Quectel BG95
TCPUDP Application Note

Confidentiality Level: (Tick the Box ■)		
Top Secret □	Confidential □	Public □

Document Control Records

Revision History			
Date	Revision	Revision Description	Author
2023-08-20	0	Initial	Floyd.Wang
2023-11-19	1	Modified by Copywriter Team	Floyd.Wang Kelly.Chen

Contents

1 Purpose

TCP and UDP are the most common transport layer protocols, which are relied by multiple upper-layer applications l for data transmission, such as FTP, RTP, and so on. In the development of IoT projects, it is also often necessary to use TCP/UDP for direct data transmission with servers.

The BG95 cellular network module provides a built-in TCP/IP protocol stack. This document aims to guide MCU developers in quickly completing network function development using BG95's built-in TCP/IP protocol stack.

2 Scope

This document applies to products with MCU mounted with BG95 module.

3 Terms and Definitions

Quectel: Quectel Wireless Solutions Co., Ltd.
TCP: Transmission Control Protocol
UDP: User Datagram Protocol
DNS: Domain Name System

4 API Design

Quectel has designed a set of reference APIs using BG95's AT commands to implement data transmission and reception functions for TCP/UDP. See specific illustrations in Table 1.

Table 1: TCP/IP API Reference Design

API	Functionality
QL_data_call_start()	Control the module to initiate data call and retrieve IP, DNS, etc.
QL_data_call_stop()	Control the module to disconnect the data call.
QL_socket_create()	Create socket connections in either server mode or client mode.
QL_socket_listen()	In server mode, listen for incoming connection requests from clients.
QL_socket_close()	Close socket connections.
QL_socket_cfg()	Configure parameters such as keep-alive time, retransmission count, etc.
QL_send_data()	Send data.
QL_recv_data()	Receive data.

QL_dns_query()	Query IP address through domain name

Please refer to the appendix document for the detailed design of these APIs:
Quectel_BG95_TCP(IP)_API_design.docx

See AT command reference table corresponding to API as below.

Table 2: AT Command Reference Table

API	AT Command
QL_data_call_start()	AT+QIACT
QL_data_call_stop()	AT+QIDEACT
QL_socket_create()	AT+QIOPEN
QL_socket_listen()	+QIURC: "incoming"
QL_socket_close()	AT+CEREG?
QL_socket_cfg()	AT+QICFG
QL_send_data()	AT+QISEND
QL_recv_data()	AT+QIRD

In the following **Chapter 5** and **Chapter 6**, we will explain how to accomplish data transmission and reception for TCP and UDP protocols based on these APIs.

5 TCP Network Programming

5.1. TCP Communication Process

The BG95 module is equipped with a built-in TCP/IP protocol stack, which can automatically handle socket creation, IP address and port binding, data transmission, and other operations. The MCU only needs to handle data transmission, data processing, and uses the API designed in **Chapter 4**. The basic flow of process is shown in **Figure 1**.

