Web Development Application Notes

RT-THREAD Documentation Center

WWW.RT-THREAD.ORG

Friday 28th September, 2018

Table of contents

This application note describes how to use the standardized API to develop network

Network applications.

1 Purpose and structure of this paper

1.1 Purpose and Background of this Paper

More and more single-chip microcomputers need to access Ethernet to send and receive data. There are also many access solutions on the market. You can use a single-chip microcomputer plus a PHY chip with its own hardware protocol stack to access the network, or you can use a single-chip microcomputer running a software protocol stack plus a PHY chip to access the network. Different access solutions require calling different APIs, which reduces the portability of upper-level applications.

In order to facilitate users to develop network applications, RT-Thread introduces a network framework and provides standardized API interfaces for developing network applications. At the same time, RT-Thread also provides a large number of network component packages to facilitate users to quickly develop their own applications.

1.2 Structure of this paper

This article first introduces the RT-Thread network framework and standardized APIs, then introduces the basic applications implemented using these APIs: tcp client and udp client, and finally introduces the network component package provided by RT-Thread, and gives code examples for running NTP (obtaining time through the network) and MQTT (sending and receiving data through MQTT) on the Zhengdian Atom STM32F4 Explorer development board.

2 Network Framework Introduction

RT-Thread provides a network management framework, as shown in the network framework diagram:

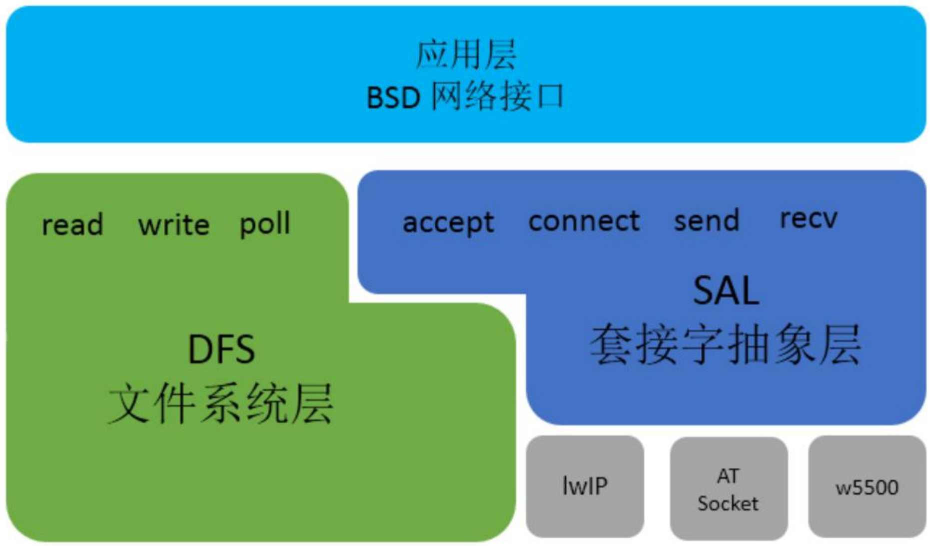

Figure 1: *sal_frame*

The top layer is the network application layer, which provides a set of standard BSD Socket APIs, such as socket, connect and other functions. Most network development applications in the system.

The second part is the file system layer. In the RT-Thread system, the DFS file system program can use standard interface functions to implement different file system operations. The network socket interface also supports the file system structure. The network socket descriptor created when using the network socket interface is managed by the file system, so the network socket descriptor can also use the standard file operation interface. The file system layer provides interfaces for the upper application layer: read, write, close, poll/select, etc.

The third part is the socket abstraction layer, through which the RT-Thread system can adapt to different network protocol stacks in the lower layer and provide a unified network programming interface to the upper layer to facilitate the access of different protocol stacks. The socket abstraction layer provides interfaces for the upper application layer: accept, connect, send, recv, etc.

The fourth part is the protocol stack layer, which includes several commonly used TCP/IP protocol stacks, such as the lightweight TCP/IP protocol stack lwIP commonly used in embedded development and the AT Socket network function implementation independently developed by RT-Thread. These protocol stacks or network function implementations directly contact the hardware to complete the conversion of data from the network layer to the transport layer.

The interface provided by RT-Thread's network application layer is mainly based on the standard BSD Socket API, which ensures that the program Programs can be written and debugged on a PC and then ported to the RT-Thread operating system.

3 API Introduction

3.1 BSD Socket API

BSD Socket (Berkeley Socket) was originally developed by the University of California, Berkeley for Unix systems. Most operating systems implement the Berkeley socket interface, and mainstream programming languages also support the use of BSD Socket Develop network applications. BSD Socket can be said to be the standard interface for connecting to the Internet.

RT-Thread also implements the BSD Socket interface API, which can be used in other operating systems or programming languages. Network applications implemented by Socket can be directly run in RT-Thread without any modification.

3.2 Creating a socket

```
int socket(int domain, int type, int protocol);
```

Used to allocate a socket descriptor and the resources it uses according to the specified address family, data type, and protocol.

parameter	describe
domain	Protocol family type
type	agreement type
protocol	The actual transport layer protocol used
return	describe
0	On success, returns an integer representing the socket descriptor.
-1	fail

domain

Protocol family

- PF_INET: IPv4
- PF_INET6: IPv6.

type

agreement type

- SOCK_STREAM: reliable connection-oriented service or Stream Sockets
- SOCK_DGRAM: Datagram Sockets
- SOCK_RAW: raw protocol of the network layer

3.3 Bind Socket (bind)

```
int bind(int s, const struct sockaddr *name, socklen_t namelen);
```

Used to bind the port number and IP address to the specified socket. When using socket() to create a socket, only

The protocol family is given, but no address is assigned. Before the socket can accept connections from other hosts, it must be bound()

Bind it to an address and port number.

parameter	describe
s	Socket Descriptor
name	Pointer to the sockaddr structure, representing the address to be bound
namelen	The length of the sockaddr structure
return	describe
0	success
-1	fail

3.4 Listening Sockets (listen)

```
int listen(int s, int backlog);
```

Used by the TCP server to listen for connections on the specified socket.

parameter	describe
s	Socket Descriptor
backlog	Indicates the maximum number of connections that can be waiting at one time
return	describe
0	success
-1	fail

3.5 Accept Connection

```
int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
```

When the application listens for connections from other hosts, it uses the accept() function to initialize the connection.

Each connection creates a new socket and removes the connection from the listen queue.

parameter	describe
s	Socket Descriptor
addr	Client device address information
addrlen	The length of the client device address structure

parameter	describe
return	describe
0	On success, returns the newly created socket descriptor
-1	fail

3.6 Establish a connection

```
int connect(int s, const struct sockaddr *name, socklen_t namelen);
```

Used to establish a connection with the specified socket.

parameter	describe
s	Socket Descriptor
name	Server address information
namelen	The length of the server address structure
return	describe
0	On success, returns the newly created socket descriptor
-1	fail

3.7 TCP data sending (send)

```
int send(int s, const void *dataptr, size_t size, int flags);
```

Send data, commonly used in TCP connections.

parameter	describe
s	Socket Descriptor
dataptr	Pointer to the data being sent
size	The length of the data sent
flags	Flag, usually 0
return	describe
>0	If successful, returns the length of the data sent
<=0	fail

3.8 TCP Data Reception (recv)

int recv(int s, **void** *mem, size_t len, **int** flags);

Receive data, commonly used for TCP connections.

parameter	describe
s	Socket Descriptor
mem	Received data pointer
len	Received data length
flags	Flag, usually 0
return	describe
>0	If successful, returns the length of the received data
=0	The destination address has been transmitted and the connection is closed
<0	fail

3.9 UDP data sending (sendto)

int sendto(int s, **const void** *dataptr, size_t size, **int** flags, **const struct** sockaddr *to, socklen_t tolen);

Send data, commonly used for UDP connections.

parameter	describe
s	Socket Descriptor
dataptr	Pointer to the data being sent
size	The length of the data sent
flags	Flag, usually 0
to	Target address structure pointer
tolen	Length of target address structure
return	describe
>0	If successful, returns the length of the data sent
<=0	fail

3.10 UDP data reception (recvfrom)

int recvfrom(int s, void *mem, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);

Receive data, commonly used for UDP connections.

parameter	describe
s	Socket Descriptor
mem	Received data pointer
len	Received data length
flags	Flag, usually 0
from	Receive address structure pointer
fromlen	Length of receiving address structure
return	describe
>0	If successful, returns the length of the received data
=0	The receiving address has been transmitted and the connection has been closed
<0	fail

3.11 Close socket (closesocket)

int closesocket(int s);

Close the connection and release resources.

parameter	describe
s	Socket Descriptor
return	describe
0	success
-1	fail

3.12 Close the socket according to the settings (shutdown)

int shutdown(int s, int how);

Provides more permissions to control the socket closing process.

parameter	describe
s	Socket Descriptor
how	Socket control mode
return	describe
0	success
-1	fail

how

- 0: Stop receiving current data and refuse to receive future data;
- 1: Stop sending data and discard unsent data;
- 2: Stop receiving and sending data.

