EC200U-CN QuecOpen Socket Communication API Reference Manual

LTE Standard Module Series

Version: 1.0.0

Date: 2021-02-25

Status: Preliminary

www.quectel.com

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. For more information, please visit:
http://www.quectel.com/support/sales.htm.

For technical support, or to report documentation errors, please visit:
http://www.quectel.com/support/technical.htm
Or email to support@quectel.com.

General Notes

Quectel offers the information as a service to its customers. The information provided is based upon customers' requirements. Quectel makes every effort to ensure the quality of the information it makes available. Quectel does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information. All information supplied herein is subject to change without prior notice.

Disclaimer

While Quectel has made efforts to ensure that the functions and features under development are free from errors, it is possible that these functions and features could contain errors, inaccuracies and omissions. Unless otherwise provided by valid agreement, Quectel makes no warranties of any kind, implied or express, with respect to the use of features and functions under development. To the maximum extent permitted by law, Quectel excludes all liability for any loss or damage suffered in connection with the use of the functions and features under development, regardless of whether such loss or damage may have been foreseeable.

Duty of Confidentiality

The Receiving Party shall keep confidential all documentation and information provided by Quectel, except when the specific permission has been granted by Quectel. The Receiving Party shall not access or use Quectel's documentation and information for any purpose except as expressly provided herein. Furthermore, the Receiving Party shall not disclose any of the Quectel's documentation and information to any third party without the prior written consent by Quectel. For any noncompliance to the above requirements, unauthorized use, or other illegal or malicious use of the documentation and information, Quectel will reserve the right to take legal action.

Copyright

The information contained here is proprietary technical information of Quectel. Transmitting, reproducing, disseminating and editing this document as well as using the content without permission are forbidden. Offenders will be held liable for payment of damages. All rights are reserved in the event of a patent grant or registration of a utility model or design.

Copyright © Quectel Wireless Solutions Co., Ltd. 2021. All rights reserved.

About the Document

Revision History

Version	Date	Author	Description
-	2021-02-25	Larson LI	Creation of the document
1.0.0	2021-02-25	Larson LI	Preliminary

Contents

Table Index

Figure Index

1 Introduction

Quectel EC200U-CN module supports QuecOpen® solution. QuecOpen is an open-source embedded development platform based on Linux system, which is intended to simplify the design and development of IoT applications. For more information on QuecOpen, see ***document [1]***.

This document describes the operation process of socket communication under Quectel EC200U-CN QuecOpen® module.

To communicate with Quectel EC200U-CN QuecOpen® module, you need to go through three major processes: network registration, network activation, and socket communication:

- Network registration is executed automatically when the module is started without manual execution;
- Network activation is also the process of data call. see ***document [1]*** for details.
- Socket communication is detailed introduced in this document.

2 Socket Communication

When performing socket communication, pay attention to the following problems:

1. Before socket communication, make sure that the data call is successful. Use *ql_get_data_call_info()* to query whether the network channel that needs socket communication has performed the data call successfully.
2. When socket communication is performed after the data call, the network channel that needs to be communicated must be bound. No matter it is UDP or TCP, socket communication can be performed after binding the network channel.

2.1. Data Call and Socket Communication

Data call is required before socket communication. The data call process is shown in the figure below:

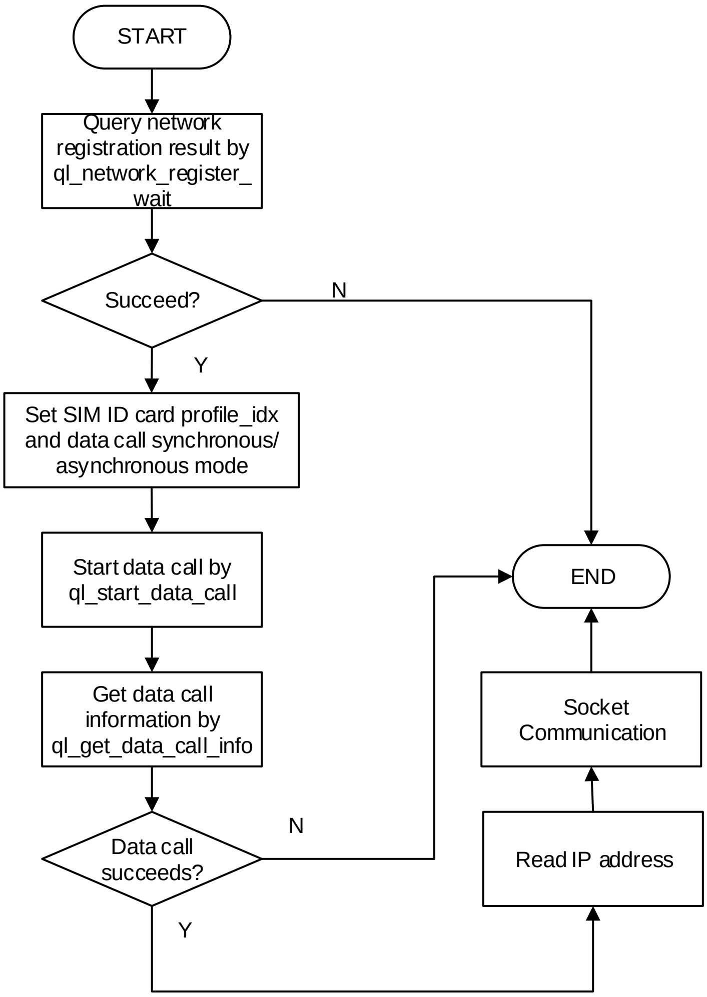

START
Query network registration result by ql_network_register_wait
Succeed?
N
Y
Set SIM ID card profile_idx and data call synchronous/ asynchronous mode
Start data call by ql_start_data_call
Get data call information by ql_get_data_call_info
Data call succeeds?
N
Y
Read IP address
Socket Communication
END

Figure 1: Data Call Process

2.2. Process of Socket Communication

The basic process of socket communication is shown in the figure below.

