QSTM32

TCP/UDP 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
2025-08-29	V2.0	The second generation initial version	Wells Li Linkin Wang

Quectel Confidential

Contents

Quectel Confidential

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 cellular communication module features a built-in TCP/IP protocol stack. This document is designed to assist MCU developers in efficiently implementing network functionality by utilizing the module's integrated TCP/IP capabilities.

2 Scope

This document applies to products with MCU mounted with cellular 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 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
socket()	Create a new socket.
close()	Close the socket connection and releases the associated resources.
connect()	Establish a connection to a remote socket.
bind()	Assign a local protocol address (IP and port) to a socket.
sendto()	Send data to a specified address (UDP).
send()	Send data over a connected socket (TCP).
recvfrom()	Receive data and captures the address of the sender (UDP).
recv()	Receive data from a connected socket(TCP).

Quectel Confidential

setsockopt()	Set options and parameters for the socket(only the receive timeout option is supported)
listen()	Place the socket in server mode and prepares it to listen for incoming connection requests.
accept()	Accept an incoming connection request on a listening socket, creating a new socket for the connection.
select()	Monitor multiple sockets for activity.

Please refer to the appendix document for the detailed design of these APIs:
Quectel_QSTM32_SDK_API_Design_V2.0

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 cellular 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**.

Quectel Confidential

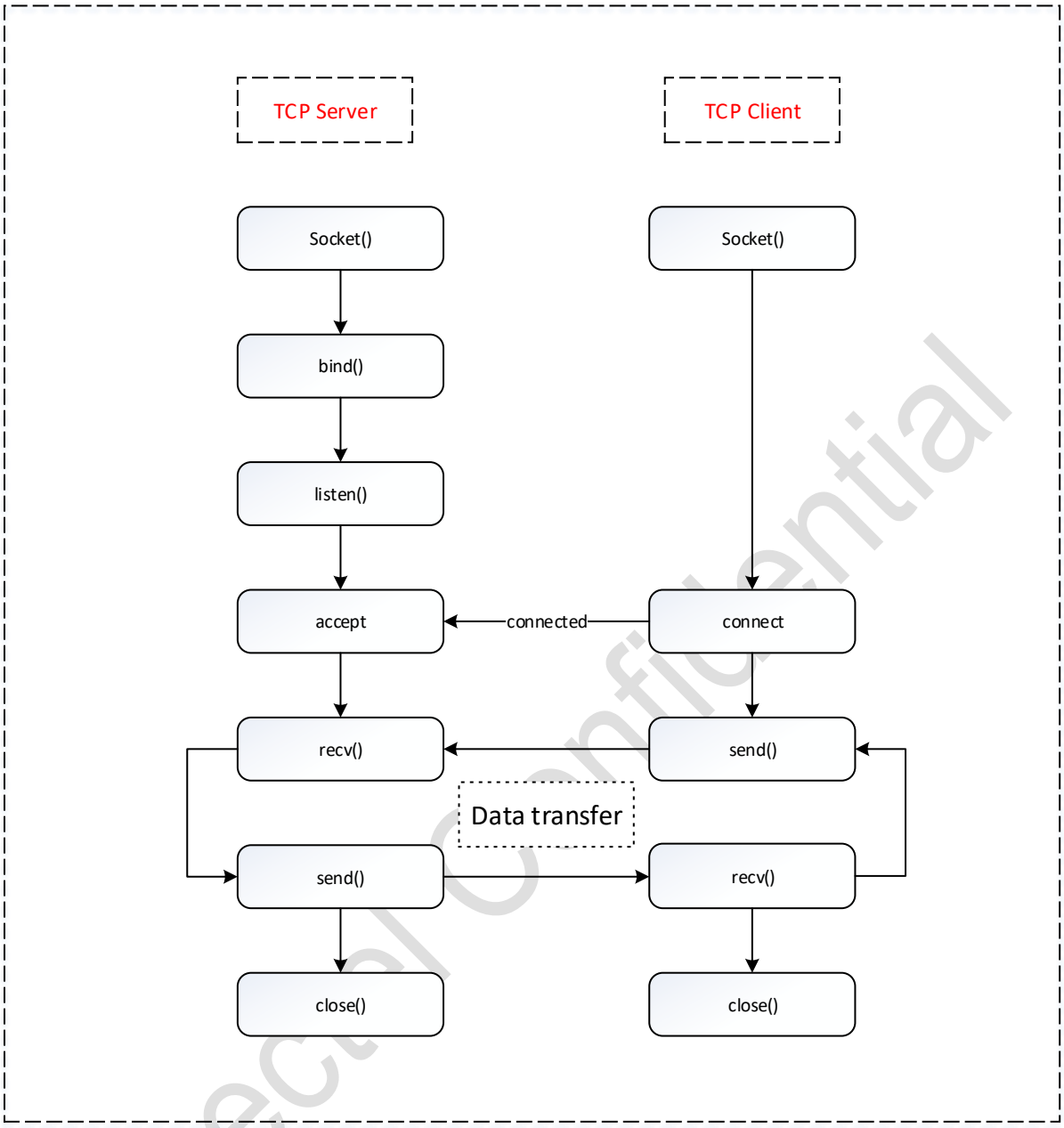

Figure 1: TCP Communication Process

Quectel Confidential

5.1.1. TCP Client Mode Application