3.13 Setting Socket Options (setsockopt)

int setsockopt(int s, **int** level, **int** optname, **const void** *optval, socklen_t optlen);

Set the socket mode, modify socket configuration options.

parameter	describe
s	Socket Descriptor
level	Protocol stack configuration options
optname	The name of the option to be set
optval	Set the buffer address of the option value
optlen	Set the buffer length for option values
return	describe
=0	success
<0	fail

level

- SOL_SOCKET: socket layer
- IPPROTO_TCP: TCP layer
- IPPROTO_IP: IP layer

optname

- SO_KEEPALIVE: Set the keep-alive option
- SO_RCVTIMEO: Set the socket data receive timeout
- SO_SNDTIMEO: Set the socket data send timeout

3.14 Get Socket Options (getsockopt)

int getsockopt(int s, int level, int optname, void *optval, socklen_t * optlen);

Get socket configuration options.

parameter	describe
s	Socket Descriptor
level	Protocol stack configuration options
optname	The name of the option to be set
optval	Get the buffer address of the option value
optlen	Get the buffer length address of the option value
return	describe
=0	success
<0	fail

3.15 Get remote address information (getpeername)

int getpeername(int s, struct sockaddr *name, socklen_t *namelen);

Get the remote address information connected to the socket.

parameter	describe
s	Socket Descriptor
name	Address structure pointer for receiving information
namelen	Length of the address structure of the received information
return	describe
=0	success
<0	fail

3.16 Get local address information (getsockname)

int getsockname(int s, **struct** sockaddr *name, socklen_t *namelen);

Get local socket address information.

parameter	describe
s	Socket Descriptor
name	Address structure pointer for receiving information
namelen	Length of the address structure of the received information
return	describe
=0	success
<0	fail

3.17 Configure socket parameters (ioctlsocket)

int ioctlsocket(int s, **long** cmd, **void** *arg);

Sets the socket control mode.

parameter	describe
s	Socket Descriptor
cmd	Socket operation commands
arg	Parameters of the operation command
return	describe
=0	success
<0	fail

cmd

• FIONBIO: Enable or disable the non-blocking mode of the socket. The arg parameter is 1 for enabling non-blocking and 0 for disabling non-blocking.
block.

3.18 Debug API

Here are three network information viewing commands provided by RT-Thread. It is very convenient to enter the command in the shell. Check the network connection status to facilitate debugging.

3.18.1. ifconfig

ifconfig can print out the board's current network connection status, IP address, gateway address, dns and other information.

```
msh />ifconfig
network interface: e0 (Default)
MTU: 1500
MAC: 00 04 9f 05 44 e5
FLAGS: UP LINK_UP ETHARP
ip address: 192.168.12.127
gw address: 192.168.10.1
net mask  : 255.255.0.0
dns server #0: 192.168.10.1
dns server #1: 223.5.5.5
msh />
```

Figure **2:** *an010_ifconfig*

3.18.2. netstate

netstate can print out all TCP/IP connection information of the board

```
Active PCB states:
#0 192.168.12.186:49153 <==> 139.196.135.135:1883 snd_nxt 0x00001
AEA rcv_nxt 0x9D4D8F35 state: ESTABLISHED
Listen PCB states:
TIME-WAIT PCB states:
Active UDP PCB states:
#0 4 0.0.0.0:68 <==> 0.0.0.0:67
```

Figure **3:** *an011_netstate*

3.18.3. dns

DNS can print the DNS server address currently in use, and can also enter the DNS server IP address to manually set DNS server address

```
msh />dns                           查看dns
dns server #0: 192.168.10.1
dns server #1: 223.5.5.5
msh />dns 114.114.114.114           设置dns
dns : 114.114.114.114
msh />dns                           查看dns
dns server #0: 114.114.114.114
dns server #1: 223.5.5.5
msh />
```

Figure **4:** *an011_dns*

4. Preparation

• Prepare RT-Thread source code • Prepare

ENV • A development

board that can access the Internet. Here we take the Zhengdian Atom STM32F4 Explorer development board as an example. •

Port the underlying network driver. For driver porting, refer to the Network Protocol Stack Driver Porting Notes. • Network

debugging tool

4.1 Hardware Connection Preparation

The DHCP function is enabled by default in RT-Thread's BSP. A DHCP server is required to assign IP addresses. Common connection extensions are shown in the figure:

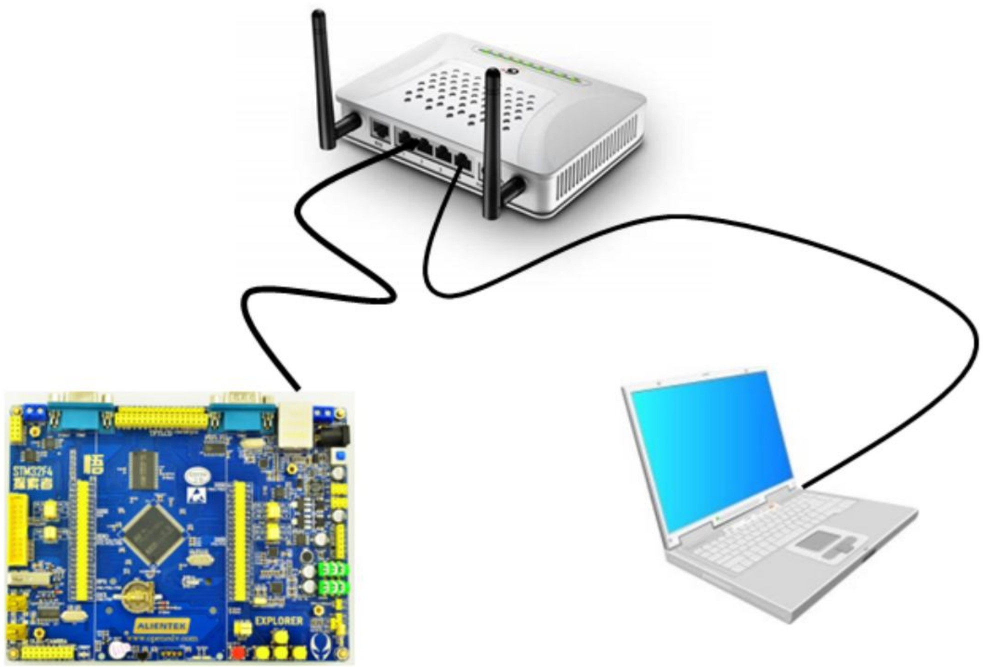

Figure **5:** *eth_RJ45*

If there is no convenient actual environment, you can also configure a fixed IP through ENV first, and then connect directly to the debugger with a network cable. Trial computer.

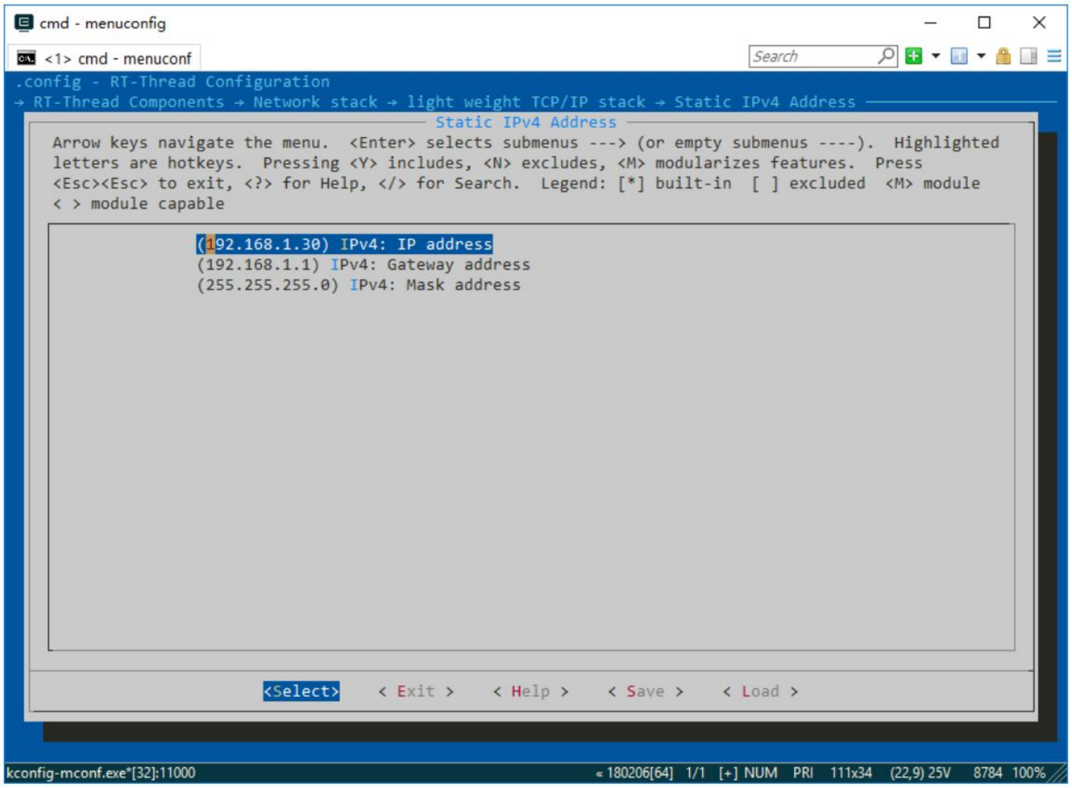

Figure **6:** *ipadress*

The computer and development board need to be set with IP addresses in the same network segment.