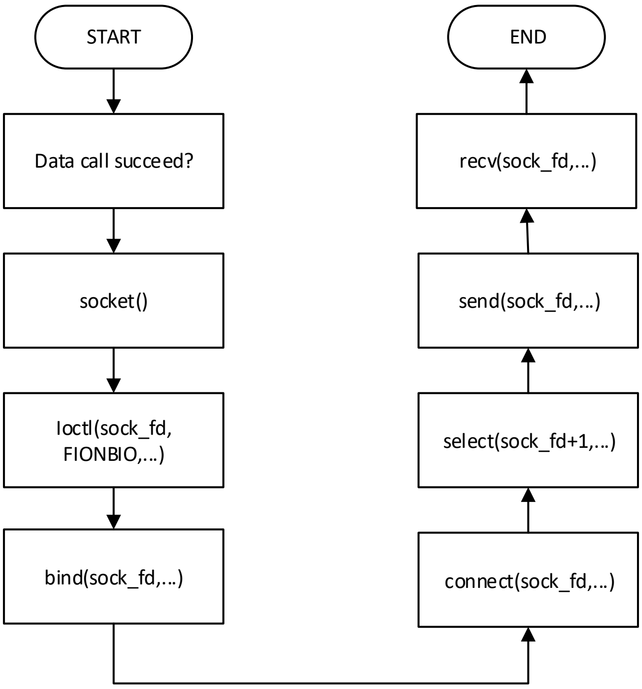


Figure 2: Socket Communication Process

2.3. Source Code Example of Socket Communication

2.3.1. Example of Data Call Process

The dialing process is as follows (take nSim = 0, profile_idx = 1 as an example):

```
QL_SOCKET_LOG("========== socket demo start ==========");
QL_SOCKET_LOG("wait for network register done");

while((ret = ql_network_register_wait(nSim, 120)) != 0 && i < 10){
        i++;
        ql_rtos_task_sleep_s(1);
}
if(ret == 0){
        i = 0;
        QL_SOCKET_LOG("====network registered!!!!====");
```

```
    }else{
        QL_SOCKET_LOG("====network register failure!!!!!====");
        goto exit;
    }

    ql_set_data_call_asyn_mode(nSim, profile_idx, 0);

    QL_SOCKET_LOG("===start data call====");
    ret=ql_start_data_call(nSim, profile_idx, QL_PDP_TYPE_IP, "uninet", NULL, NULL, 0);
    QL_SOCKET_LOG("===data call result:%d", ret);
    if(ret != 0){
        QL_SOCKET_LOG("====data call failure!!!!=====");
    }
    memset(&info, 0x00, sizeof(ql_data_call_info_s));

    ret = ql_get_data_call_info(nSim, profile_idx, &info);
    if(ret != 0){
        QL_SOCKET_LOG("ql_get_data_call_info ret: %d", ret);
        ql_stop_data_call(nSim, profile_idx);
        goto exit;
    }
    QL_SOCKET_LOG("info->profile_idx: %d", info.profile_idx);
    QL_SOCKET_LOG("info->ip_version: %d", info.ip_version);

    QL_SOCKET_LOG("info->v4.state: %d", info.v4.state);
    inet_ntop(AF_INET, &info.v4.addr.ip, ip4_addr_str, sizeof(ip4_addr_str));
    QL_SOCKET_LOG("info.v4.addr.ip: %s\r\n", ip4_addr_str);

    inet_ntop(AF_INET, &info.v4.addr.pri_dns, ip4_addr_str, sizeof(ip4_addr_str));
    QL_SOCKET_LOG("info.v4.addr.pri_dns: %s\r\n", ip4_addr_str);

    inet_ntop(AF_INET, &info.v4.addr.sec_dns, ip4_addr_str, sizeof(ip4_addr_str));
    QL_SOCKET_LOG("info.v4.addr.sec_dns: %s\r\n", ip4_addr_str);
```

2.3.2. Example of Socket Communication Process

2.3.2.1. Creating a Socket

```
ret = socket(AF_INET, SOCK_STREAM, 0);
    if(ret < 0)
    {
        printf("*** socket create fail ***\r\n");
        goto exit;
    }
sock_fd = ret;
```

2.3.2.2. Setting Socket to Blocking/Non-blocking Mode

```
ioctl(sock_fd, FIONBIO, &sock_nbio);
```

2.3.2.3. Binding to Local NIC

```
ip4_local_addr.sin_family = AF_INET;
    ip4_local_addr.sin_port = htons(ql_soc_generate_port());
    ip4_local_addr.sin_addr = ip4_addr;    //ip4_addr is the queried data call information. It can be
                                             assigned to this variable after being obtained through
                                             ql_get_data_call_info()

    ret = bind(sock_fd, (struct sockaddr *)&ip4_local_addr, sizeof(ip4_local_addr));
    if(ret < 0)
    {
        printf("*** bind fail ***\r\n");
        goto exit;
    }
```

NOTE

This step must be performed, otherwise the socket connection cannot be successfully established.