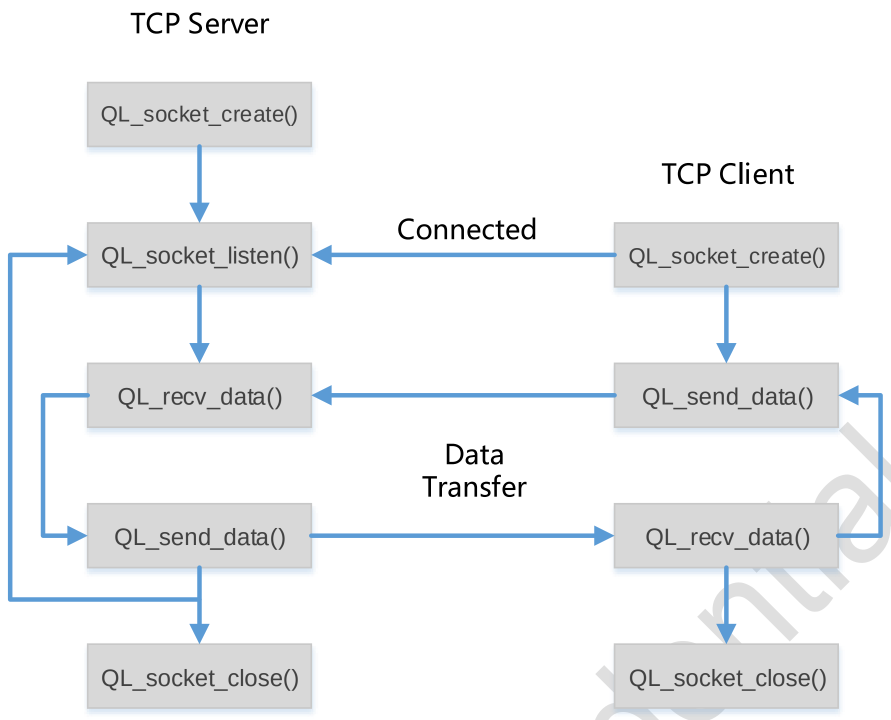

Figure 1: TCP Communication Process

5.1.1. TCP Client Mode Application

a) Call **QL_socket_create()** to create a socket, specify the remote server IP address, port number, APN and connection ID, and configure it as a client mode. The module will attempt to establish a connection with the remote server based on the incoming parameters and automatically complete the three-way handshake.
b) After the socket channel is established successfully, call **QL_send_data()** to transfer the data to be sent. The module will automatically packetize the data for transmission. The TCP transmission confirmation, retransmission, and other actions are automatically handled by the module in stage of transmission.
c) When the remote server returns data, it will be called back through **QL_recv_data()** to provide the received data. The connection ID from Step a) is used to identify among different servers. Users can define their own methods to handle the returned data from the server in **QL_recv_data()**.
d) When the data transmission is accomplished, you can call **QL_socket_close()** to close the socket channel, or the module can handle TCP keep-alive based on user configuration, waiting for the next data transmission. Similar to socket creation, the four-way handshake actions during closure are automatically handled by the module.

5.1.2. TCP Server Mode Application

a) Call **QL_socket_create()** to create a socket, specify the local IP address, port number, APN, and configure it as a server mode. The module will enter listening mode based on the incoming parameters, waiting for client connections.
b) When a client connects, it will call **QL_socket_listen()** in a callback manner, providing the client's connection ID for the user to record.
c) After the client has successfully connected, if any data is received from the client, it will call **QL_recv_data()** in a callback manner for the user to handle the data in **QL_recv_data()**.
d) If there is data to be sent, call **QL_send_data()** to transmit the data, specifying the connection ID obtained in **Step b)** to identify between different clients.
e) When data transmission is complete, you can call **QL_socket_close()** to close the socket channel.