4.2 **ENV** Configuration

RT-Thread can easily configure and generate projects through ENV

• Open env and enter the rt-thread/bsp/stm32f40x directory • Enter set RTT_CC=keil

in the env command line to set the toolchain type to keil • Enter menuconfig in the env command line to enter the

configuration interface • Change the console output to your own board in the RT-Thread

Kernel -> Kernel Device object page

Serial port number

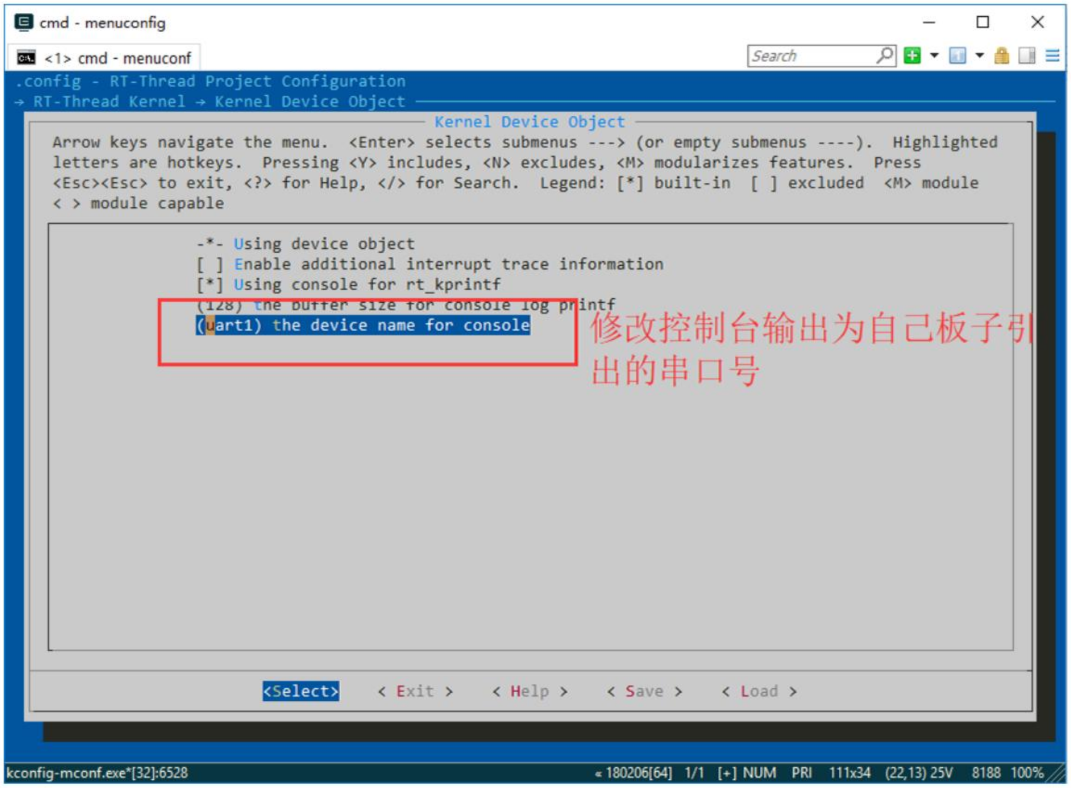

Figure 7: *ENV_uart*

- Check the maximum number of open files on the RT-Thread Components -> Device virtual file system page Is it less than 16? If so, increase the value.

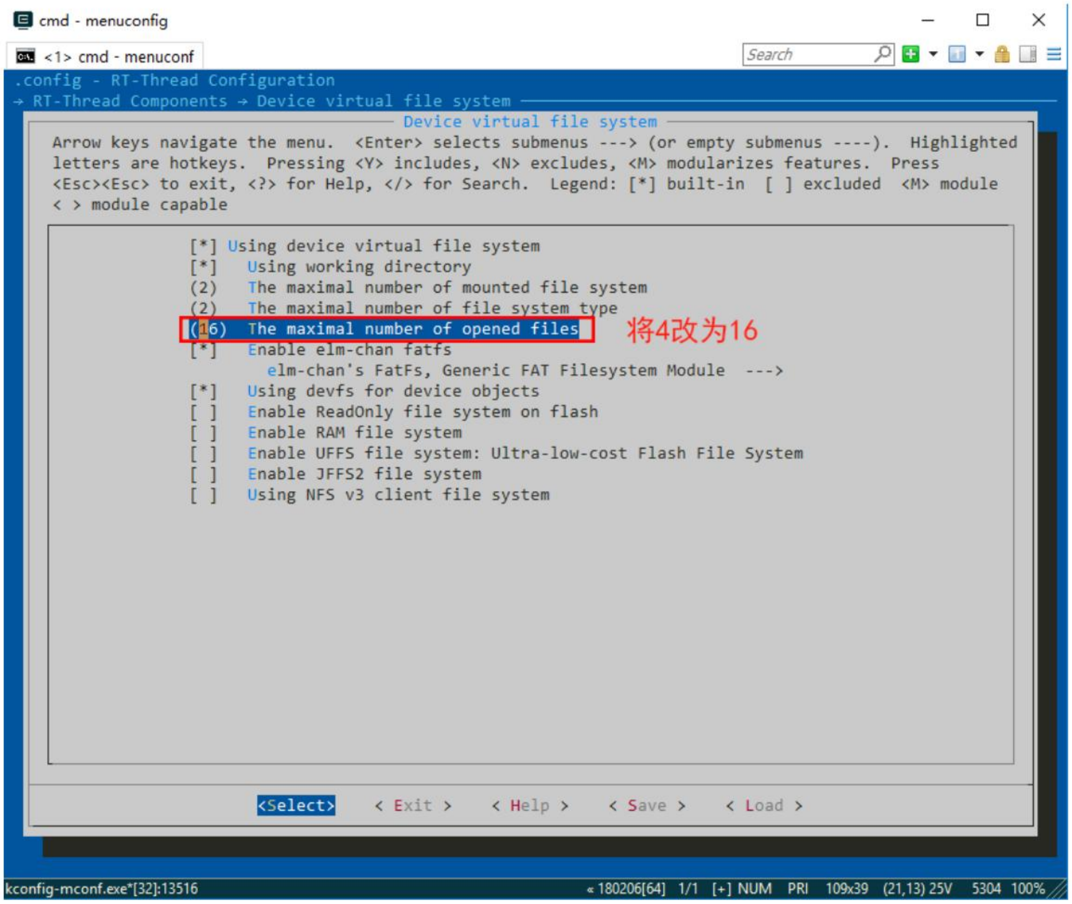

Figure **8:** *menuconfig_filesystem*

Enable the sal layer in RT-Thread Components -> Network -> Socket abstraction layer page and Enable BSD socket

