FCM360W QuecOpen Network User Guide

Wi-Fi&Bluetooth Module Series

Version: 1.0.0

Date: 2023-03-17

Status: Preliminary

At Quectel, our aim is to provide timely and comprehensive services to our customers. If you require any assistance, please contact our 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 offices. 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 us at: support@quectel.com.

Legal Notices

We offer information as a service to you. The provided information is based on your requirements and we make every effort to ensure its quality. You agree that you are responsible for using independent analysis and evaluation in designing intended products, and we provide reference designs for illustrative purposes only. Before using any hardware, software or service guided by this document, please read this notice carefully. Even though we employ commercially reasonable efforts to provide the best possible experience, you hereby acknowledge and agree that this document and related services hereunder are provided to you on an “as available” basis. We may revise or restate this document from time to time at our sole discretion without any prior notice to you.

Use and Disclosure Restrictions

License Agreements

Documents and information provided by us shall be kept confidential, unless specific permission is granted. They shall not be accessed or used for any purpose except as expressly provided herein.

Copyright

Our and third-party products hereunder may contain copyrighted material. Such copyrighted material shall not be copied, reproduced, distributed, merged, published, translated, or modified without prior written consent. We and the third party have exclusive rights over copyrighted material. No license shall be granted or conveyed under any patents, copyrights, trademarks, or service mark rights. To avoid ambiguities, purchasing in any form cannot be deemed as granting a license other than the normal non-exclusive, royalty-free license to use the material. We reserve the right to take legal action for noncompliance with abovementioned requirements, unauthorized use, or other illegal or malicious use of the material.

Trademarks

Except as otherwise set forth herein, nothing in this document shall be construed as conferring any rights to use any trademark, trade name or name, abbreviation, or counterfeit product thereof owned by Quectel or any third party in advertising, publicity, or other aspects.

Third-Party Rights

This document may refer to hardware, software and/or documentation owned by one or more third parties ("third-party materials"). Use of such third-party materials shall be governed by all restrictions and obligations applicable thereto.

We make no warranty or representation, either express or implied, regarding the third-party materials, including but not limited to any implied or statutory, warranties of merchantability or fitness for a particular purpose, quiet enjoyment, system integration, information accuracy, and non-infringement of any third-party intellectual property rights with regard to the licensed technology or use thereof. Nothing herein constitutes a representation or warranty by us to either develop, enhance, modify, distribute, market, sell, offer for sale, or otherwise maintain production of any our products or any other hardware, software, device, tool, information, or product. We moreover disclaim any and all warranties arising from the course of dealing or usage of trade.

Privacy Policy

To implement module functionality, certain device data are uploaded to Quectel's or third-party's servers, including carriers, chipset suppliers or customer-designated servers. Quectel, strictly abiding by the relevant laws and regulations, shall retain, use, disclose or otherwise process relevant data for the purpose of performing the service only or as permitted by applicable laws. Before data interaction with third parties, please be informed of their privacy and data security policy.

Disclaimer

a) We acknowledge no liability for any injury or damage arising from the reliance upon the information.
b) We shall bear no liability resulting from any inaccuracies or omissions, or from the use of the information contained herein.
c) While we have made every effort to ensure that the functions and features under development are free from errors, it is possible that they could contain errors, inaccuracies, and omissions. Unless otherwise provided by valid agreement, we make no warranties of any kind, either implied or express, and exclude all liability for any loss or damage suffered in connection with the use of features and functions under development, to the maximum extent permitted by law, regardless of whether such loss or damage may have been foreseeable.
d) We are not responsible for the accessibility, safety, accuracy, availability, legality, or completeness of information, advertising, commercial offers, products, services, and materials on third-party websites and third-party resources.

About the Document

Revision History

Version	Date	Author	Description
-	2023-03-17	Lei WANG	Creation of the document
1.0.0	2023-03-17	Lei WANG	Preliminary

Contents

Table Index

Figure Index

1 Introduction

Quectel FCM360W module supports QuecOpen® solution. QuecOpen® is an embedded development platform based on RTOS system, which is intended to simplify the design and development of IoT applications. For more information on QuecOpen®, see ***document [1]***。

This document introduces how to use the Quectel FCM360W QuecOpen® module to perform socket communication, MQTT function, TLS/SSL function and WebSocket function.

2 Socket Communication

2.1. Description

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

- Before performing socket communication, make sure that the module has been connected to a network (Wi-Fi) successfully.
- When performing socket communication after the module is successfully connected to a network, no matter it is UDP or TCP, the network channel used for communication must be bound. If socket is used as a server, socket communication can be performed after binding the network channel.