5.2. TCP Application Example

5.2.1. TCP Application Flow Diagram

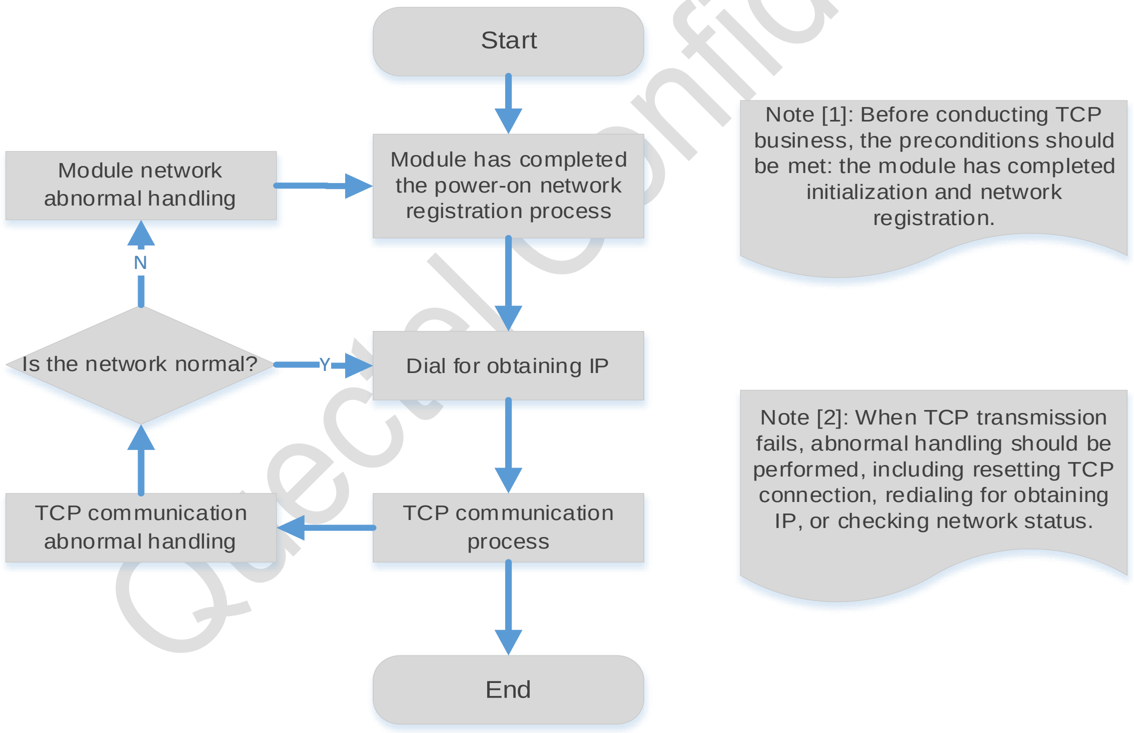
Start
Module has completed the power-on network registration process
Module network abnormal handling
Note [1]: Before conducting TCP business, the preconditions should be met: the module has completed initialization and network registration.
N
Is the network normal?
Y
Dial for obtaining IP
Note [2]: When TCP transmission fails, abnormal handling should be performed, including resetting TCP connection, redialing for obtaining IP, or checking network status.
TCP communication abnormal handling
TCP communication process
End

Figure 2: TCP Application Flow

Note [1]: Please refer to the module initialization and network registration process.
Note [2]: Please refer to Chapter 7 - TCP/UDP Exception Handling

6 UDP Network Programming

6.1. UDP Communication Process

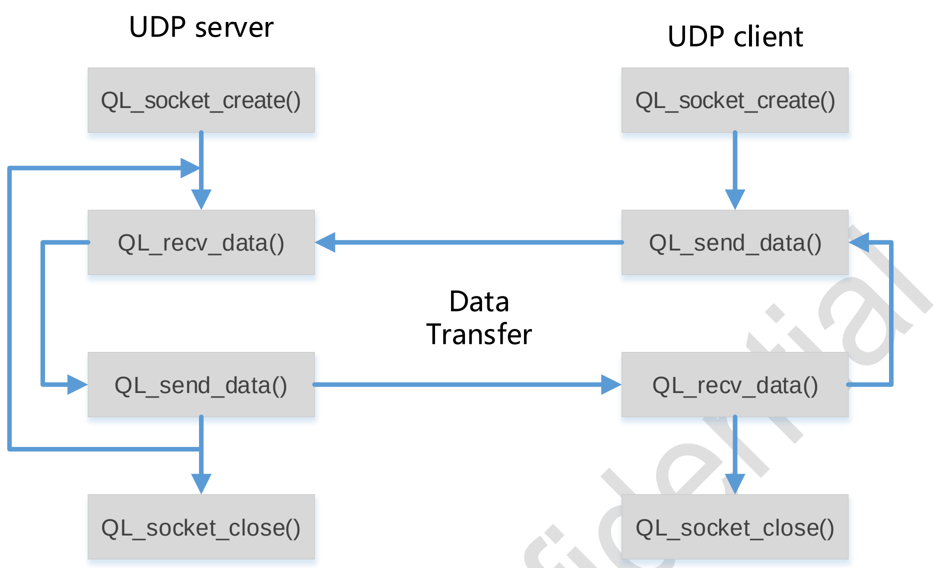

Figure 3: UDP Communication Process

6.1.1. UDP Client Mode Application

a) Call **QL_socket_create()** to create a socket, specify the connection ID and configure it as the client mode.
b) After the socket channel is established successfully, call **QL_send_data()** to transmit the data to be sent. During the transmission, specify the server's IP address, port number, and the connection ID from **Step a)**. The module will automatically packetize the data for transmission.
c) When the remote server returns data, it will be received through **QL_recv_data()**. In addition, the received data will be identified via the connection ID from **Step a)**. Thus, the users can define their own processing methods for the returned server data in **QL_recv_data()**.
d) When the data transmission is completed, call **QL_socket_close()** to close the socket channel.

6.1.2. UDP Server Mode Application

a) Call **QL_socket_create()** to create a socket, specify the connection ID, and configure it in server mode.
b) After the socket channel is established successfully, call **QL_send_data()** to transfer the data to be sent. When sending, specify the client IP address, port number, and the connection ID from **Step a)**. The module will automatically packetize the data for transmission.
c) When the client returns data, it will callback **QL_recv_data()** to provide feedback on the received

data. The connection ID from **Step a)** is used to identify different servers. Users can define their own methods to process the returned data in **QL_recv_data()**.

d) Call **QL_socket_close()** to close the socket channel when the data transmission is finished.

6.2. UDP Application Example

6.2.1. UDP Application Flow Diagram

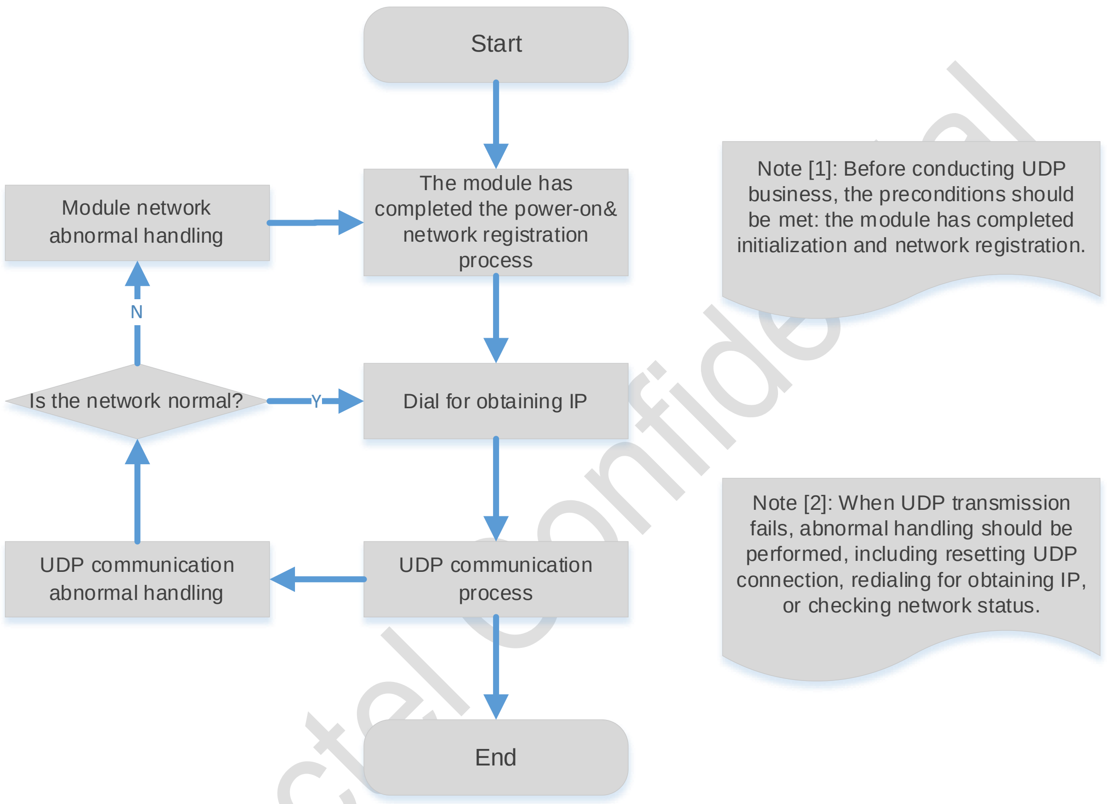
Start
The module has completed the power-on& network registration process
Module network abnormal handling
N
Is the network normal?
Y
Dial for obtaining IP
UDP communication process
UDP communication abnormal handling
End
Note [1]: Before conducting UDP business, the preconditions should be met: the module has completed initialization and network registration.
Note [2]: When UDP transmission fails, abnormal handling should be performed, including resetting UDP connection, redialing for obtaining IP, or checking network status.