a) Call **socket()** to create a socket, then call **connect()** to initiate a connection to the TCP server. The module will automatically establish the link and complete the three-way handshake using the provided parameters (server IP address and port).
b) After the socket channel is established successfully, call **send()** 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 data transmission is accomplished, you can call **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) First, call **socket()** to create a listening socket. Then, call **bind()** to associate the socket with a specific local IP address and port. Next, call **listen()** to mark the socket as passive, enabling it to accept incoming connection requests. Finally, call **accept()** to block and wait for an incoming client connection. This call returns a new connected socket for communication with that client.
b) When a client connects, accept() will return a new connected socket for communication with that client.
c) After a connection is established, the server can call **recv()** to read any data received from the client.
d) To send data to a specific client, call **send()** on that client's socket descriptor (obtained in *Step b*). The TCP stack automatically handles packetization, acknowledgment, and retransmission.
e) When data transmission is complete, you can call **close()** to close the socket channel.

6 UDP Network Programming

6.1. UDP Communication Process

Quectel Confidential

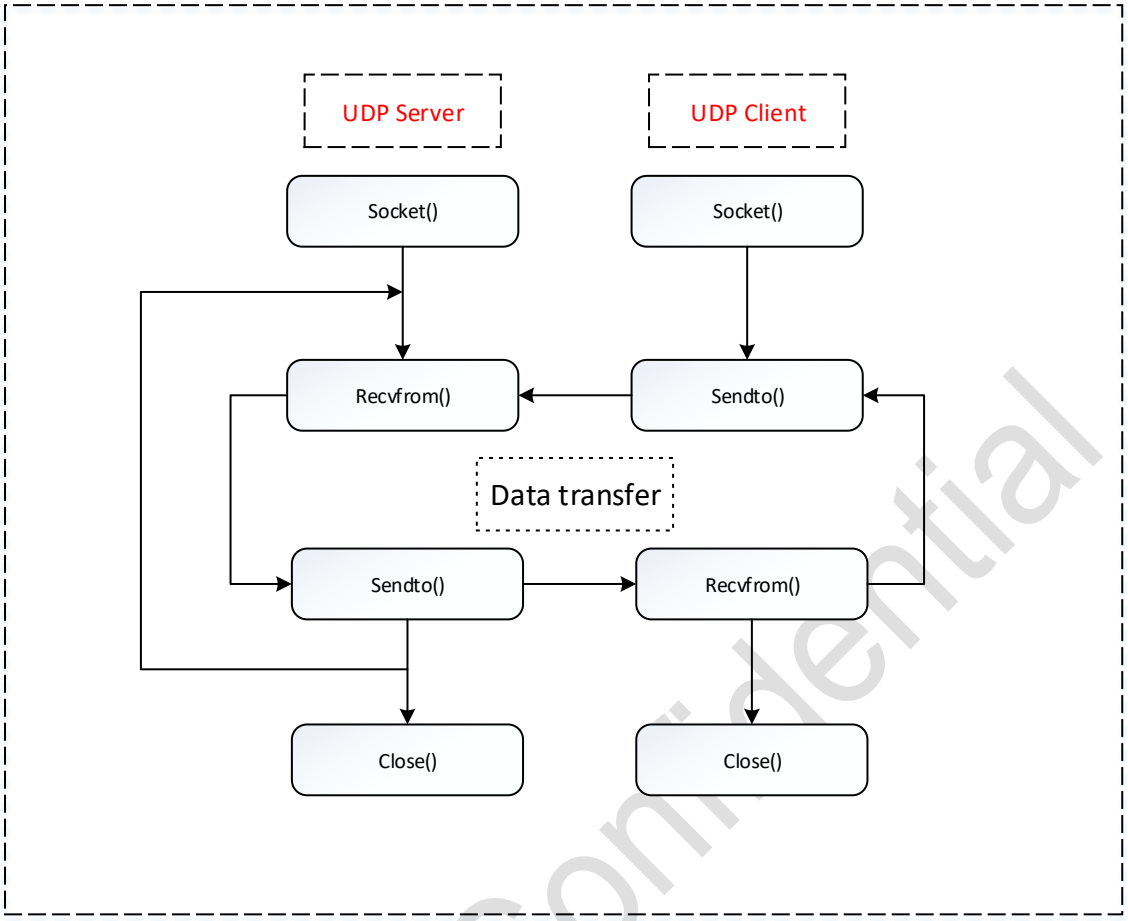

Figure 2: UDP Communication Process

6.1.1. UDP Client Mode Application

a) Call **socket()** to create a datagram socket (e.g., **SOCK_DGRAM**).
b) Call **sendto()** to send data to the server, specifying the server's IP address and port number along with the data buffer. Since UDP is connectionless, each **sendto()** call independently specifies the target address.
c) Call **recvfrom()** to receive data from the peer server. This function will return the data payload sent by the server.
d) When the data transmission is completed, call **close()** to close the socket.