2.1.1. Process of Socket Communication

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

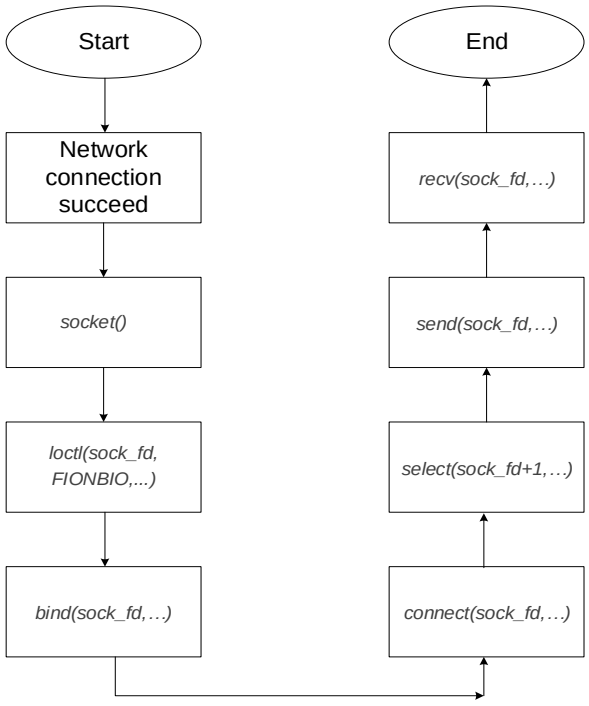

Figure 1: Socket Communication Process

2.1.2. Source Code Example of Socket Communication

2.1.2.1. Creating a Socket

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

2.1.2.2. Setting Socket to Blocking/Non-blocking Mode

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

2.1.2.3. Binding to Local Network Interface

```
ip4_local_addr.sin_family = AF_INET;
ip4_local_addr.sin_port = htons(2023);
ip4_local_addr.sin_addr.s_addr = INADDR_ANY;

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

NOTE

This step must be performed when socket is used as a sever, otherwise the socket connection cannot be successfully established.

2.1.2.4. Establishing a Socket Connection

```
ret = connect(sock_fd, (struct sockaddr *) &server_addr, sizeof(server_addr));if(ret < 0)
{
  ql_debug( "connect err: %d\r\n", err );
  goto exit;
}
```

2.1.2.5. 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);

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

if(ret <= 0)
{
    ql_debug("*** select timeout or error ***\r\n");
    goto exit;
}
if(!FD_ISSET(sock_fd, &read_fds) && !FD_ISSET(sock_fd, &write_fds))
{
    ql_debug("*** 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)
    {
        ql_debug("connect success\r\n");
    }
    else
```

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

2.1.2.6. Sending Data

```
ret = send(sock_fd, (const void*)TCP_CLIENT_SEND_STR, strlen(TCP_CLIENT_SEND_STR),0);

if(ret < 0)
{
        ql_debug("*** send fail ***\r\n");
        goto exit;
}
```

2.1.2.7. Receiving Data Sent by Server

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

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

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

if(ret <= 0)
{
```

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

2.1.2.8. Closing Socket Connection

```
close(sock_fd)
```

2.2. Socket API

2.2.1. Header File

sockets.h, the header file of Socket API, is in the *ql_components/third_party/lwip/lwip-2.1.0/src/include/lwip* directory. Unless otherwise specified, all header files mentioned in this document are in this directory.

2.2.2. Structure Type

2.2.2.1. struct sockaddr_in

The structure of socket address is used to store socket related information and is defined below:

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

- **Parameter**

Type	Parameter	Description
u8_t	*sin_len*	Length of the structure of socket address. Unit: byte.
sa_family_t	*sin_family*	The protocol family used by socket is below: AF_INET/AF_INET6/AF_UNSPEC/AF_UNIX.
u16_t	*sin_port*	TCP/UDP port number.
struct in_addr	*sin_addr*	IP address.
char	*sin_zero*	Unused.

2.2.2.2. struct addrinfo

The structure of server address information stores relevant information of the socket server, and the stored information is used when connecting to the socket server. The structure is defined below:

```
struct addrinfo
{
    int ai_flags;
    int ai_family;
    int ai_socktype;
    int ai_protocol;
    socklen_t ai_addrlen;
    struct sockaddr *ai_addr;
    char *ai_canonname;
    struct addrinfo *ai_next;
}
```

- **Parameter**

Type	Parameter	Description
int	*ai_flags*	Processing mode of addresses and domain names.
int	*ai_family*	Address family of socket: AF_INET.
int	*ai_socktype*	Socket type: SOCK_STREAM/SOCK_DGRAM.
int	*ai_protocol*	Socket protocol.
socklen_t	*ai_addrlen*	Length of the buffer pointed to. Unit: byte.
struct sockaddr	*ai_addr*	Pointer to the *sockaddr* structure.
char	*ai_canonname*	Canonical name of the host.
struct addrinfo	*ai_next*	Pointer to the next structure in list.

2.2.2.3. struct timeval

The structure of waiting time is used to set the waiting time for the function execution timeout, and is defined below:

```
struct timeval
{
   long tv_sec;
   long tv_usec;
}
```

- **Parameter**

Type	Parameter	Description
long	*tv_sec*	Second.
long	*tv_usec*	Microsecond.

2.2.2.4. 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 can monitor the content change (that is, the change of the content associated with the *fd* handle) and perform related actions. The structure of *fd_set* is used as a parameter of *select()*, which can be directly passed into the *select* mechanism after inputting a value. The structure is defined below:

```
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

2.2.3. API Description

2.2.3.1. socket

This function creates the socket descriptor *fd*.

- **Prototype**

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

- **Parameter**

domain:
[In] Protocol family, the value can be AF_INET or AF_INET6. Default: AF_INET.

type:
[In] Socket type, the value can be SOCK_STREAM, SOCK_DRAM or SOCK_RAW.

protocol:
[In] Protocol number. This value is usually 0 and can be omitted.

- **Return Value**

File descriptor greater than *0*	Successful execution
0 or less than *0*	Failed execution

2.2.3.2. bind

This function binds local network interface.

- **Prototype**

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

- **Parameter**

s:
[In] Socket descriptor, created by the socket(), represents the unique identifier of a socket.

name:
[In] Address information of the local network interface.

namelen:
[In] Address length of the local network interface. Unit: byte.

- **Return Value**

0	Successful execution
Negative integer	Failed execution

2.2.3.3. connect

This function connects to the server.