Figure 4: UDP Application Flow

Note [1]: Please refer to the <module initialization and network registration process.>

Note [2]: Please refer to **Chapter 7** - TCP/UDP Exception Handling

7 TCP/UDP Exception Handling

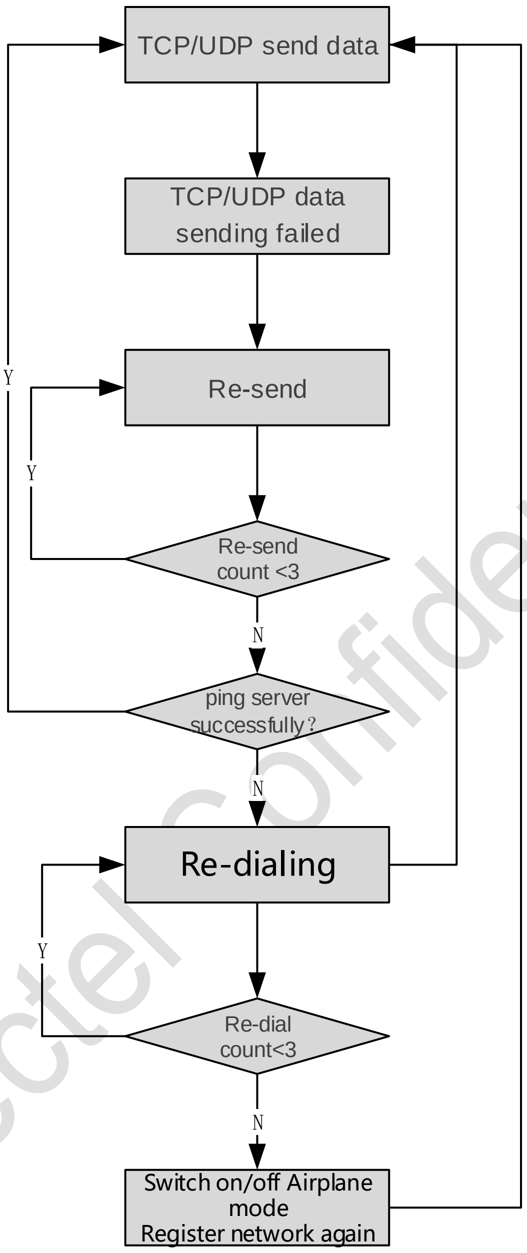
TCP/UDP send data
TCP/UDP data sending failed
Re-send
Re-send count <3
Y
N
ping server successfully？
Y
N
Re-dialing
Re-dial count<3
Y
N
Switch on/off Airplane mode Register network again

Figure 5: TCP/UDP Exception Handling

a) When data transmission fails, it is recommended to call **QL_send_data()** for multiple attempts. However, the maximum retry times shall not surpass 3.
b) After retrying sending data for 3 times, it is necessary to ensure that the module is connected to the network via ping command once it fails. If there is a problem with the ping detection, it is advised to re-dial since it can help avoid issues related to abnormal IP addresses caused by network behavior. It is recommended that the time of re-dialling shall not surpass 3.
c) If the problem still displays even after re-dialling, it is recommended to re-establish network

connection by switching on/off airplane mode or rebooting the module. It is suggested to try at intervals of 5 minutes, 10 minutes, 30 minutes, 2 hours, and 5 hours. The intervals for re-establishing network connection should be appropriately longer because turning on/off airplane mode or rebooting the module involves writing a large number of files to the Flash of module and SIM card, which will also significantly reduce the device's lifespan.

8 Appendix A Reference

Quectel_BG95&BG77&BG600L_Series_TCP(IP)_Application_Note_V1.2.pdf
Quectel_BG95_TCP(IP)_API_design.docx