6.1.2. UDP Server Mode Application

a) Call **socket()** to create a datagram socket (e.g., **SOCK_DGRAM**). Then, call **bind()** to associate the socket with a specific local IP address and port.
b) Call **recvfrom()** to receive data from any client, it returns the received data buffer and the source address (client's IP address and port number).

Quectel Confidential

c) Call **sendto()** to send a response to a specific client, specify the data to be sent and the destination address (the client's IP and port obtained from the **recvfrom()** call in *Step b*).
d) Call **close()** to close the socket.

7 TCP/UDP Exception Handling

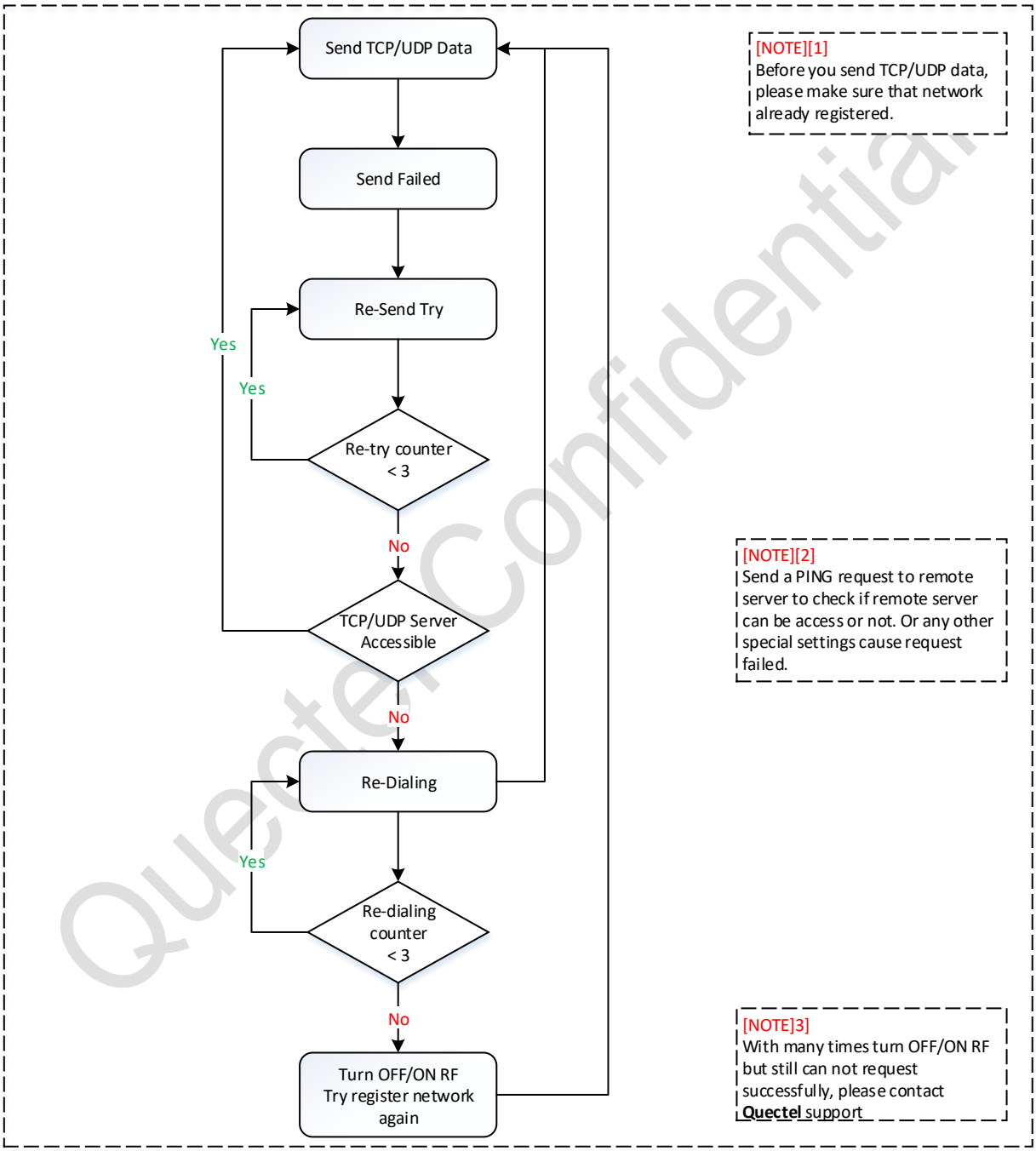

Figure 3: TCP/UDP Exception Handling

Quectel Confidential

a) When data transmission fails, it is recommended to call **send()** 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 behaviour. 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_QSTM32_SDK_API_Design_V2.0

Quectel Confidential