- **Prototype**

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

- **Parameter**

s:
[In] Socket descriptor, created by *socket()*, represents the unique identifier of a socket.

name:
[In] IP address information of the server.

namelen:
[In] The length of the server IP address. Unit: byte.

- **Return Value**

0	Successful execution
Negative integer	Failed execution

2.2.3.4. send

This function sends socket data.

- **Prototype**

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

- **Parameter**

s:
[In] Socket descriptor, created by *socket()*, represents the unique identifier of a socket.

dataptr:
[In] The starting address of the socket data.

size:
[In] The length of the socket data. Unit: byte.

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

- **Return Value**

0	Successful execution
Negative integer	Failed execution

2.2.3.5. recv

This function receives socket data.

- **Prototype**

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

- **Parameter**

s:
[In] Socket descriptor, created by *socket()*, represents the unique identifier of a socket.

mem:
[In] The starting address of the data buffer.

len:
[In] The length of the socket data. Unit: byte.

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

- **Return Value**

0	Successful execution
Negative integer	Failed execution

2.2.3.6. close

This function closes the socket connection.

- **Prototype**

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

- **Parameter**

s:
[In] Socket descriptor, created by the socket(), represents the unique identifier of a socket.

- **Return Value**

0	Successful execution
Negative integer	Failed execution

2.2.4. Application Code Example

The application code example of socket programming TCP client is provided in the SDK of the module, and the test related function example is in the *ql_sockets_demo.c* file under the *ql_application* directory. The description of related functions can be seen below:

- *ql_cmd_tcp_cont_test()*: Create a TCP client task, this function needs to be called to run the socket example;
- *do_tcp_client_thread()*: The execution function of the TCP client task, which implements simple sending and receiving of socket data.

Use the serial port tool to execute **ql_tcp_cont [ip] [prot]** through serial port 2 to create a TCP client task.

The module supports multi-channel socket communication, the following is an example of one-channel socket communication. The specific code is below:

```
#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

void do_tcp_client_thread(beken_thread_arg_t arg )
{
    int ret = 0;
    int sock_fd = -1;
    int sock_error = 0;
    socklen_t optlen = 0;
    int sock_nbio = 0;

    fd_set read_fds, write_fds;
    struct timeval t;

    u8 recv_buf[128] = {0};


  OSStatus err = kNoErr;
  struct ip_port_msg *i_pMsg;
  struct sockaddr_in server_addr;


    struct sockaddr_in ip4_local_addr = {0};
```

```
i_pMsg = (struct ip_port_msg *)arg;
if(i_pMsg == NULL)
{
    ql_debug( "arg err!\r\n" );
    goto exit;
}

ret = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP);
   if(ret < 0)
   {
     ql_debug("*** 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(1234);
   ip4_local_addr.sin_addr.s_addr = INADDR_ANY;

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

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = i_pMsg->IP.addr;    //0x0728a8c0 /*192.168.40.7*/
server_addr.sin_port = htons(i_pMsg->sin_port);   //htons( 12345 );


ret = connect(sock_fd, (struct sockaddr *) &server_addr, sizeof(server_addr));
if(ret < 0)
{
    ql_debug( "connect err: %d\r\n", err );
    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);

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

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

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

```
        ql_debug("*** 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)
    {
      ql_debug("*** send fail ***\r\n");
      goto exit;
     }

_recv_:


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

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

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

    if(ret <= 0)
    {
        ql_debug("*** 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)
      {
         ql_debug("recv data: [%d]%s\r\n", ret, recv_buf);
      }
      else if(ret == 0)
        {
            ql_debug("*** peer closed ***\r\n");
            goto exit;
        }
        else
        {
            if(!(errno == EINTR || errno == EWOULDBLOCK || errno == EAGAIN))
            {
```

```
                    ql_debug("*** error occurs ***\r\n");
                    goto exit;
                }
                else
                {
                    ql_debug("wait for a while\r\n");
                    rtos_delay_milliseconds(20);
                    goto _recv_;
                }
            }
        }

exit:
            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_KEEPALIVE, &linger.l_linger,
sizeof(linger.l_linger));
                close(sock_fd);
            }

    rtos_delete_thread( NULL );
}

        static int ql_cmd_tcp_cont_test(cmd_tbl_t *t, int argc, char *argv[])
        {
            struct ip_port_msg *i_pMsg = NULL;
            int port;
            i_pMsg = (struct ip_port_msg *)os_malloc(sizeof(struct ip_port_msg));
            if (!inet_aton(argv[1], &(i_pMsg->IP.addr)))
            {
                ql_debug("inet_aton Failed\r\n");
                goto exit;
            }
            port = (int)strtoul(argv[2], NULL, 0);
            i_pMsg->sin_port = port;
            int ret = ql_rtos_task_create(socket_test_thread_handle,
                                          (unsigned short)1024 * 4,
                                          RTOS_HIGHER_PRIORTIY_THAN(3),
                                          "connt_Server",
```

