while parsing TCP packet with SACK option longer than 40.

[DTS2016071303586]: Running codenomicon suite gives a memory leak.

Web Security:

NA

Security of Product Development, Release, and Installation:

- Anti-Virus Scan tool: Refer to the Release Notes.

Security of Database:

NA

Protection of Sensitive Data:

NA

Security of System Management and Maintenance:

NA

Monitor Interface and Prevent Illegal Monitoring:

NA

Third party component security design

Included in Security Threat & Vulnerability Requirement Analysis

1.6.2 Network Security Declaration

This section lists the network security related declarations

User should take care of the following security considerations in order to avoid printing of user sensitive information:

Debug Log :

1.The DebugFlag can be enabled or disabled using Macros in opt.h header file by setting to the flag LWIP_DBG_ON By default, Debug Flag is Disabled.

Option Code	Recommended Value	Impact
LWIP_DEBUGF	LWIP_DBG_OFF	If value is set to LWIP_DBG_ON(1) [Enabled], then it may expose some user sensitive information.

It is important to understand the declarations to eliminate any related and consequential security risks. User should take care for the following security considerations:

- LWIP_RAND macro is expected to be registered by the user by invoking SSP registration function, which is used in DHCP and DNS to generate random numbers for Transaction ID. Application need to register suitable random number generator function with this callback.
- Application need to register suitable random number generator function with this callback.
- Application data given to lwIP for transmitting with the help of UDP or TCP will be temporarily kept in lwIP buffer, later memory cleaning is not done explicitly before freeing the buffer. This is based on the assumption that, application will not provide user sensitive data as plain text to lwIP. As it generally uses transport security protocol (like TLS or DTLS) and sends encrypted message to lwIP, in case of user sensitive data.
- [VPPTECH-262]: Provided solution for following security issues:

  - Provided fix for event indication callback locking mechanism.

  - Provided fix for not allowing IP packet greater than 65535 length.

  - Initialization of dhcps callback.

1.7 lwIP Version Building

The purpose of this is to provide details information for compiling VPP lwIP libraries.

Compilation Dependency:

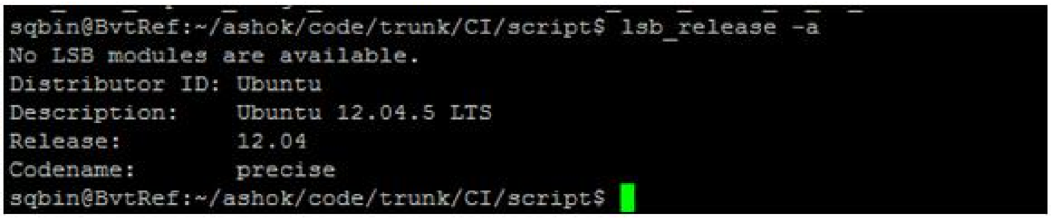
1. Remove the read only attribute of the entire directory.
2. Cross compilers are installed correctly.
3. The Code of VPP components is downloaded from SVN properly.

Environment setup:

- **Hardware requirements**

Following Hardware systems required to compile below lwIP Platforms.

Any UbuntuX86 64 bit PC with at least 5GB data disk.



```
sqbin@BvtRef:~/ashok/code/trunk/CI/script$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 12.04.5 LTS
Release:        12.04
Codename:       precise
sqbin@BvtRef:~/ashok/code/trunk/CI/script$
```

- **Software requirements**

Following software tools used for LWIP Compilation

1. Compiler : arm-hisiv500-linux

Version Compilation flow

Modify compile script and compile

- Download the code from SVN
- Transfer the code to the compile Environment

cd to code/trunk/CI/script

Chmod +x build_lwIP_tcp_hi3516a.sh build_lwIP_tcp_hi3518ev200.sh build_lwIP_tcp_hi3519_cortex-a17.sh build_lwIP_tcp_hi3519_cortex-a7.sh

dos2unix build_lwIP_tcp_hi3516a.sh build_lwIP_tcp_hi3518ev200.sh build_lwIP_tcp_hi3519_cortex-a17.sh build_lwIP_tcp_hi3519_cortex-a7.sh