2.3.2.4. Establishing Socket Connection

```
ret = connect(sock_fd, (struct sockaddr *)ip4_svr_addr, sizeof(struct sockaddr));

    printf("connect ret: %d, errno: %u\r\n", ret, errno);

    if(ret == -1 && errno != EINPROGRESS)
    {
        printf("*** connect fail ***\r\n");
        goto exit;
    }
```

2.3.2.5. Monitoring Whether a Response from Server to Establish a Connection is Received

```
t.tv_sec = TCP_CONNECT_TIMEOUT_S;
t.tv_usec = 0;

FD_ZERO(&read_fds);
FD_ZERO(&write_fds);

FD_SET(sock_fd, &read_fds);
FD_SET(sock_fd, &write_fds);

ret = select(sock_fd + 1, &read_fds, &write_fds, NULL, &t);

printf("select ret: %d\r\n", ret);

if(ret <= 0)
{
    printf("*** select timeout or error ***\r\n");
    goto exit;
}

if(!FD_ISSET(sock_fd, &read_fds) && !FD_ISSET(sock_fd, &write_fds))
{
    printf("*** connect fail ***\r\n");
    goto exit;
}
else if(FD_ISSET(sock_fd, &read_fds) && FD_ISSET(sock_fd, &write_fds))
{
    optlen = sizeof(sock_error);
    ret = getsockopt(sock_fd, SOL_SOCKET, SO_ERROR, &sock_error, &optlen);
    if(ret == 0 && sock_error == 0)
    {
        printf("connect success\r\n");
    }
    else
    {
        printf("*** connect fail, sock_err = %d, errno = %u ***\r\n", sock_error, errno);
        goto exit;
    }
}
else if(!FD_ISSET(sock_fd, &read_fds) && FD_ISSET(sock_fd, &write_fds))
{
    printf("connect success\r\n");
}
```

```
    else if(FD_ISSET(sock_fd, &read_fds) && !FD_ISSET(sock_fd, &write_fds))
    {
        printf("*** connect fail ***\r\n");
        goto exit;
    }
    else
    {
        printf("*** connect fail ***\r\n");
        goto exit;
    }
```

2.3.2.6. Sending Data

```
ret = send(sock_fd, (const void*)TCP_CLIENT_SEND_STR, strlen(TCP_CLIENT_SEND_STR), 0);
    if(ret < 0)
    {
        printf("*** send fail ***\r\n");
        goto exit;
    }
```

2.3.2.7. Receiving Data Sent by Server

```
    t.tv_sec = TCP_RECV_TIMEOUT_S;
    t.tv_usec = 0;

    FD_ZERO(&read_fds);
    FD_SET(sock_fd, &read_fds);

    ret = select(sock_fd + 1, &read_fds, NULL, NULL, &t);

    printf("select ret: %d\r\n", ret);

    if(ret <= 0)
    {
        printf("*** select timeout or error ***\r\n");
        goto exit;
    }

    if(FD_ISSET(sock_fd, &read_fds))
    {
        ret = recv(sock_fd, recv_buf, sizeof(recv_buf), 0);
        if(ret > 0)
        {
            printf("recv data: [%d]%s\r\n", ret, recv_buf);
        }
        else if(ret == 0)
        {
            printf("*** peer closed ***\r\n");
            goto exit;
        }
        else
        {
            if(!(errno == EINTR || errno == EWOULDBLOCK || errno == EAGAIN))
            {
                printf("*** error occurs ***\r\n");
                goto exit;
            }
            else
            {
                printf("wait for a while\r\n");
                ql_rtos_task_sleep_ms(20);
                goto _recv_;
            }
        }
    }
```

2.3.2.8. Closing Socket Connection

```
close(sock_fd)
```

3 Socket Communication Data Structures and APIs

3.1. Data Structure

3.1.1. struct ql_data_call_info

The structure of data call saves the obtained data call information.

```
typedef struct
{
        int profile_idx;
        int ip_version;
        struct v4_info v4;
        struct v6_info v6;
}ql_data_call_info_s;
```

- **Parameter**

Type	Parameter	Description
int	*profile_idx*	PDP channel. Range: 1–7.
int	*ip_version*	IP address type obtained by the data call. Default: 0. 0: IPv4 1: IPv6 2: IPv4v6
v4_info	*v4*	Stores the data structure related to IPv4 information.
V6_info	*v6*	Stores the data structure related to IPv4 information.

3.1.2. struct sockaddr_in

The structure of socket address structure saves socket related information.

```
struct sockaddr_in {
    u8_t sin_len;
    u8_t sin_family;
    u16_t sin_port;
    struct in_addr sin_addr;
#define SIN_ZERO_LEN    8
    char sin_zero[SIN_ZERO_LEN];
};
```

- **Parameter**

Type	Parameter	Description
u8_t	*sin_len*	Length of the structure
u8_t	*sin_family*	The protocol family used by socket is as follows: AF_INET/AF_INET6/AF_UNSPEC/AF_UNIX
u16_t	*sin_port*	TCP/UDP port number
in_addr	*sin_addr*	IP address
char	*sin_zero*	unused

3.1.3. struct addrinfo

The structure of server address information saves relevant information of the socket server, and the saved information is used when connecting to the socket server.

```
struct addrinfo
{
        int                         ai_flags;          /* Input flags. */
        int                         ai_family;         /* Address family of socket. */
        int                         ai_socktype;       /* Socket type. */
        int                         ai_protocol;       /* Protocol of socket. */
        socklen_t                   ai_addrlen;        /* Length of socket address. */
        struct sockaddr    *ai_addr;                   /* Socket address of socket. */
        char                       *ai_canonname;    /* Canonical name of service location. */
        struct addrinfo    *ai_next;                  /* Pointer to next in list. */
};
```

- **Parameter**

Type	Parameter	Description
int	*ai_flags*	Input flags

int	*ai_family*	Address family of socket: AF_INET
int	*ai_socktype*	Socket type: SOCK_STREAM/SOCK_DGRAM
int	*ai_protocol*	Protocol of socket
socklen_t	*ai_addrlen*	Length of the buffer pointed to
struct sockaddr	*ai_addr*	Pointer to the *sockaddr* structure
char	*ai_canonname*	Canonical name of service location
struct addrinfo	*ai_next*	Pointer to next in list