```
                                        do_tcp_client_thread,
                                        (void *)i_pMsg);
        if (ret != TRUE)
        {
            os_printf(LM_OS, LL_INFO, "Error: Failed to create at_ssl thread: %d\r\n", ret);
            return CMD_RET_FAILURE;
        }
exit:
        if (i_pMsg)
        {
            os_free(i_pMsg);
        }
        return CMD_RET_SUCCESS;
}
    port = num_string2uint16num(argv[2]);
    if(port < 0)
    {
        ql_debug("port err:%d\r\n", port);
    }

    i_pMsg->sin_port = port;

    err = rtos_create_thread( NULL,
                        BEKEN_APPLICATION_PRIORITY,
                        "connt_Server",
                        do_tcp_client_thread,
                        0x600,
                        i_pMsg );
    if(kNoErr != err)
    {
        ql_debug("rtos_create_thread Failed(%d)\r\n", err);
        goto exit;
    }

    return;

exit:

    if(i_pMsg)
    {
        os_free(i_pMsg);
    }
}
```

3 MQTT

3.1. Overview

3.1.1. The Introduction of MQTT

MQTT is a message publish/subscribe transport protocol which is based on client-server. The MQTT protocol is lightweight, simple, open and easy to implement, which makes it widely applicable. Including constrained environments, such as Machine-to-Machine (M2M) communication and the Internet of Things (IoT).

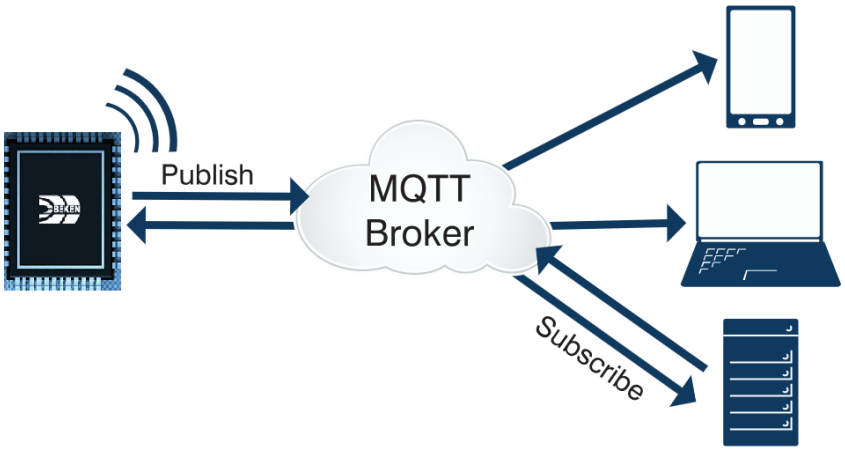

Figure 2: MQTT Application

MQTT has three kinds of message publishing qualities of service:

- "At most once": Message publishing is completely dependent on the underlying TCP/IP network, which may occur message loss or duplication. This method is mainly used for the push of ordinary APPs. If the user's smart device is not connected to the network when the message is pushed, and the pushed message has not been received, even if it is connected to the network again, the pushed message cannot be received.
- "At least once": Message arrival is guaranteed, but message duplication may occur.
- "Only Once": Make sure the message arrives once. This level can be used in some more demanding billing systems. In billing systems, message loss or duplication can lead to wrong results. This highest-quality message publishing service can also be used to push instant messaging Apps to ensure that

users receive messages and only receive them once.

3.1.2. MQTT Client

An application or a device that uses the MQTT protocol can always establish a network connection to the server. The client can realize the following functions:

- Publish information that other clients may subscribe to;
- Subscribe to messages published by other clients;
- Unsubscribe or delete messages from an application;
- Disconnect from the server.

3.2. MQTT API

3.2.1. Header File

mqtt_client.h, the header file of MQTT API, is in the *ql_components/third_party/mqtt* directory. Unless otherwise specified, all header files mentioned in this document are in this directory.

3.2.2. API Description

3.2.2.1. trs_mqtt_client_init

This function initializes the MQTT context. Before creating the MQTT connection, define and initialize an *mqtt_client.*

- **Prototype**

```
trs_mqtt_client_handle_t trs_mqtt_client_init(const trs_mqtt_client_config_t *config)
```

- **Parameter**

config:
[In] MQTT context to be initialized. See *mqtt_client.h* for details.

- **Return Value**

trs_mqtt_client_handle_t	Handle for operating the MQTT client context
The return value is void	Failed execution
The return value is not void	Successful execution

3.2.2.2. trs_mqtt_client_set_uri

This function sets the URI for MQTT client connection.

- **Prototype**

```
int trs_mqtt_client_set_uri(trs_mqtt_client_handle_t client, const char *uri)
```

- **Parameter**

client:
[In] The handle of the MQTT client context. See *mqtt_client.h* for details.

uri:
[In] The connected URI.

- **Return Value**

0	Successful execution
Other Values	Failed execution

3.2.2.3. trs_mqtt_client_start

This function creates the server for MQTT client connection.

- **Prototype**

```
int trs_mqtt_client_start(trs_mqtt_client_handle_t client)
```

- **Parameter**

client:
[In] The handle of MQTT context. See *mqtt_client.h* for details.

- **Return Value**

0	Successful execution
Other Values	Failed execution

3.2.2.4. trs_mqtt_client_stop

This function disconnects the MQTT client from the server.

- **Prototype**

```
int trs_mqtt_client_stop(trs_mqtt_client_handle_t client)
```

- **Parameter**

client:
[In] MQTT context. See *mqtt_client.h* for details.

- **Return Value**

0	Successful execution
Other Values	Failed execution

3.2.2.5. trs_mqtt_client_subscribe

This function subscribes the topic message of MQTT client.

- **Prototype**

```
int trs_mqtt_client_subscribe(trs_mqtt_client_handle_t client, const char *topic, int qos)
```

- **Parameter**

client:
[In] MQTT context. See *mqtt_client.h* for details.

topic:
[In] MQTT server topic. See *mqtt_client.h* for details.

qos:
[In] MQTT message service quality. See *mqtt_client.h* for details.

- **Return Value**

0	Successful execution
Other Values	Failed execution

3.2.2.6. trs_mqtt_client_unsubscribe

This function unsubscribes the topic message of MQTT client.

- **Prototype**