Execute as sh <script name> or ./<Script name>

Example :

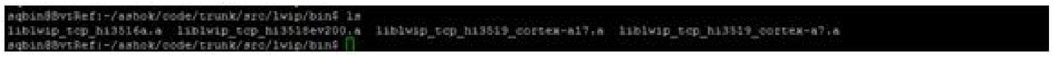
1. build_lwIP_tcp_hi3516a.sh
2. build_lwIP_tcp_hi3518ev200.sh
3. build_lwIP_tcp_hi3519_cortex-a17.sh
4. build_lwIP_tcp_hi3519_cortex-a7.sh

Or

1. Sh build_lwIP_tcp_hi3516a.sh
2. Sh build_lwIP_tcp_hi3518ev200.sh
3. Sh build_lwIP_tcp_hi3519_cortex-a17.sh
4. Sh build_lwIP_tcp_hi3519_cortex-a7.sh

Find the lib

Lib will be stored in the below path : /code/trunk/src/lwip/bin



```
sqbin@8vtRef:~/ashok/code/trunk/src/lwip/bin$ ls
liblwip_tcp_hi3516a.a  liblwip_tcp_hi3518ev200.a  liblwip_tcp_hi3519_cortex-a17.a  liblwip_tcp_hi3519_cortex-a7.a
sqbin@8vtRef:~/ashok/code/trunk/src/lwip/bin$
```

Preparing the Include files.

Include will be present in following directory

…/code/trunk/src/lwip/include.

1.8 FAQs

- How do we solve compilation errors?

Solution: Please see setting compilation environment section and ensure the correct compiler is set according to the environment you are using.

1.9 Glossary

Term	Abbreviation	Description
Address Resolution Protocol	ARP	Address Resolution Protocol is a network protocol used to convert an IP Address to a Physical Address.
Dynamic Host Configuration Protocol	DHCP	Dynamic Host Configuration Protocol is a standardized networking protocol used on IP networks for dynamically distributing network configuration parameters such as IP Addresses for interfaces and services.
Lightweight IP	lwIP	lwIP (lightweight IP) is a widely used open source TCP/IP stack designed for embedded systems.

Term	Abbreviation	Description
Lite-OS	-	Lite-OS is an operating system developed inside Huawei based on CMSIS software standard.
Internet Control Message Protocol	ICMP	The Internet Control Message Protocol (ICMP) is used by network devices, like routers, to send error messages indicating a requested service is not available or a host could not be reached.
Internet Protocol	IP	The protocol within TCP/IP that governs the breakup of data messages into packets, the routing of the packets from sender to destination network and station, and the reassembly of the packets into the original data messages at the destination. IP runs at the internetwork layer in the TCP/IP model equivalent to the network layer in the ISO/OSI reference model.
Internet of Things	IoT	The Internet of Things is a computing concept that describes a scenario where everyday physical objects will be connected to the Internet and be able to identify themselves to other devices.
Transmission Control Protocol	TCP	The protocol within TCP/IP that governs the breakup of data messages into packets to be sent using Internet Protocol, and the reassembly and verification of the complete messages from packets received by IP. A connection-oriented, reliable protocol £¨reliable in the sense of ensuring error-free delivery, TCP corresponds to the transport layer in the ISO/OSI reference model.

Term	Abbreviation	Description
User Datagram Protocol	UDP	A TCP/IP standard protocol that allows an application program on one device to send a datagram to an application program on another. UDP uses IP to deliver datagrams. UDP provides application programs with the unreliable connectionless packet delivery service. That is, UDP messages may be lost, duplicated, delayed, or delivered out of order. The destination device does not actively confirm whether the correct data packet is received.
Point to Point Protocol over Ethernet	PPPoE	PPPoE provides a standard for connecting multiple clients on an Ethernet local area network (LAN) network to a remote broadband access server (BAS). Ethernet is used to connect multiple PPPoE clients and form a LAN. Through the remote BAS, the clients can be connected to the Internet. Identity authentication and charging for each accessed client are achieved by using the PPP.