3.1.4. struct v6_address_status

The structure of IPv6 server address status stores IPv6 addresses, DNS that has resolved IPv6 addresses, and resolves domain name when using IPv6 addresses for network communication.

```
struct v6_address_status
{
       struct in6_addr ip;              //IP address for IPv6
       struct in6_addr pri_dns;      //Primary DNS server address for IPv6
       struct in6_addr sec_dns;    //Secondary DNS server address for IPv6
};
```

- **Parameter**

Type	Parameter	Description
struct in6_addr	*ip*	IPv6 IP address
struct in6_addr	*pri_dns*	Primary DNS server address for IPv6
struct in6_addr	*sec_dns*	Secondary DNS server address for IPv6

3.1.5. struct timeval

After configuring the content of the timeval type variable, the time interval counter is equivalent to setting a timer, which can be used to set the waiting time.

```
struct timeval {
       long       tv_sec;          /* seconds */
       long       tv_usec; /* and microseconds */
};
```

- **Parameter**

Type	Parameter	Description
long	*tv_sec*	second
long	*tv_usec*	Microsecond

3.1.6. struct fd_set

This data structure is in the select mechanism. It is actually an array, each element of which can be associated with an open file handle. If the connection is established, once the content associated with the file handle changes, the select mechanism is used to monitor the content change (that is, the change of the content associated with the fd handle) and perform related actions. The variable of fd_set is used as a parameter of *select()*, which can be directly passed into the select mechanism after filling.

```
typedef struct fd_set {
   unsigned char fd_bits [(FD_SETSIZE * 2 + 7)/8];
} fd_set;
```

- **Parameter**

Type	Parameter	Description
unsigned char	fd_bits	File handle array

3.2. APIs

3.2.1. ql_start_data_call

This function starts a data call.

- **Prototype**

```
int ql_start_data_call(int profile_idx, int ip_version, char *apn_name, char *username, char *password, int auth_type)
```

- **Parameter**

profile_idx:
[In] PDP channel number. Range: 1–7.

ip_version:

[In] IP type

0	IPv4
1	IPv6
2	IPv4v6

apn_name:
[In] APN name

username:
[In] Username

password:
[In] Password

auth_type:
[In] Authentication type

0	NONE
1	PAP
2	CHAP
3	PAP or CHAP

- **Return Value**

0	This function is executed successfully.
-1	This function fails to be executed.

3.2.2. socket

This function creates the socket file descriptor fd.

- **Prototype**

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

- **Parameter**

domain:
[In] Address. Default: AF_INET. Values can be AF_INET or AF_INET6.

type:
[In] Socket type: SOCK_STREAM、SOCK_DRAM or SOCK_RAW

protocol:
[In] Protocol number. Usually it 0 and can be omitted.

- **Return Value**

File descriptor greater than 0	This function is executed successfully.
0 or less than 0	This function fails to be executed.

3.2.3. bind

This function binds local NIC.

- **Prototype**

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

- **Parameter**

s:
[In] Socket descriptor

name:
[In] Address information of the local NIC

namelen:
[In] Address length of the local NIC

- **Return Value**

0	This function is executed successfully.
Negative integer	This function fails to be executed.

3.2.4. connect

This function connects to the server.

- **Prototype**

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

- **Parameter**

s:
[In] Socket descriptor

name:
[In] Address information of the server

namelen:
[In] Address length of the server

- **Return Value**

0	This function is executed successfully.
Negative integer	This function fails to be executed.

3.2.5. send

This function sends socket data.

- **Prototype**

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

- **Parameter**

s:
[In] Socket descriptor

dataptr:
[In] First address of the data

size:
[In] Data length

flags:
[In] Flag bit, generally set to 0

- **Return Value**

0	This function is executed successfully.
Negative integer	This function fails to be executed.

3.2.6. recv

This function receives socket data.

- **Prototype**

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

- **Parameter**

s:
[In] Socket descriptor

mem:
[In] First address of the data buffer

len:
[In] Data length

flags:
[In] Flag bit, generally set to 0

- **Return Value**

0	This function is executed successfully.
Negative integer	This function fails to be executed.

3.2.7. close

This function closes the socket.

- **Prototype**

```
int close( int s)
```

- **Parameter**

s:
[In] Socket descriptor

- **Return Value**

0	This function is executed successfully.
Negative integer	This function fails to be executed.

4 Code Demo

4.1. Application Code Example (One Channel of Socket Communication)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "ql_type.h"
#include "ql_rtos.h"
#include "ql_application.h"
#include "ql_data_call.h"
#include "sockets.h"
#include "netdb.h"

#define TCP_SERVER_DOMAIN "220.180.239.212"
#define TCP_SERVER_PORT 8064
#define TCP_CONNECT_TIMEOUT_S 10
#define TCP_RECV_TIMEOUT_S 10
#define TCP_CLOSE_LINGER_TIME_S 10
#define TCP_CLIENT_SEND_STR "tcp client send string"
#define PROFILE_IDX 1

static struct in_addr ip4_addr = {0};

static void ql_nw_status_callback(int profile_idx, int nw_status)
{
    printf("profile(%d) status: %d\r\n", profile_idx, nw_status);
}

static void datacall_satrt(void)
{
    printf("wait for network register done\r\n");

    if(ql_network_register_wait(120) != 0)
    {
        printf("*** network register fail ***\r\n");
    }
    else
    {
        printf("doing network activing ...\r\n");

        ql_wan_start(ql_nw_status_callback);
        ql_set_auto_connect(1, TRUE);
        ql_start_data_call(1, 0, "3gnet.mnc001.mcc460.gprs", NULL, NULL, 0);
    }
}

static void do_tcp_client_test(void)
{
    int                 sock_nbio     = 1;
    int                 ret           = 0;
    int                 sock_fd       = -1;
    int                 sock_error   = 0;
    socklen_t           optlen = 0;
    fd_set              read_fds, write_fds;
    struct timeval t;
    struct addrinfo         * res, hints;
    struct sockaddr_in      * ip4_svr_addr;
    struct sockaddr_in      ip4_local_addr = {0};
    u8 dns_success = 0;
    u8 recv_buf[128] = {0};

    memset(&hints, 0, sizeof(struct addrinfo));