```
int trs_mqtt_client_unsubscribe(trs_mqtt_client_handle_t client, const char *topic)
```

- **Parameter**

client:
[In] MQTT context. See *mqtt_client.h* for details.

topic:
[In] MQTT topic. See *mqtt_client.h* for details.

- **Return Value**

0	Successful execution
Other Values	Failed execution

3.2.2.7. trs_mqtt_client_publish

This function publishes the message.

- **Prototype**

```
int trs_mqtt_client_publish(trs_mqtt_client_handle_t client, const char *topic, const char *data, int len,
int qos, int retain)
```

- **Parameter**

client:
[In] MQTT context. See *mqtt_client.h* for details.

topic:
[In] Topic of the published message.

data:
[In] Content of the published topic.

len:
[In] Size of the topic content.

qos:
[In] QoS level of the published message.

retain:
[In] Reserved.

- **Return Value**

0	Successful execution
Other Values	Failed execution

3.2.2.8. trs_mqtt_client_destroy

This function destroys the handle of the MQTT context.

- **Prototype**

```
int trs_mqtt_client_destroy(trs_mqtt_client_handle_t client)
```

- **Parameter**

client:
[In] The handle of the MQTT context. See *mqtt_client.h* for details.

- **Return Value**

0	Successful execution
Other Values	Failed execution

3.2.2.9. set_mqtt_msg

This function sends the MQTT message for internal communication.

- **Prototype**

```
int set_mqtt_msg(SET_TYPE_EN type, MQTT_CFG_MSG_ST *msg, const char *data, int data_len)
```

- **Parameter**

type:
[In] MQTT message type. See ***Chapter 3.2.2.9.1*** for details.

msg:
[In] MQTT client type. See ***Chapter 3.2.2.9.2*** for details.

data:
[In] Content of the published message.

data_len:
[In] Size of the published message.

- **Return Value**

0	Successful execution
Other Values	Failed execution

3.2.2.9.1. SET_TYPE_EN

This enumeration is the type that defines the message. The definition is below:

```
typedef enum
{
   SET_TOPIC = 1,
   SET_DATA,
}SET_TYPE_EN;
```

- **Member**

Member	Description
SET_TOPIC	Type of topic message.
SET_DATA	Type of the published message.

3.2.2.9.2. QTT_CFG_MSG_ST

The structure of the MQTT message is defined below:

```
typedef struct
{
```

```
    MQTT_CFG_MSG_TYPE cfg_type;
    const char *topic;
    int topic_len;
    const char *data;
    int data_len;
    int qos;
    int retain;
} MQTT_CFG_MSG_ST
```

- **Parameter**

Type	Parameter	Description
MQTT_CFG_MSG_TYPE	*cfg_type*	Type of MQTT message. See ***Chapter 3.2.2.9.2*** for details.
const char	*topic*	Name of the MQTT topic.
int	*topic_len*	Length of the topic name.
const char	*data*	Content of the topic message.
int	*data_len*	Length of the topic message content.
int	*qos*	Service quality of the MQTT message.
int	*retain*	Reserved.

3.3. MQTT Operating Instruction

The MQTT API sample code is in the *ql_application/ql_mqtt_demo.c* directory.

3.3.1. Compile Sample Application

ql_mqtt_demo.c is compiled by default. *ql_mqtt_demo.c* file is already added to *ql_application/Make.defs*. The example is shown below:

```
CSRCS += ql_sl_tls_demo.c
CSRCS += ql_osi_demo.c
CSRCS += ql_mqtt_demo.c
```

Figure 3: Compile Sample Application

3.3.2. Function Testing

After compiling and burning the version, add the instance to the following CLI command line to test the MQTT connection function. At the same time, users need to ensure that the network data package is reachable with respect to the MQTT server's network.

Table 1: CLI Command

Command	Description
ql_mqtt_start uri mqtt://192.168.128.27 port 1883 client_id 4545	Set URI port client ID to create MQTT connection.
ql_mqtt_sub topic "123" qos 1	Publish a subject named "123" with communication quality 1.
ql_mqtt_unsub topic "123"	Unpublish the subject named "123".
ql_mqtt_pub topic "123" data “message” qos 1 retain 1	Sent "message" to the subject named "123".
ql_mqtt_stop	Disconnect MQTT.

4 TLS/SSL API

4.1. Header File

sl_tls.h, the header file of TLS/SSL API, is in the *ql_components/third_party/mbedtls* directory. Unless otherwise specified, all header files mentioned in this document are all located in this directory.

4.2. API Description

4.2.1. ssl_create

This function creates a TLS/SSL connection. when the network is available, users need to enter a valid URL or IP address string and the target port to create a TLS/SSL handle.

- **Prototype**

```
MbedTLSSession * ssl_create(char *url,char *port)
```

- **Parameter**

url:
[In] Valid URL or IP address string.

port:
[In] Target port string.

- **Return Value**

TLS/SSL handle. The handle contains information about the connection.

4.2.2. ssl_txdat_sender

This function sends data.

- **Prototype**

```
int ssl_txdat_sender(MbedTLSSession *tls_session,int len,char *data)
```

- **Parameter**

tls_session:
[In] TLS/SSL handle.

len:
[In] Length of the data to be sent. Unit: byte.

data:
[In] Data to be sent.

- **Return Value**

Greater than or equal to *0*	The number of bytes of data sent
Less than *0*	Failed execution

4.2.3. ssl_read_data

This function reads the received data.

- **Prototype**