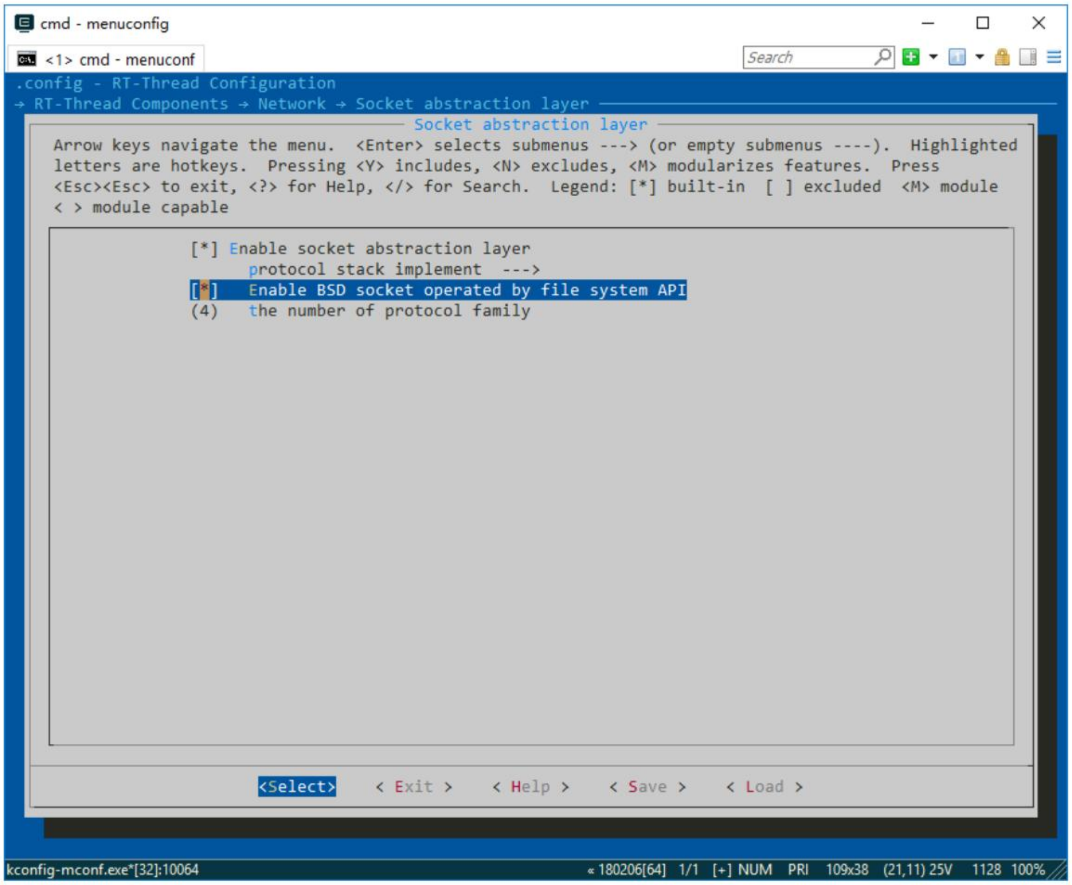

Figure 9: *an011_sai*

- Enable lwip in RT-Thread Components -> Network -> light weight TCP/IP stack

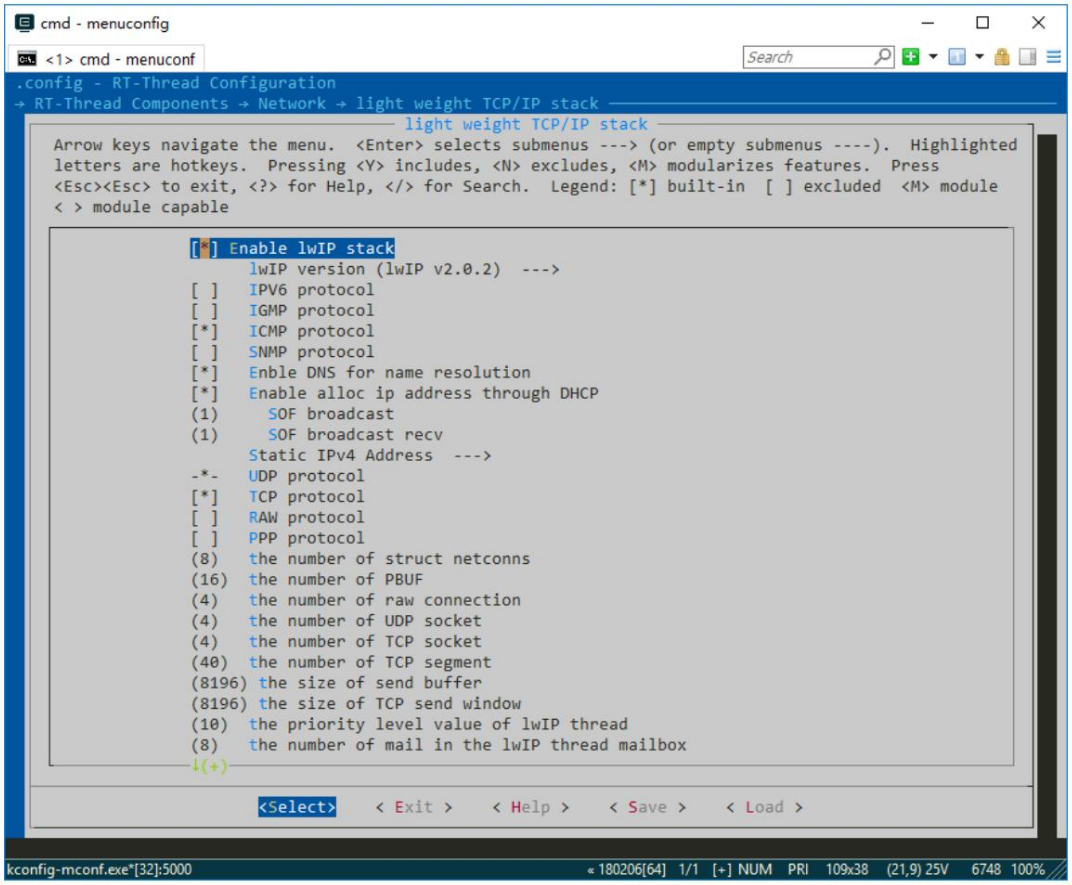

Figure **10:** *an011_lwip*

- In RT-Thread online packages -> misellaneous packages -> samples: RT-Thread kernel and components samples->network sample options page enable tcp client and udp client (Basic application)

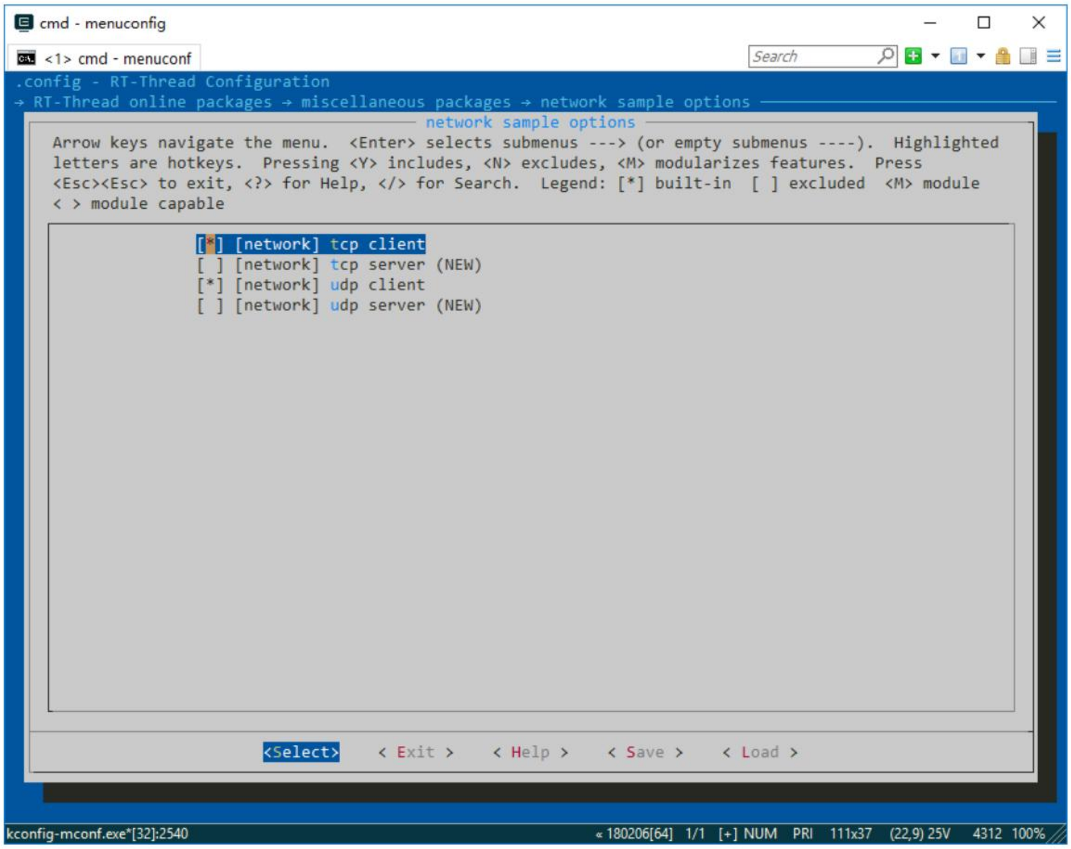

Figure **11:** *ENV_client*

- In RT-Thread online packages -> IoT - internet of things -> netutils: Networking Utilities for RT-Thread page enables ntp (advanced application)

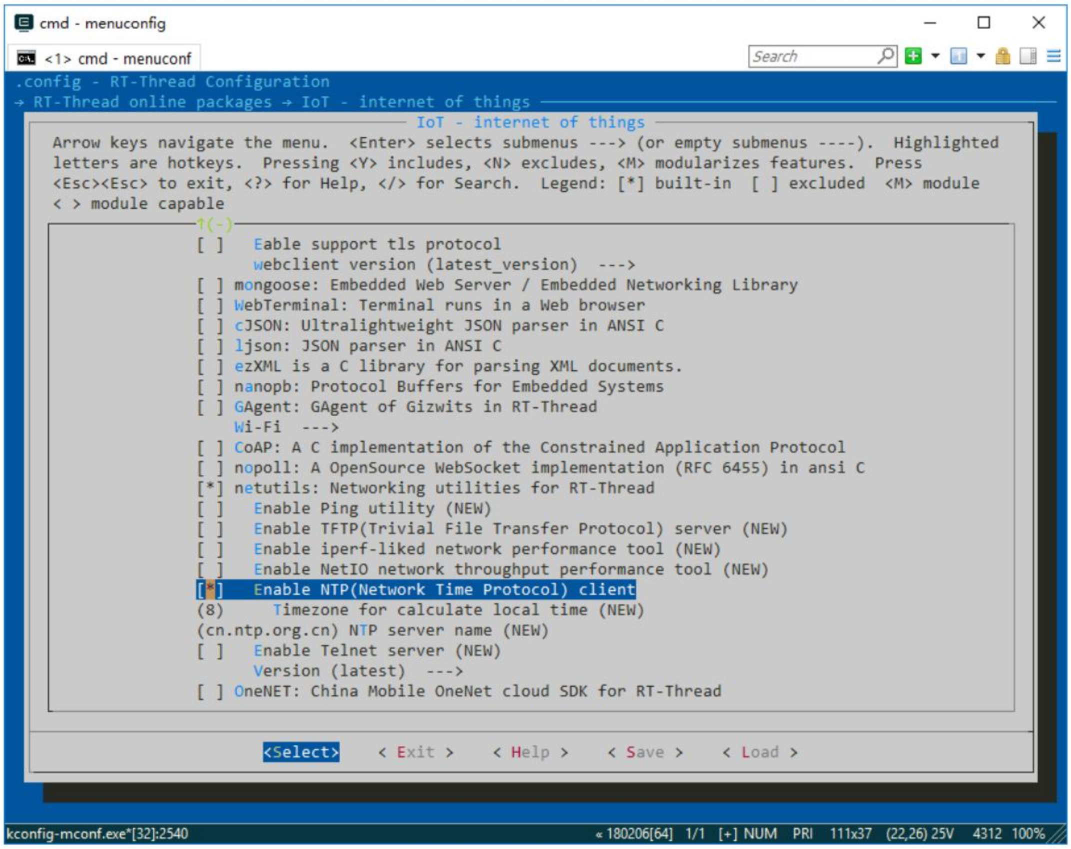

Figure **12:** *ENV_ntp*

- Enable MQTT (advanced applications)

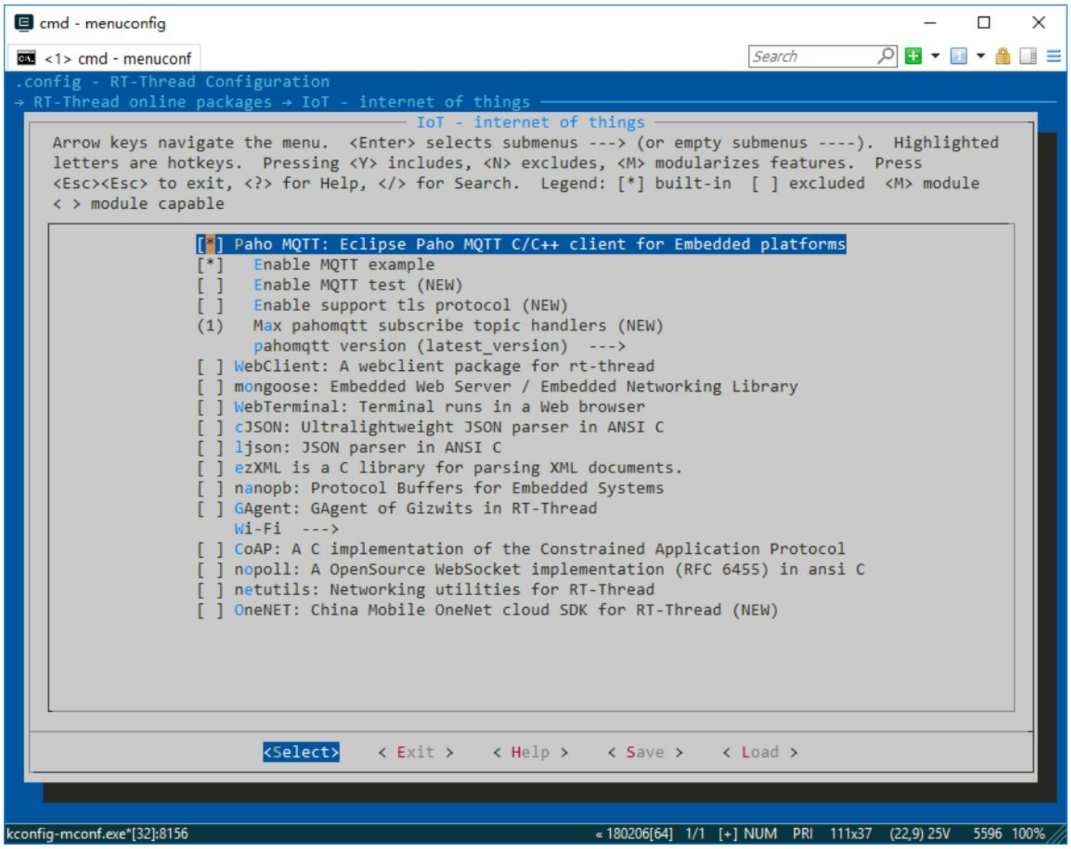

Figure **13:** *ENV_mqtt*

• Press ESC to exit the configuration interface •

Enter scons –target=mdk5 -s in the env command line to generate the mdk5 project. • Open the project and compile • Download

the code

4.3 Network Testing

Compile the project generated by env and download it to the board. You can see that two lights of the network port will light up and one will flash.

This indicates that the PHY has been initialized normally.

Enter ifconfig in the shell to print the network status of the board. If the IP address is obtained normally, it means that the network driver is normal.

The preparation is complete.

```
msh />ifconfig
network interface: e0 (Default)
MTU: 1500
MAC: 00 04 9f 05 44 e5
FLAGS: UP LINK_UP ETHARP
ip address: 192.168.12.127
gw address: 192.168.10.1
net mask  : 255.255.0.0
dns server #0: 192.168.10.1
dns server #1: 223.5.5.5
msh />
```

Figure **14:** *ifconfig*

5 Basic Applications

In practical applications, the microcontroller generally acts as a client to exchange data with the server. Here, the TCP client and udp client as examples.

5.1 tcpclient

This example shows how to create a TCP client to communicate with a remote server.

Enter tcpclient URL PORT in the shell to connect to the server

Program function: Receive and display the information sent from the server, and exit the program if the information beginning with 'q' or 'Q' is received

5.1.1. Source code analysis

```
void tcpclient(int argc, char **argv) {

    int ret;
    char *recv_data;
    struct hostent *host;
    int sock, bytes_received; struct
    sockaddr_in server_addr; const char *url; int
    port;


    /* The number of received parameters is less than 3*/
    if (argc < 3) {

        rt_kprintf("Usage: tcpclient URL PORT\n");
```

```
        rt_kprintf("Like: tcpclient 192.168.12.44 5000\n");
        return ;
    }

    url = argv[1]; port =
    strtoul(argv[2], 0, 10);

    /* Get the host address through the function entry parameter url (if it is a domain name, domain name resolution will be performed) */
    host = gethostbyname(url);

    /* Allocate a buffer to store received data */
    recv_data = rt_malloc(BUFSZ); if (recv_data
    == RT_NULL) {

        rt_kprintf("No memory\n"); return;

    }

    /* Create a socket, type is SOCKET_STREAM, TCP type*/ if ((sock =
    socket(AF_INET, SOCK_STREAM, 0)) == -1) {

        /* Failed to create socket*/
        rt_kprintf("Socket error\n");

        /* Release the receive buffer */
        rt_free(recv_data);
        return;
    }

    /* Initialize the pre-connected server address*/
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port); server_addr.sin_addr
    = *((struct in_addr *)host->h_addr); rt_memset(&(server_addr.sin_zero), 0,
    sizeof(server_addr.sin_zero)) ;

    /* Connect to the server */
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(struct sockaddr)) == -1)

    {
        /* Connection failed*/
        rt_kprintf("Connect fail!\n"); closesocket(sock);



        /*Release the receive buffer*/
        rt_free(recv_data);
        return;
```

```
    }

    while (1) {

        /* Receive maximum BUFSZ - 1 byte of data from sock connection*/
        bytes_received = recv(sock, recv_data, BUFSZ - 1, 0); if (bytes_received <
        0) {

            /* Receiving failed, close this connection*/
            closesocket(sock);
            rt_kprintf("\nreceived error,close the socket.\r\n");

            /* Release the receive buffer */
            rt_free(recv_data);
            break;

        } else if (bytes_received == 0) {

            /* Print the warning message that the recv function returns 0*/
            rt_kprintf("\nReceived warning,recv function return 0.\r\n");

            continue;
        }

        /* Received data, clear the end*/
        recv_data[bytes_received] = '\0';

        if (strncmp(recv_data, "q", 1) == 0 || strncmp(recv_data, "Q", 1)
             == 0)
        {
            /* If the first letter is q or Q, close the connection */
            closesocket(sock);
            rt_kprintf("\n got a 'q' or 'Q',close the socket.\r\n");

            /* Release the receive buffer */
            rt_free(recv_data); break;

        }
        else
        {
            /* Display the received data on the control terminal */
            rt_kprintf("\nReceived data = %s ", recv_data);
        }

        /* Send data to sock connection */
        ret = send(sock, send_data, strlen(send_data), 0);
```

```
            if (ret < 0) {

                /* Receiving failed, close this connection*/
                closesocket(sock);
                rt_kprintf("\nsend error,close the socket.\r\n");

                rt_free(recv_data); break;


            } else if (ret == 0) {

                /* Print the warning message that the send function return value is 0*/
                rt_kprintf("\n Send warning,send function return 0.\r\n");
            }

        } return;
}
```

5.1.2. Operation results

Use the network debugging tool to build a TCP server on your computer and record the open ports.

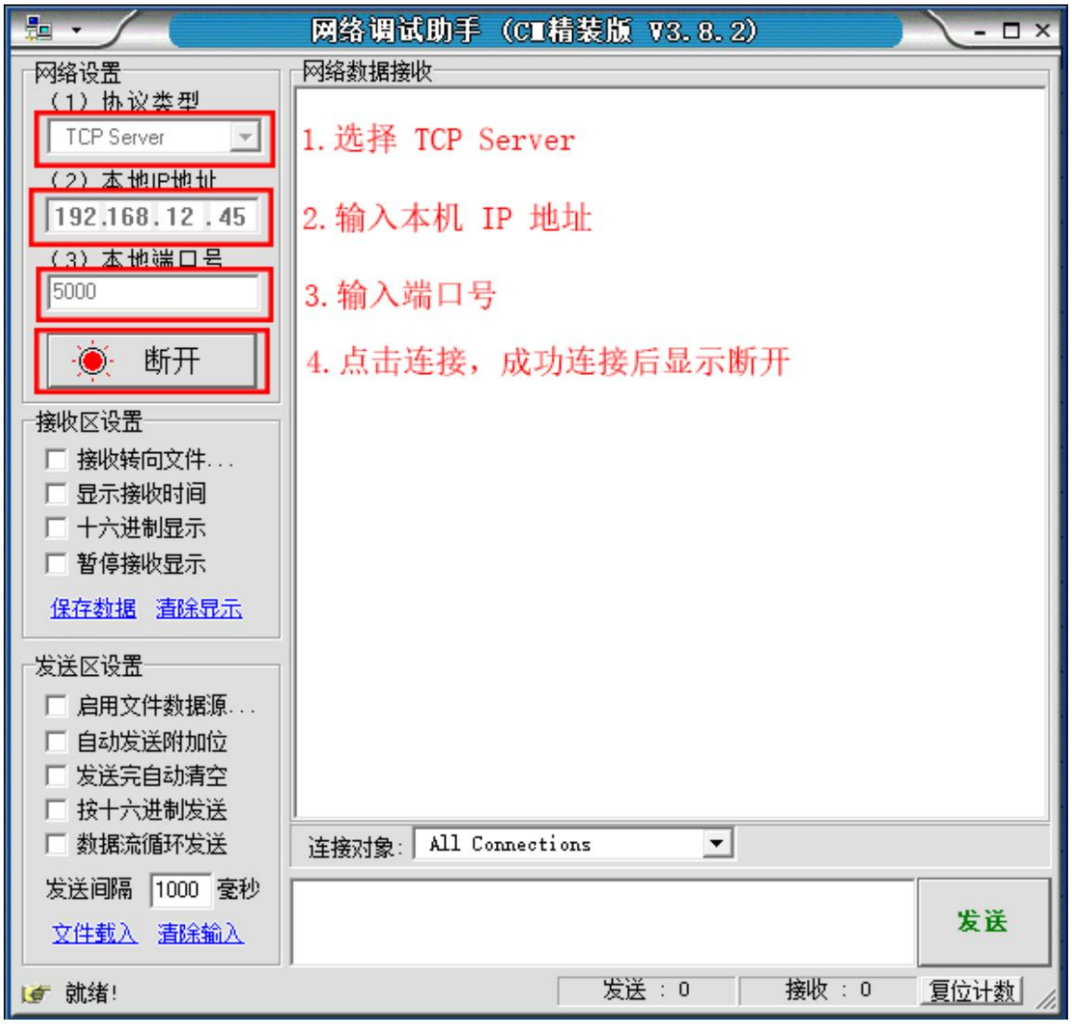

Figure **15:** *TCP Server_set*

In the shell, enter the IP address of the tcpclient PC and the port number you just recorded.

```
msh />tcpclient 192.168.12.45 5000
```

Use the server to send Hello RT-Thread!, and the received information will be displayed in the shell

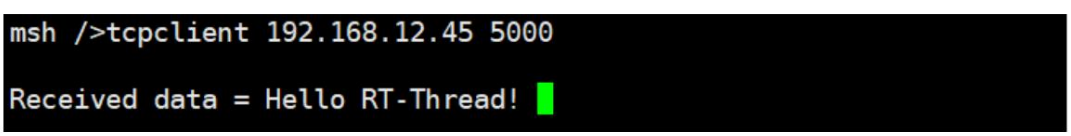

Figure **16:** *tcpclient_shell*

The server will receive the message This is TCP Client from RT-Thread.

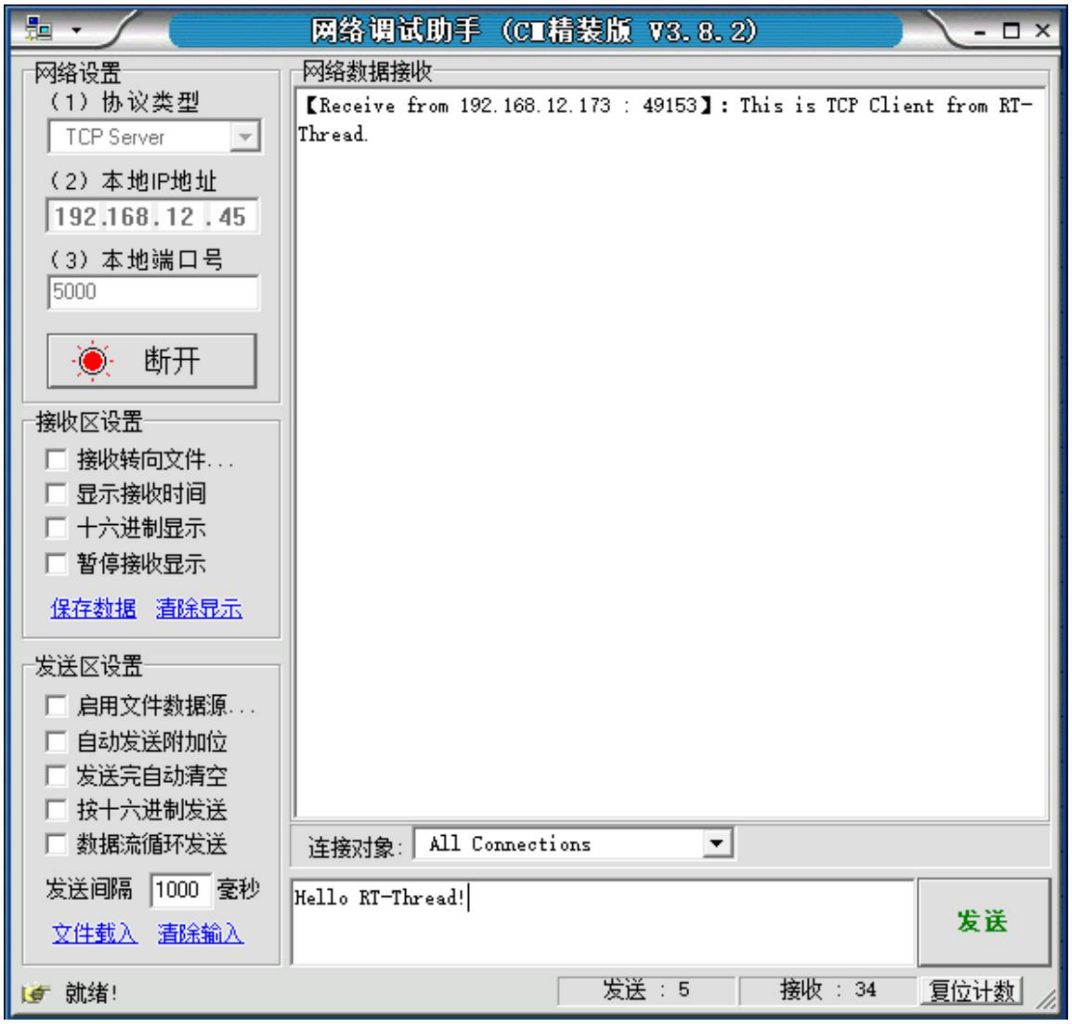

Figure **17:** *tcpclient*

5.2 udpclient

This example shows how to create a UDP client and send data to a remote server.

Enter udpclient URL PORT in the shell to connect to the server

Program function: send information to the server (default 10)

5.2.1. Source code analysis

```
void udpclient(int argc, char **argv) {

    int sock, port, count; struct
    hostent *host; struct
    sockaddr_in server_addr;
```

```
const char *url;

/* The number of received parameters is less than 3*/
if (argc < 3) {

        rt_kprintf("Usage: udpclient URL PORT [COUNT = 10]\n"); rt_kprintf("Like: tcpclient
        192.168.12.44 5000\n"); return ;

}

url = argv[1]; port =
strtoul(argv[2], 0, 10);

if (argc > 3) count
        = strtoul(argv[3], 0, 10);
else
        count = 10;

/* Get the host address through the function entry parameter url (if it is a domain name, domain name resolution will be performed) */
host = (struct hostent *) gethostbyname(url);

/* Create a socket, type is SOCK_DGRAM, UDP type*/ if ((sock =
socket(AF_INET, SOCK_DGRAM, 0)) == -1) {

        rt_kprintf("Socket error\n");
        return;
}

/* Initialize the pre-connected server address*/
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(port); server_addr.sin_addr
= *((struct in_addr *)host->h_addr); rt_memset(&(server_addr.sin_zero), 0,
sizeof(server_addr.sin_zero)) ;

/* Send count data in total*/
while (count) {

        /* Send data to the remote server*/
        sendto(sock, send_data, strlen(send_data), 0,
                    (struct sockaddr *)&server_addr, sizeof(struct sockaddr));

        /* Thread sleeps for a while*/
        rt_thread_delay(50);

        /*Decrement the count value*/
        count --;
```

```
    }

    /* Close this socket */
    closesocket(sock);
}
```

5.2.2. Operation results

Use the network debugging tool to build a UDP server on your computer and record the open ports.

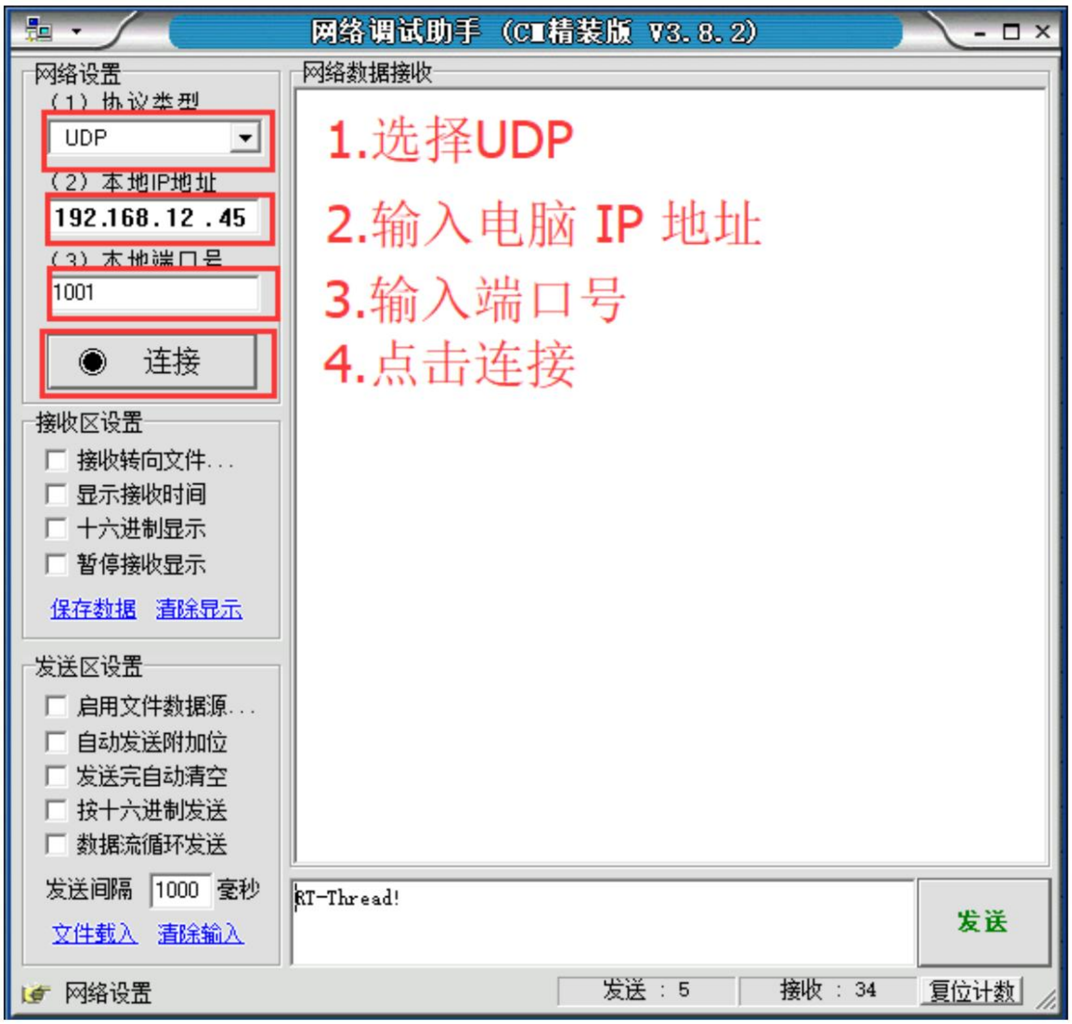

Figure **18:** *udp_set*

In the shell, enter the IP address of the udpclient PC and the port number you just recorded.

```
udpclient 192.168.12.45 1001
```

The server will receive 10 messages of This is UDP Client from RT-Thread.

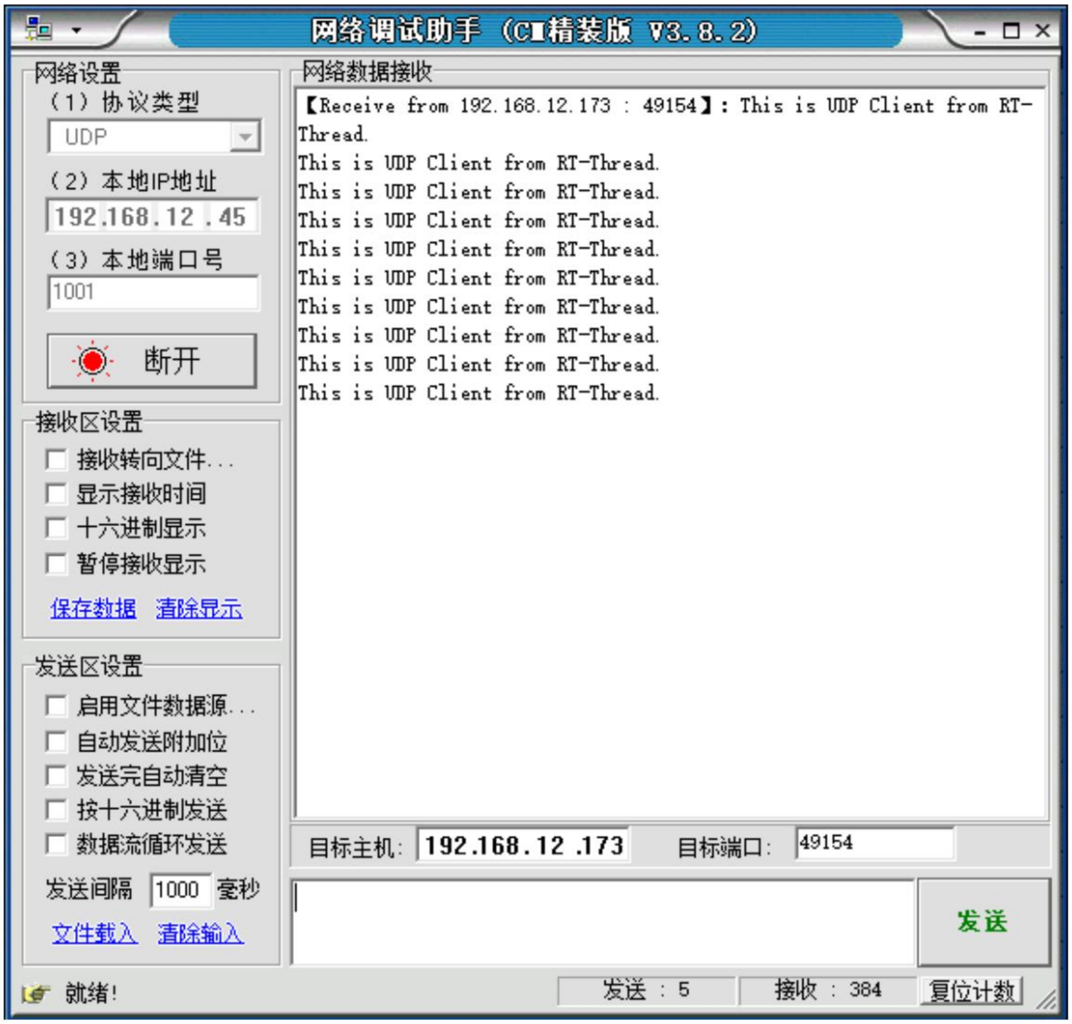

Figure **19:** *udpclient*

6 Advanced Applications

In order to facilitate the development of network applications, RT-Thread provides a rich set of network component packages, such as netutils network utility Toolset, webclient, cJSON, paho-mqtt, etc. Users can directly install env to use each component package, eliminating the need for porting and speeding up network application development.

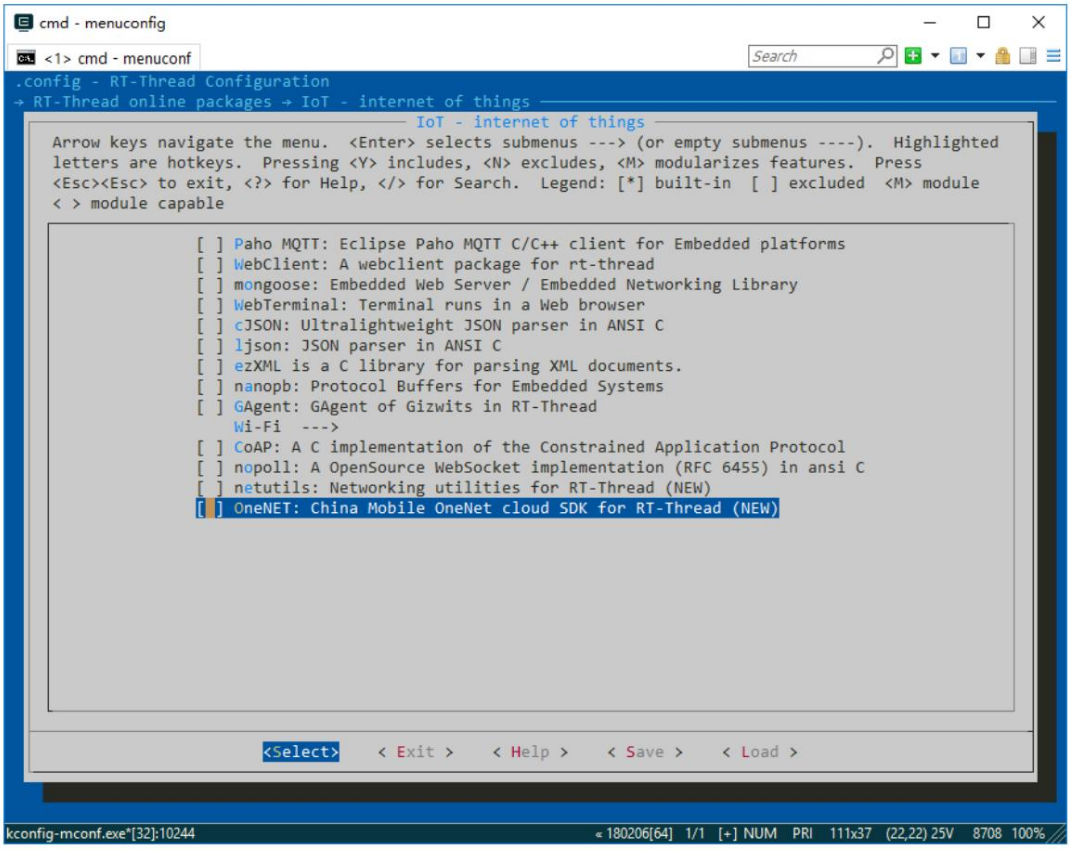

Figure **20:** *ENV_IoT*

Here we take the NTP (time synchronization) gadget and paho-mqtt in the netutils network gadget collection as examples. explain

6.1 NTP

NTP (Network Time Protocol) is a network time protocol, which is used to synchronize the time of each computer in the network.

RT-Thread implements the NTP client, which can obtain local time through the network and synchronize the RTC time of the board.

between.

For ENV configuration, refer to the ENV configuration in the previous preparation section.

Enter ntp_sync in msh to get the local time from the default NTP server (cn.ntp.org.cn). The default time zone is the Eastern Time Zone.

```
msh />ntp_sync
```

If you are prompted with a timeout or connection failure after entering ntp_sync , you can enter the NTP server after ntp_sync.

address, the program will get the time from the new server.

```
msh />ntp_sync edu.ntp.org.cn
```

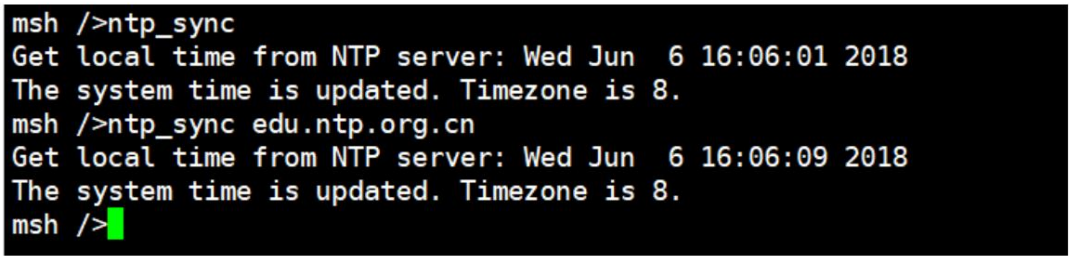

Figure **21:** *gettime*

6.2 MQTT

Paho MQTT It is a client of the MQTT protocol implemented in Eclipse. This package is in Eclipse paho-mqtt A set of MQTT client programs designed based on the source code package.

MQTT uses the publish/subscribe messaging model. When sending a message, you need to specify the topic name to which the message is sent. Before receiving messages, you need to subscribe to a topic name, and then you can receive the message content sent to this topic name.

RT-Thread MQTT client features:

• Automatic reconnection

after disconnection • Pipe model, non-blocking

API • Event callback

mechanism • TLS encrypted transmission

For ENV configuration, refer to the ENV configuration in the previous preparation section.

Enter the mq_start command in msh , the client will automatically connect to the server and subscribe to the /mqtt/test topic

```
msh />mq_start
```

The mq_pub command can be used to send messages to all clients subscribed to /mqtt/test. We use mq_pub to send RT-Thread!

```
msh />mq_pub RT-Thread!
```

Since we subscribed to the /mqtt/test topic before, the shell will soon display the RT-Thread! message sent by the server.

interest.

```
msh />mq_start
[MQTT] inter mqtt_connect_callback!
[MQTT] ipv4 address port: 1883
[MQTT] HOST = 'iot.eclipse.org'
msh />[MQTT] MQTT server connect success
[MQTT] Subscribe #0 /mqtt/test OK!
[MQTT] inter mqtt_online_callback!
msh />mq_pub RT-Thread!
msh />[MQTT] mqtt sub callback: /mqtt/test RT-Thread!
```

Figure **22:** *mqtttest*