    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;

    if(getaddrinfo_with_pcid(TCP_SERVER_DOMAIN, NULL, &hints, &res, PROFILE_IDX) != 0)
    {
        printf("*** DNS fail ***\r\n");
        goto exit;
    }

    dns_success = 1;

    ret = socket(AF_INET, SOCK_STREAM, 0);
    if(ret < 0)
    {
        printf("*** socket create fail ***\r\n");
        goto exit;
    }
```

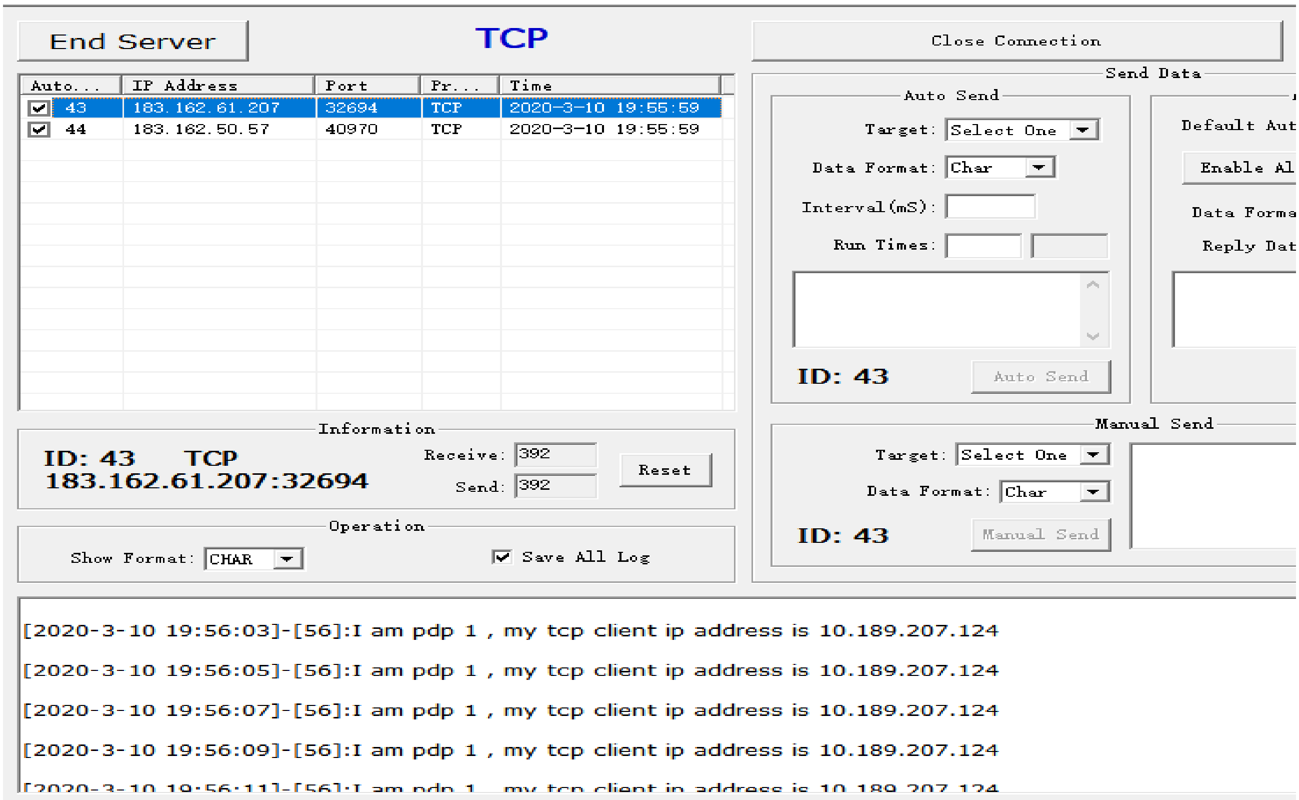
```
sock_fd = ret;

ioctl(sock_fd, FIONBIO, &sock_nbio);

ip4_local_addr.sin_family = AF_INET;
ip4_local_addr.sin_port = htons(ql_soc_generate_port());
ip4_local_addr.sin_addr = ip4_addr;

ret = bind(sock_fd, (struct sockaddr *)&ip4_local_addr, sizeof(ip4_local_addr));
if(ret < 0)
{
    printf("*** bind fail ***\r\n");
    goto exit;
}

ip4_svr_addr = (struct sockaddr_in *)res->ai_addr;
ip4_svr_addr->sin_port = htons(TCP_SERVER_PORT);

ret = connect(sock_fd, (struct sockaddr *)ip4_svr_addr, sizeof(struct sockaddr));

printf("connect ret: %d, errno: %u\r\n", ret, errno);

if(ret == -1 && errno != EINPROGRESS)
{
    printf("*** connect fail ***\r\n");
    goto exit;
}

t.tv_sec = TCP_CONNECT_TIMEOUT_S;
t.tv_usec = 0;

FD_ZERO(&read_fds);
FD_ZERO(&write_fds);

FD_SET(sock_fd, &read_fds);
FD_SET(sock_fd, &write_fds);

ret = select(sock_fd + 1, &read_fds, &write_fds, NULL, &t);

printf("select ret: %d\r\n", ret);

if(ret <= 0)
{
    printf("*** select timeout or error ***\r\n");
    goto exit;
}

if(!FD_ISSET(sock_fd, &read_fds) && !FD_ISSET(sock_fd, &write_fds))
{
    printf("*** connect fail ***\r\n");
    goto exit;
}
else if(FD_ISSET(sock_fd, &read_fds) && FD_ISSET(sock_fd, &write_fds))
{
    optlen = sizeof(sock_error);
    ret = getsockopt(sock_fd, SOL_SOCKET, SO_ERROR, &sock_error, &optlen);
    if(ret == 0 && sock_error == 0)
    {
        printf("connect success\r\n");
    }
    else
    {
        printf("*** connect fail, sock_err = %d, errno = %u ***\r\n", sock_error, errno);
        goto exit;
    }
}
else if(!FD_ISSET(sock_fd, &read_fds) && FD_ISSET(sock_fd, &write_fds))
{
    printf("connect success\r\n");
}
else if(FD_ISSET(sock_fd, &read_fds) && !FD_ISSET(sock_fd, &write_fds))
{
    printf("*** connect fail ***\r\n");
    goto exit;
}
else
{
    printf("*** connect fail ***\r\n");
    goto exit;
}

ret = send(sock_fd, (const void*)TCP_CLIENT_SEND_STR, strlen(TCP_CLIENT_SEND_STR), 0);
if(ret < 0)
{
    printf("*** send fail ***\r\n");
    goto exit;
}
```

```
_recv_:

    t.tv_sec = TCP_RECV_TIMEOUT_S;
    t.tv_usec = 0;

    FD_ZERO(&read_fds);
    FD_SET(sock_fd, &read_fds);

    ret = select(sock_fd + 1, &read_fds, NULL, NULL, &t);

    printf("select ret: %d\r\n", ret);

    if(ret <= 0)
    {
        printf("*** select timeout or error ***\r\n");
        goto exit;
    }

    if(FD_ISSET(sock_fd, &read_fds))
    {
        ret = recv(sock_fd, recv_buf, sizeof(recv_buf), 0);
        if(ret > 0)
        {
            printf("recv data: [%d]%s\r\n", ret, recv_buf);
        }
        else if(ret == 0)
        {
            printf("*** peer closed ***\r\n");
            goto exit;
        }
        else
        {
            if(!(errno == EINTR || errno == EWOULDBLOCK || errno == EAGAIN))
            {
                printf("*** error occurs ***\r\n");
                goto exit;
            }
            else
            {
                printf("wait for a while\r\n");
                ql_rtos_task_sleep_ms(20);
                goto _recv_;
            }
        }

    }

exit:

    if(dns_success) freeaddrinfo(res);

    if(sock_fd >= 0)
    {
        struct linger linger = {0};

        linger.l_onoff = 1;
        linger.l_linger = TCP_CLOSE_LINGER_TIME_S;

        setsockopt(sock_fd, SOL_SOCKET, SO_LINGER, &linger, sizeof(linger));
        setsockopt(sock_fd, IPPROTO_TCP, TCP_CLOSE_TIMEROUT, &linger.l_linger,
sizeof(linger.l_linger));

        close(sock_fd);
    }
}

static void sockets_tcp_client_test(void * argv)
{
    struct ql_data_call_info info = {0};
    char ip4_addr_str[16] = {0};

    printf("========== sockets tcp test will start ...\r\n");

    datacall_satrt();

    ql_get_data_call_info(1, 0, &info);

    printf("info.profile_idx: %d\r\n", info.profile_idx);
    printf("info.ip_version: %d\r\n", info.ip_version);
    printf("info.v4.state: %d\r\n", info.v4.state);
    printf("info.v4.reconnect: %d\r\n", info.v4.reconnect);

    inet_ntop(AF_INET, &info.v4.addr.ip, ip4_addr_str, sizeof(ip4_addr_str));
    printf("info.v4.addr.ip: %s\r\n", ip4_addr_str);

    inet_ntop(AF_INET, &info.v4.addr.pri_dns, ip4_addr_str, sizeof(ip4_addr_str));
    printf("info.v4.addr.pri_dns: %s\r\n", ip4_addr_str);
    inet_ntop(AF_INET, &info.v4.addr.sec_dns, ip4_addr_str, sizeof(ip4_addr_str));
    printf("info.v4.addr.sec_dns: %s\r\n", ip4_addr_str);

    ip4_addr = info.v4.addr.ip;

    if(info.v4.state)
    {
        do_tcp_client_test();
    }

    printf("========== sockets tcp test finished\r\n");
}

//application_init(sockets_tcp_client_test, "sockets_tcp_client_test", 4, 4);
```

4.2. Socket Communication Result Display (with Two Channels)

EC200U-CN QuecOpen module supports multi-channel socket communication. The following is the demonstration of two-channel socket communication. The essence of the two-channel socket is to establish two sockets for communication. The implementation of one-channel socket communication in the two-channel socket communication is the same as that of the one-channel socket communication.

1. Socket communication results on the server are as follows:



Auto...	IP Address	Port	Pr...	Time
☑ 43	183.162.61.207	32694	TCP	2020-3-10 19:55:59
☑ 44	183.162.50.57	40970	TCP	2020-3-10 19:55:59

ID: 43 TCP
183.162.61.207:32694

Receive: 392
Send: 392

Show Format: CHAR

☑ Save All Log

Target: Select One
Data Format: Char
Interval(mS):
Run Times:

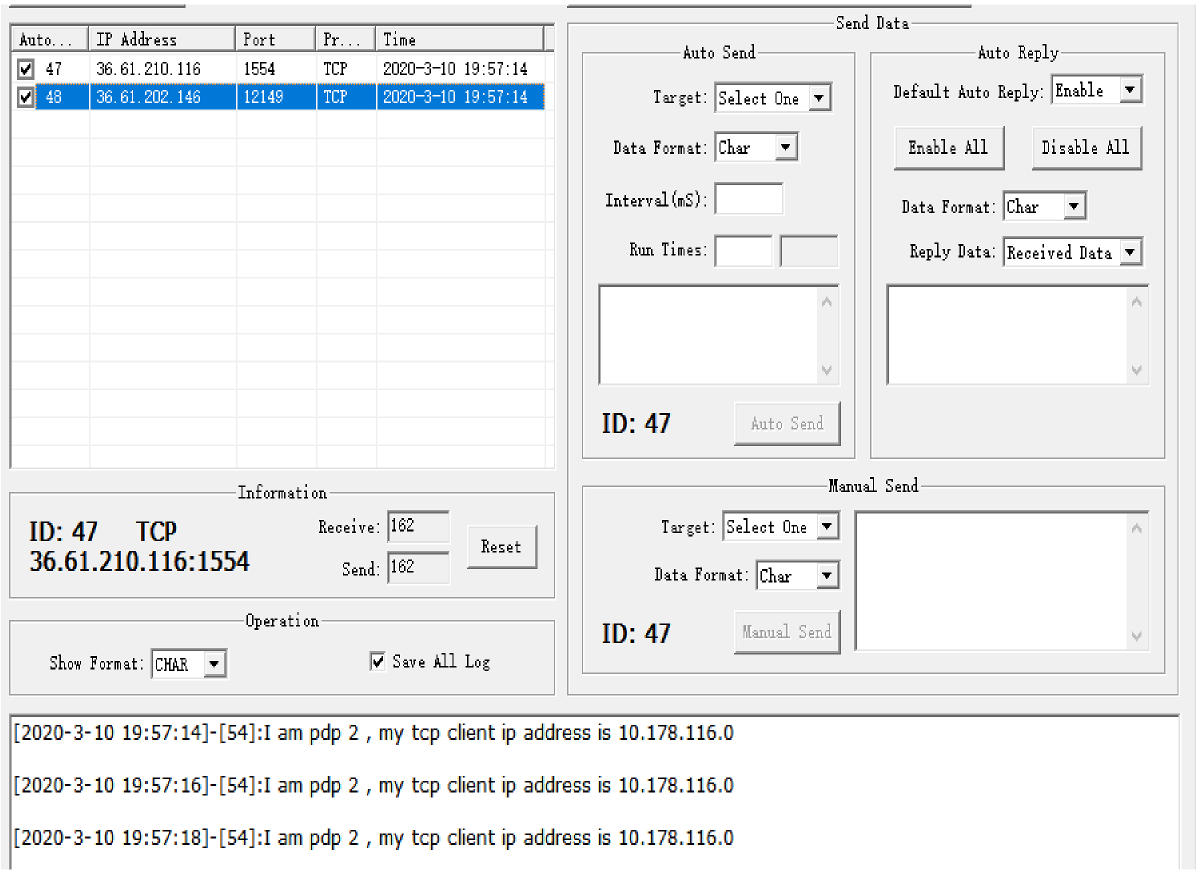
ID: 43

Target: Select One
Data Format: Char

ID: 43

```
[2020-3-10 19:56:03]-[56]:I am pdp 1 , my tcp client ip address is 10.189.207.124
[2020-3-10 19:56:05]-[56]:I am pdp 1 , my tcp client ip address is 10.189.207.124
[2020-3-10 19:56:07]-[56]:I am pdp 1 , my tcp client ip address is 10.189.207.124
[2020-3-10 19:56:09]-[56]:I am pdp 1 , my tcp client ip address is 10.189.207.124
[2020-3-10 19:56:11]-[56]:I am pdp 1 , my tcp client ip address is 10.189.207.124
```

Figure 3: Result of the First Socket Communication



Auto...	IP Address	Port	Pr...	Time
☑ 47	36.61.210.116	1554	TCP	2020-3-10 19:57:14
☑ 48	36.61.202.146	12149	TCP	2020-3-10 19:57:14

Information

ID: 47 TCP
36.61.210.116:1554

Receive: 162
Send: 162

Operation

Show Format: CHAR
☑ Save All Log

Send Data

Auto Send

Target: Select One
Data Format: Char
Interval(mS):
Run Times:
ID: 47

Auto Reply

Default Auto Reply: Enable
Data Format: Char
Reply Data: Received Data

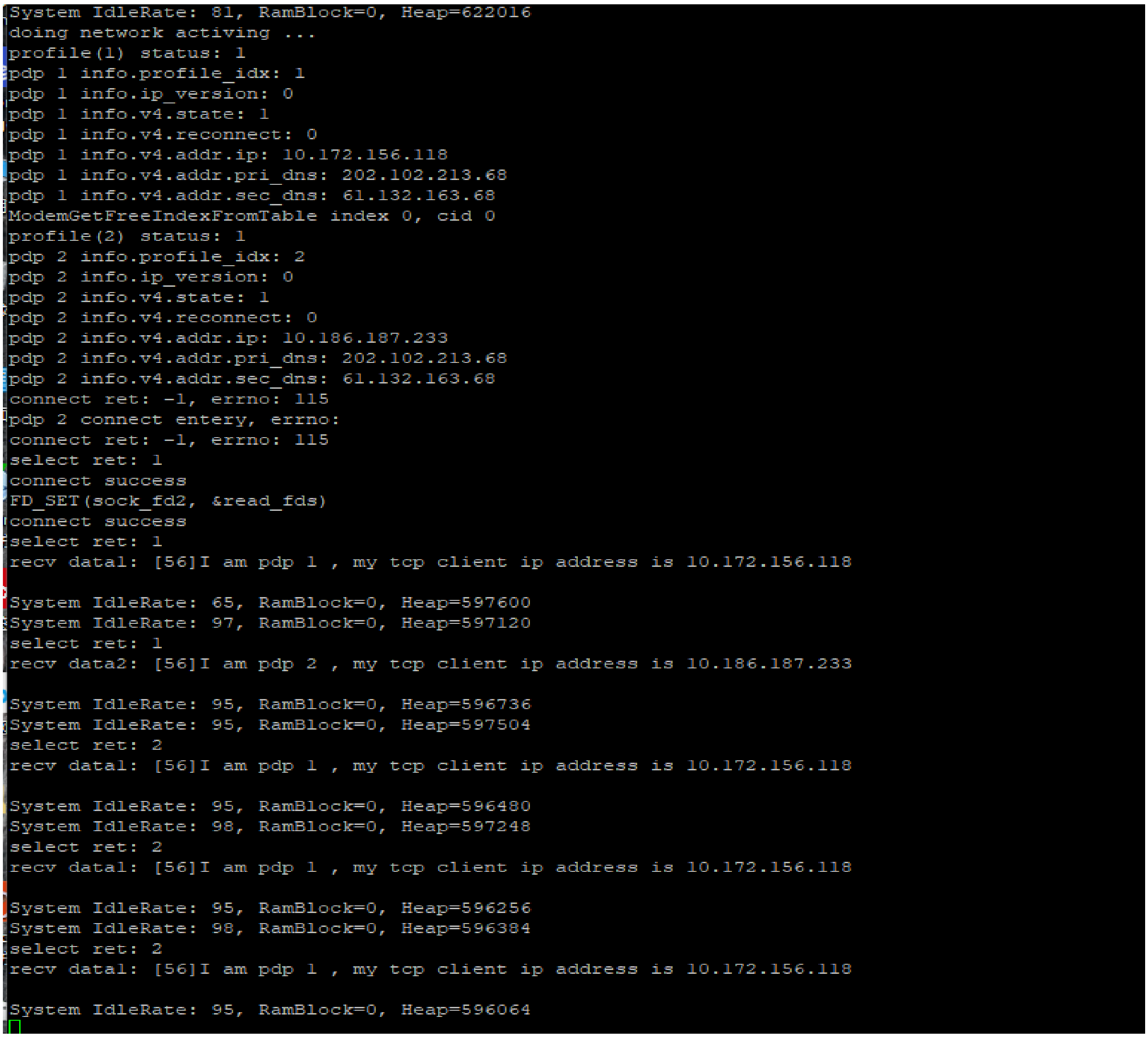
Manual Send

Target: Select One
Data Format: Char
ID: 47

```
[2020-3-10 19:57:14]-[54]:I am pdp 2 , my tcp client ip address is 10.178.116.0

[2020-3-10 19:57:16]-[54]:I am pdp 2 , my tcp client ip address is 10.178.116.0

[2020-3-10 19:57:18]-[54]:I am pdp 2 , my tcp client ip address is 10.178.116.0
```

Figure 4: Result of the Second Socket Communication

2. Socket communication logs on the module are as follows:



```
System IdleRate: 81, RamBlock=0, Heap=622016
doing network activing ...
profile(1) status: 1
pdp 1 info.profile_idx: 1
pdp 1 info.ip_version: 0
pdp 1 info.v4.state: 1
pdp 1 info.v4.reconnect: 0
pdp 1 info.v4.addr.ip: 10.172.156.118
pdp 1 info.v4.addr.pri_dns: 202.102.213.68
pdp 1 info.v4.addr.sec_dns: 61.132.163.68
ModemGetFreeIndexFromTable index 0, cid 0
profile(2) status: 1
pdp 2 info.profile_idx: 2
pdp 2 info.ip_version: 0
pdp 2 info.v4.state: 1
pdp 2 info.v4.reconnect: 0
pdp 2 info.v4.addr.ip: 10.186.187.233
pdp 2 info.v4.addr.pri_dns: 202.102.213.68
pdp 2 info.v4.addr.sec_dns: 61.132.163.68
connect ret: -1, errno: 115
pdp 2 connect entery, errno:
connect ret: -1, errno: 115
select ret: 1
connect success
FD_SET(sock_fd2, &read_fds)
connect success
select ret: 1
recv data1: [56]I am pdp 1 , my tcp client ip address is 10.172.156.118

System IdleRate: 65, RamBlock=0, Heap=597600
System IdleRate: 97, RamBlock=0, Heap=597120
select ret: 1
recv data2: [56]I am pdp 2 , my tcp client ip address is 10.186.187.233

System IdleRate: 95, RamBlock=0, Heap=596736
System IdleRate: 95, RamBlock=0, Heap=597504
select ret: 2
recv data1: [56]I am pdp 1 , my tcp client ip address is 10.172.156.118

System IdleRate: 95, RamBlock=0, Heap=596480
System IdleRate: 98, RamBlock=0, Heap=597248
select ret: 2
recv data1: [56]I am pdp 1 , my tcp client ip address is 10.172.156.118

System IdleRate: 95, RamBlock=0, Heap=596256
System IdleRate: 98, RamBlock=0, Heap=596384
select ret: 2
recv data1: [56]I am pdp 1 , my tcp client ip address is 10.172.156.118

System IdleRate: 95, RamBlock=0, Heap=596064
```

5 Appendix A References

Table 1: Related Documents

SN	Document Name	Remark
[1]	Quectel_EC200U-CN_QuecOpen_Quick_Start_Guide	Quick start guide for EC200U-CN QuecOpen module
[2]	Quectel_EC200U-CN_QuecOpen_Data_Call_API_Reference_Manual	EC200U-CN QuecOpen data call API reference manual

Table 2: Term and Abbreviation

Abbreviation	Description
API	Application Program Interface
DNS	Domain Name System
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
TCP	Transmission Control Protocol
UDP	User Datagram Protocol