```
int ssl_read_data(MbedTLSSession *session,unsigned char *msg,unsigned int mlen,unsigned int timeo
ut_ms)
```

- **Parameter**

session:
[In] TLS/SSL handle.

msg:
[Out] The buffer for storing the received data.

mlen:
[In] The size of the buffer for storing received data. Unit: byte.

timeout_ms:
[In] The time spent waiting (blocked) to receive data. Unit: millisecond.

- **Return Value**

Greater than or equal to *0*	The number of bytes of data received
Less than *0*	Failed execution

4.2.4. ssl_close

This function closes the TLS/SSL connection.

- **Prototype**

```
int ssl_close(MbedTLSSession *session)
```

- **Parameter**

session:
[In] TLS/SSL handle.

- **Return Value**

None

4.3. Operating Instructions

The code example file of TLS/SSL API is in *ql_application/ql_sl_tls_demo.c*. TLS/SSL is disabled by default. You need to call *ql_sl_tls_demo.c* after wlan_cli is enabled to add the specific example to CLI command line to test the TLS/SSL connection creation feature. At this point, the network should be available.

You can execute the following CLI commands to test the related APIA:

Table 2: CLI Command

Command	Description
tls create 192.168.19.101 443	Create a TLS/SSL connection. Connect to the server on port 443 of 192.168.19.101. If the connection is created successfully, a thread is started to receive the data of this connection.
tls sender test_send_data_str	Send data to the server. *test_send_data_str* is the data sent to the server. The data should not be separated by spaces; otherwise, it will

	be parsed into parameters by CLI commands.
tls create	Create a TLS/SSL connection. Connect to the server on port 443 of *www.baidu.com*. If the connection is created successfully, a thread is started to receive the data of this connection.
tls sender	Send data to the server. As shown in the code example below, the data is a request to obtain the web page through GET.

- **Code Example**

Add the TLS/SSL test command to CLI by adding *ql_application/ql_sl_tls_demo.c.* to *ql_application/Make.defs*.

```
CSRCS += ql_ota_demo.c
CSRCS += ql_adc_demo.c
CSRCS += ql_flash_demo.c
CSRCS += ql_sl_tls_demo.c
CSRCS += ql_osi_demo.c
CSRCS += ql_mqtt_demo.c
CSRCS += ql_gpio_demo.c
CSRCS += ql_ap_ble_network_demo.c
CSRCS += ql_spi_demo.c
CSRCS += ql_app.c
```

Figure 4: Add TLS/SSL Example

4.3.1. TLS/SSL Authentication

The following steps are required to establish a secure communication connection by mbedTLS:

- Initialize TLS/SSL context.
- Establish TLS/SSL handshake.
- Send or receive data.
- Close the connection after the interaction is completed.

Among these steps, the most critical step is the establishment of the TLS/SSL handshake. If authentication is required, certificate verification is needed; If no authentication is required, no certificate verification is needed.

The TLS/SSL authentication methods include non-authentication, one-way authentication, and two-way a uthentication.

- One-way authentication: The client or server verifies the legality of the peer's certificate.
- Two-way authentication: The client and the server verify the certificate legality of each other.

For this feature, you can use the following configuration to open the corresponding macros in *ql_components/third_party/mbedtls/ui/tls_client.h*.

```
#define CFG_USE_CA_CERTIFICATE 0
#define CFG_USE_CA_CERTIFICATE_VERIFY 0
```

When *CFG_USE_CA_CERTIFICATE* is set to 1, TLS/SSL loads and parses the CA certificate, and sets the data needed to verify the peer certificate. TLS/SSL can parse one or more certificates in buffer and add them to the root certificate list. If some certificates can be parsed, the number of failed certificates is returned. If no certificates can be parsed, the first error in parsing is returned. CA certificate is stored in *ql_components/third_party/mbedtls/ui/tls_certificate.h* dorectory.

When *CFG_USE_CA_CERTIFICATE_VERIFY* is set to 1, TLS/SSL is set to *MBEDTLS_SSL_VERIFY_REQUIRED* certificate verification mode, if the certificate verification fails, the communication is not continued. Otherwise *MBEDTLS_SSL_VERIFY_NONE* certificate verification mode will be used, the certificate is not verified.

4.3.2. TLS/SSL Debugging Information

TLS/SSL debugging information can be enabled by CFG_OUT_PUT_MBEDTLS_DEBUG_INFO. If *CFG_OUT_PUT_MBEDTLS_DEBUG_INFO* is set to 1, the log will be output through *ql_debug()*.Users can redefine *my_debug() in ./quectel_api/src/mbedtls/mbedtls-port/src/tls_client.c* to get the logs they are interested in. The following is the description of *my_debug():*

```
static void my_debug( void *ctx, int level,
                      const char *file, int line,
                      const char *str )
{
    ((void) level);

   // mbedtls_fprintf( (FILE *) ctx, "%s:%04d: %s", file, line, str );
    ///fflush(  (FILE *) ctx  );

    os_printf(LM_OS, LL_INFO,"[mbed]%s:%04d: %s\r\n", file, line, str);
}
```

5 WebSocket Function

5.1. Environment Setup

5.1.1. Hardware Preparation

- Prepare a Quectel FCM360W module.
- Prepare a laptop.

5.1.2. Software Preparation

5.1.2.1. Firmware Downloading

1. Get corresponding firmware package, development tools, and documentation from Quectel Technical Support.

2. Use the _V1.0.21_V1.7.7 tool of ECR6600 development toolbox to download the firmware. Get corresponding information from Quectel Technical Support.

3. After the downloading is completed, reboot the module and get the log information through serial port 0 with a serial port tool. The log information is below:

```
[00:00:00.124]adv activity success
[00:00:00.136]ps_set_mode: cfm_cnt 0
[00:00:00.139]E fhost_set_wpa_map L:68
[00:00:00.149]set key failed, status 0x2
[00:00:00.151]set key failed, status 0x2
[00:00:00.154]set key failed, status 0x2
[00:00:00.157]set key failed, status 0x2
[00:00:00.162]set key failed, status 0x2
[00:00:00.165]set key failed, status 0x2
[00:00:00.168]wpa state: wl0 DISCON->INACT.
[00:00:00.173]wpa state: wl0 INACT->DISCON.
[00:00:00.182]set key failed, status 0x2
[00:00:00.185]set key failed, status 0x2
[00:00:00.188]set key failed, status 0x2
[00:00:00.191]set key failed, status 0x2
[00:00:00.193]set key failed, status 0x2
[00:00:00.199]set key failed, status 0x2
[00:00:00.202]wpa state: wl1 DISCON->INACT.
[00:00:00.204]wpa state: wl1 INACT->DISCON.
```

Figure 5: Log Obtained with a Serial Port Tool

5.2. WebSocket Feature Test

This chapter introduces the WebSocket feature test (data loopback test) process.

5.2.1. Precondition

The hardware and software environment have been prepared.

5.2.1.1. Test Procedure

5.2.1.1.1. Connect to Wi-Fi

1. Execute the following command on the serial port tool to connect the module to the Wi-Fi hotspot named "quectel_wifi_test".

```
ql_wifi_sta wifi_test 012345678
```

The parameters in the above command are defined below:
ql_wifi_sta: command category
wifi_test: SSID
012345678: Password

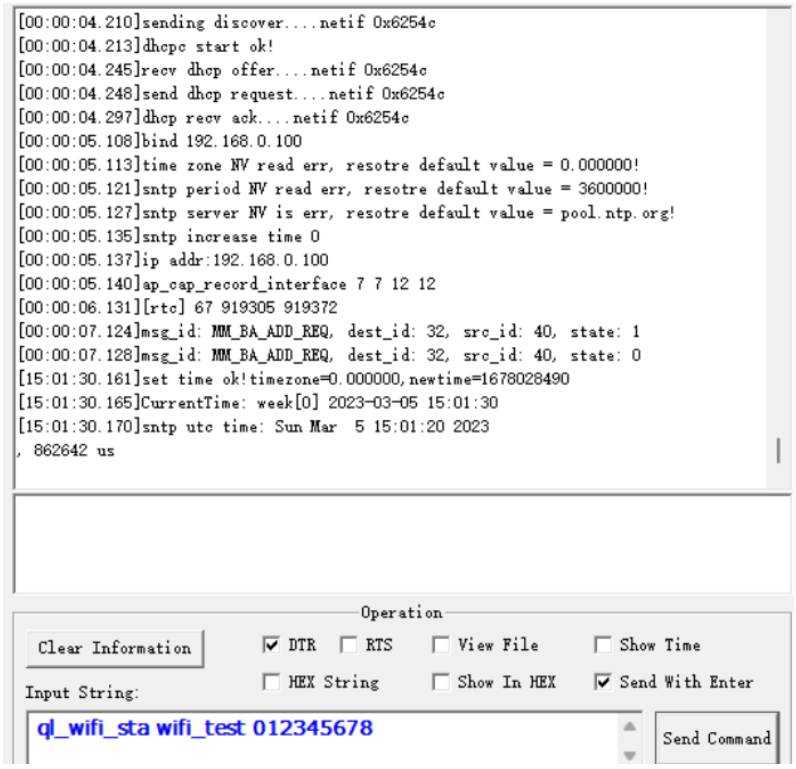

Figure 6: Connect the Module to Wi-Fi Hotspot

2. Connect the laptop to the same Wi-Fi hotpot.

5.2.1.1.2. Open WebSocket Client

Input "**ql_httpserver Start**" on the module:

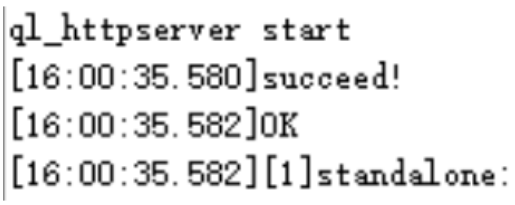

Figure 7: Open WebSocket Client

5.2.1.1.3. Connect to WebSocket Server

1. Execute **ifconfig** on the serial port tool to view module IP address.

```
[16:12:01.047]  dns2:[202:102:213:68]
[16:12:01.053]     name:[wl]
[16:12:01.053]      num:[1]
[16:12:01.055]    flags:[0x6e]
[16:12:01.058]       hw:[00:06:06:00:00:01]
[16:12:01.064]      mtu:[1500]
[16:12:01.066]  ip_addr:[0.0.0.0]
[16:12:01.069]  netmask:[0.0.0.0]
[16:12:01.072]       gw:[0.0.0.0]
[16:12:01.075]————————————————————————
[16:12:01.077]     name:[wl]
[16:12:01.080]      num:[0]
[16:12:01.083]    flags:[0x6f]
[16:12:01.086]       hw:[00:06:06:00:00:00]
[16:12:01.091]      mtu:[1500]
[16:12:01.094]  ip_addr:[192.168.0.100]
[16:12:01.096]  netmask:[255.255.255.0]
[16:12:01.099]       gw:[192.168.0.1]
[16:12:01.102]————————————————————————
[16:12:01.107]OK
```

Figure 8: View Module IP Address

2. Open a browser and input the module IP address, such as *192.168.0.100/setting*.

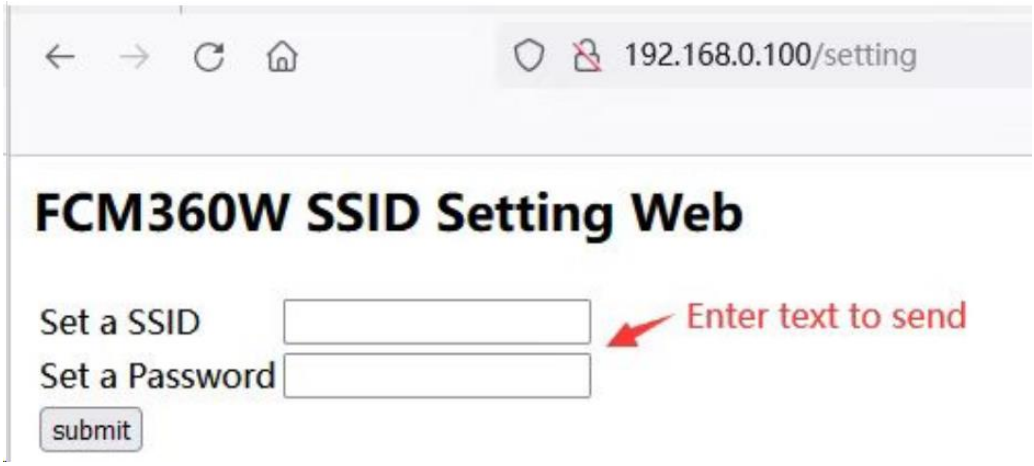

Figure 9: Open the Web in the Browser

5.2.1.1.4. Send Message

1. Input the data to be sent in "Set a SSID", " Set a Password", and click "**submit**" to send the data out.

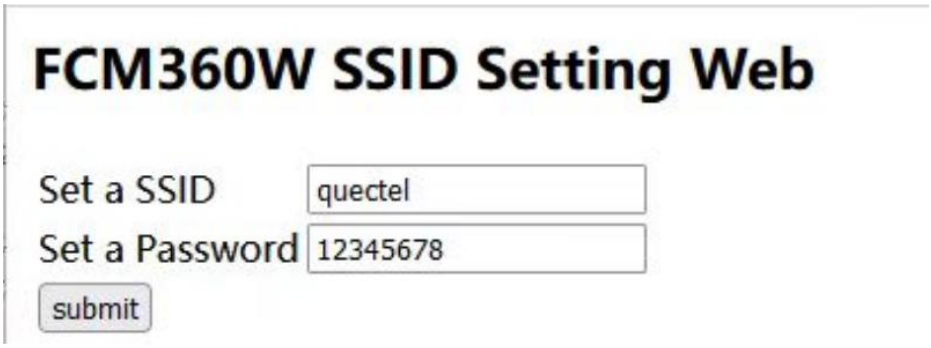

Figure 10: Send Message

2. If the data is sent out successfully, WebSocket receives the message from the client.

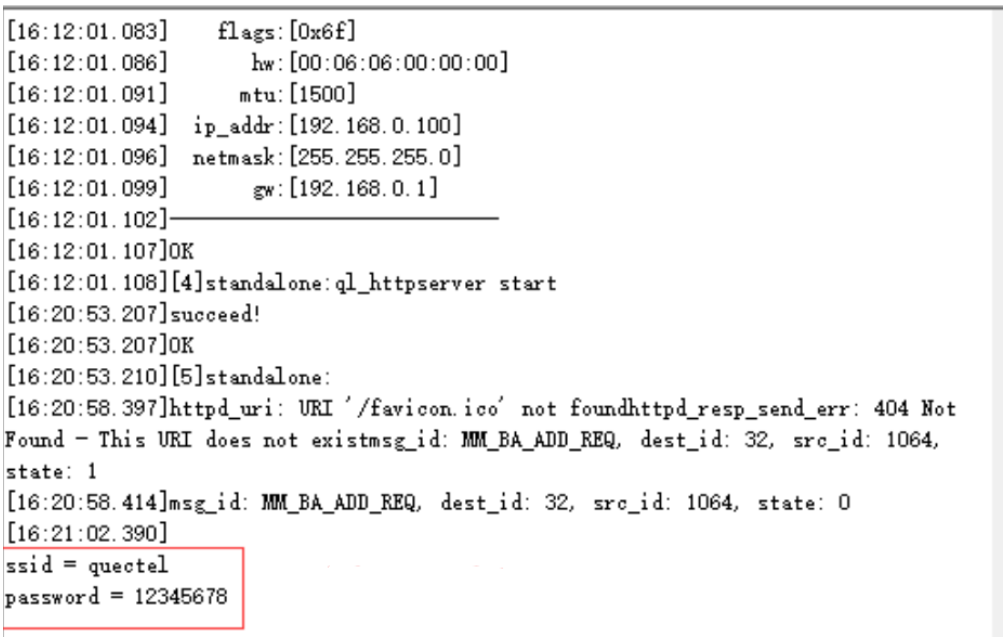

Figure 11: Messages Received by the Module

6 Appendix References

Table 3: Related Document

Document Name
[1] Quectel_FCM360W_QuecOpen_ Quick_Start_Guide

Table 4: Terms and Abbreviations

Abbreviation	Description
API	Application Programming Interface
ID	Identifier
IoT	Internet of Things
IP	Internet Protocol
MQTT	Message Queuing Telemetry Transport
QoS	Quality of Service
RTOS	Real-Time Operating System
SDK	Software Development Kit
SSID	Service Set Identifier
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator