EEL4598/5718:
Computer Communication Project

Problem #3:
A TCP-based socket communication system

Jing Xu , Tongqing Yu

Program Timeline and responsibility of each team member:

Week 12 (11.16-11.20):
Collect and classify socket programing information and some java code using samples. (Tongqing Yu)

Week 13 (11.23-11.28):
Learning socket programing and build a simple communication system which can only host one client. (Jing Xu)

Week 14(11.30-12.4):
Increase the number of available clients by applying multithread socket communication system. (Jing Xu)
Collate and modify all code in one format. (Tongqing Yu)
Capture and analyze the transmitting packets using Wireshark. (Tongqing Yu)

Week 15(12.7-12.9):
Present demo to TA. (Jing Xu)
Writing a report of this project. (Jing Xu, Tongqing Yu)
Collate all code in one format. (Tongqing Yu)

A TCP-based Socket Communication system

Abstract: Network communication protocol laid the foundation of today's internet, which provided a guarantee for rapid economic development and convenient life. Among many communication protocols, TCP is widely applied in different cases because of its reliable transmission. In this paper, a TCP-based socket communication system was set and analyzed to further grasp the knowledge learned in lectures.

I. Working principle of transmission control protocol (TCP)

1. TCP/IP model[1]

In TCP/IP model, applications are in the application layers, which is above transport layer, internet layer and link layer. The transport layer provides session and datagram communication services to the application layer through two core protocols, that are transmission control protocol (TCP) and user datagram protocol (UDP).

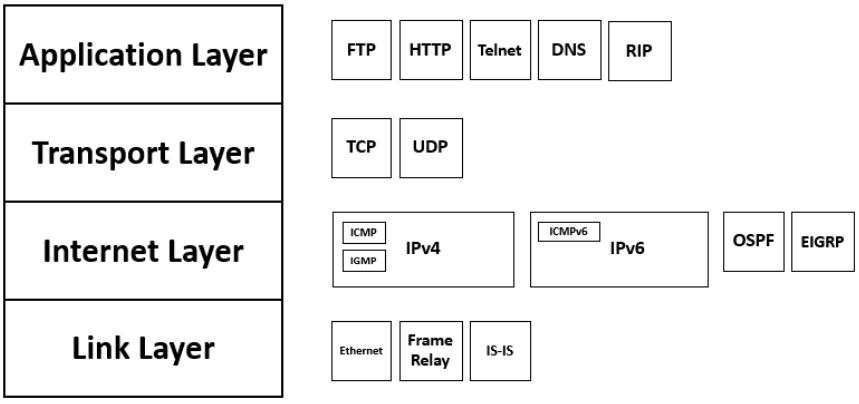


Fig 1 TCP/IP protocol

For TCP, there are five main characteristic:

- TCP provides a reliable connection-oriented byte stream service
- In a TCP connection, only two parties communicate with each other. Broadcast and multicast are not available for TCP
- TCP uses checksum, validation, and retransmission mechanisms to ensure reliable transmission
- TCP sorts data in segments and uses cumulative validation to ensure that the order of data is constant and non-repetitive
- TCP uses the sliding window mechanism to control the flow, and dynamically changes the window size to control congestion

Although TCP is reliable, it doesn't mean that TCP can promise the message can be received. What TCP can provide is that, if possible, it can deliver the data to the recipient, or notify the user by giving up retransmission and breaking the connection. Thus, TCP is not a 100% reliable protocol. What it can provide is reliable delivery of data or reliable notification of failure.
To ensure reliable transmission, TCP defines a three-time handshake model in data transmission. Three-way Handshake means that the client and server send a total of three packets when establishing a TCP connection.

2. Three-way handshakes[2]

The purpose of the three handshakes is to connect to the specified port of the server, establish a TCP connection, synchronize the serial number and confirmation number of both sides of the connection, and exchange TCP window size information. In socket programming, when the client executes connect() three handshakes will be triggered.

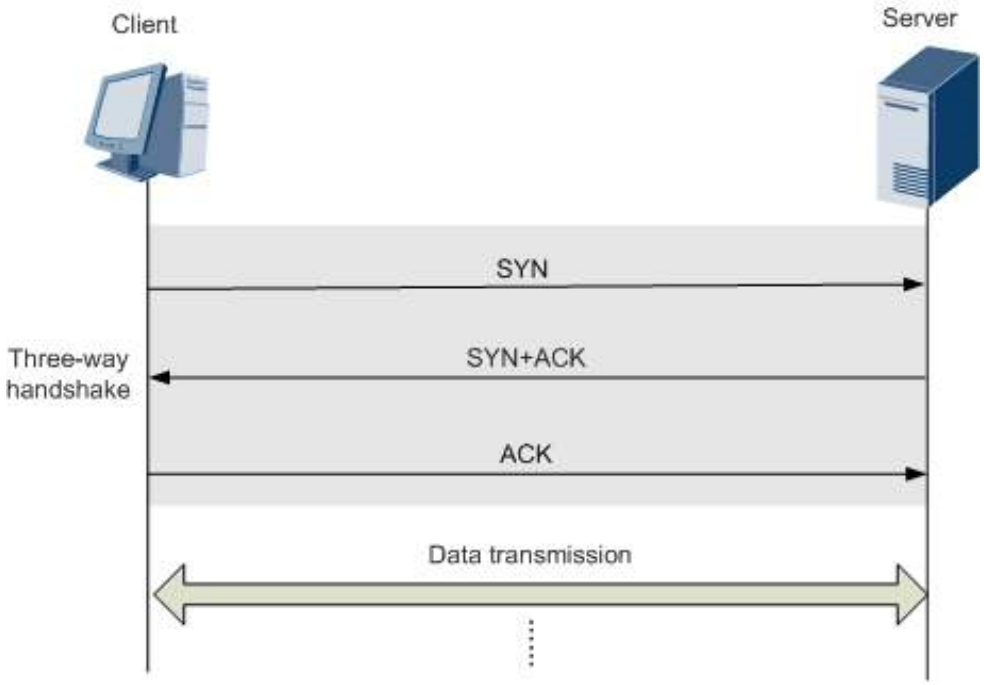


Fig 2 Three-way handshake

- First handshake (SYN=1, seq =x):
  The client sends a TCP packet with the SYN flag at position 1, indicating the port of the server to which the client intends to connect, and the initial Sequence Number X, saved in the header field Sequence Number.

  After sending, the client enters SYN_SEND state.

- Second handshake (SYN=1, ACK=1, seq=y, ACKnum=x+1):
  The server sends back a confirmation packet (ACK) reply. That is, both the SYN and ACK flags are 1. The server chooses its own "ISN serial Number" and puts it in the Seq field, and at the same time sets the Acknowledgement Number to use the customer's ISN plus 1, that is, X+1.

  After sending, the server enters the state of SYN_RCVD.

- Third handshake (ACK=1, ACKnum=y+1)
  The client sends the acknowledgment packet (ACK) again, with the SYN flag bit 0 and the ACK flag bit 1, and sends the serial number field +1 that the server sent ACK to the other party in the confirmation field, and puts the +1 that says "ISN" in the data segment.

  After sending, the client enters an ESTABLISHED state. When the server receives the packet, it also enters an ESTABLISHED state, and the TCP handshake ends.

3. Four-way handshakes[2]

TCP's disconnect requires sending four packets, so it's called a four-way handshake, also called an improved triple handshake. Either the client or the server can initiate a wave action actively. In socket programming, either party performs a close() operation to generate a wave action.

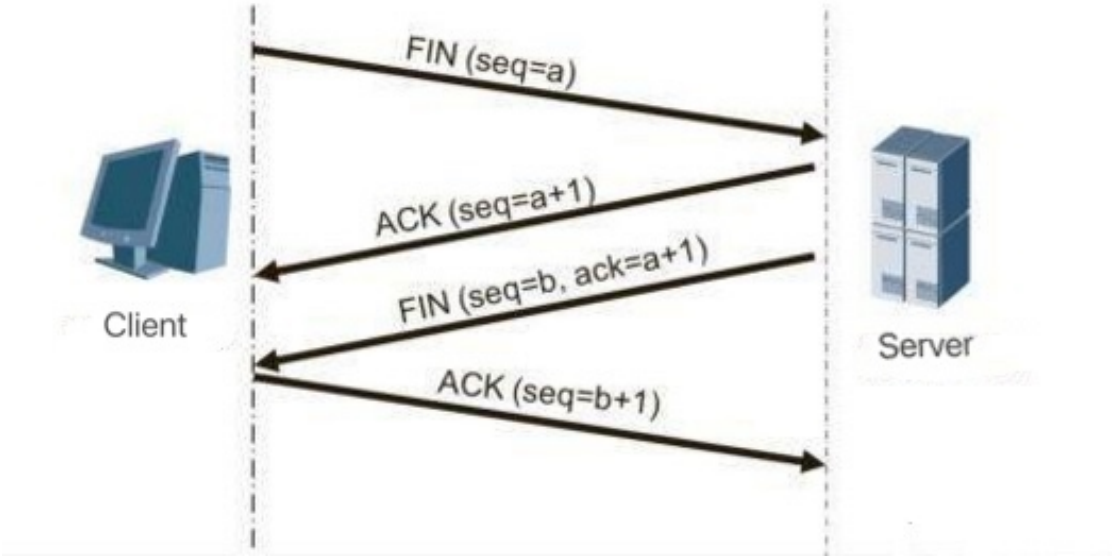


Fig 3 Four-way handshake

- First wave (FIN=1, seq =x)
  Suppose the client wants to close the connection, the client sends a packet with the FIN flag position 1 to indicate that it has no data to send, but still can accept the data.

  After sending, the client enters the FIN_WAIT_1 state.

- Second wave (ACK=1, ACKnum=x+1)
  The server confirms the FIN packet from the client and sends a confirmation packet indicating that it has accepted the client's request to close the connection but is not ready to do so.
  After sending, the server enters CLOSE_WAIT state. After the client receives the acknowledgement package, it enters FIN_WAIT_2 state and waits for the server to close the connection.

- Third wave (FIN=1, seq =y)
  When the server is ready to close the connection, send an end connection request to the client with FIN set to 1.

  After sending, the server enters the LAST_ACK state and waits for the last ACK from the client.

- Fourth wave (ACK=1, ACKnum=y+1)
  The client receives the shutdown request from the server, sends an acknowledgement packet, and enters the TIME_WAIT state, waiting for a possible ACK packet that requires retransmission.

  After the server receives the confirmation packet, it closes the connection and enters the CLOSED state.

  After waiting for a fixed time (two Maximum life cycles, 2MSL, 2 Maximum Segment Lifetime), the client did not receive an ACK from the server and thought that the server had CLOSED the connection normally, so it CLOSED the connection and entered the CLOSED state.

4. TCP Keepalive[3]

TCP connection, in fact, is a software-only concept; there is no such thing as a "connection" at the physical level. The two sides of TCP communication establish interactive connections, but the data interaction does not always exist. Some connections will actively release the connection after the data interaction, while others will not.

In long period of countless according to interact, interact off electricity, crash may occur on both sides, abnormal restart all sorts of accidents, such as when the accident happened, the normal TCP connection has not had time to release, in the software level, on the other side of the connection does not know that in the case, it will continue to maintain this connection, the accumulation of long time can cause a lot of half open the connection, cause the system resource consumption and waste, in order to solve this problem, at the transport layer can be done using TCP KeepAlive mechanism. Most major operating systems support this feature in the kernel.

The basic principle of TCP KeepAlive is that a probe packet is sent to the connection counterpart at regular intervals. If an ACK is received, the connection is considered to be alive. If no response is received after a certain number of retries, the TCP connection is discarded.

TCP-Keepalive-HOWTO has a detailed description of the FEATURES of TCP Keepalive. The limitations of TCP KeepAlive are highlighted here. First of all, TCP KeepAlive can be monitored by sending a probe packet, which will bring extra traffic to the network. In addition, TCP KeepAlive can only monitor the survival of the connection at the kernel level, and the survival of the connection does not necessarily represent the availability of the service. For example, when a server's CPU utilization reaches 100% and the server is already jammed and unable to respond to requests, TCP KeepAlive will still consider the connection alive. Therefore, TCP KeepAlive is of relatively small value to application layer applications. Application layer programs that need to be connected to maintain life, such as QQ, will often realize their own heartbeat function in the application layer.

II. Program implementation with Java

1. socketClient

Create a Java class socketClient and instantiate it with hostname and port number. When the client and server run on the same computer, hostname = “localhost”. When the client and server run on the different computer, we need to change the hostname in the program to the IP address of the computer the server is running on. The server listen on port number 8000 in our project.

```
public static void main(String[] args){

    //When communicate within one computer
    String hostname = "localhost" ;

    //When the server is running on anther computer, hostname is  server side computer's IP address
    //String hostname = "100.64.9.115";

    int port = 8000;

    socketClient client = new socketClient(hostname, port);
    client.execute();
}
```

Figure 4: screenshot of socketClient

Within the socketClient class, we use java net.Socket to create socket. For input and output, we use java.io.BufferedReader and java.io.PrintWriter to read and send messages. Try-catch is applied when client is trying to get message and send message. From the client, user could get notification of all the connected users and see messages of all the users.

2. socketServer

Create socketServer class and instantiate it. Here we let the server listen on port number 8000.

```
    //start the server
    public static void main(String[] args) {
        // initialize the port the server is listening to
        // Here we set the port as 8000
        int port = 8000;
        socketServer server = new socketServer(port);
        server.execute();
    }
```

Figure 5: screenshot of socketServer

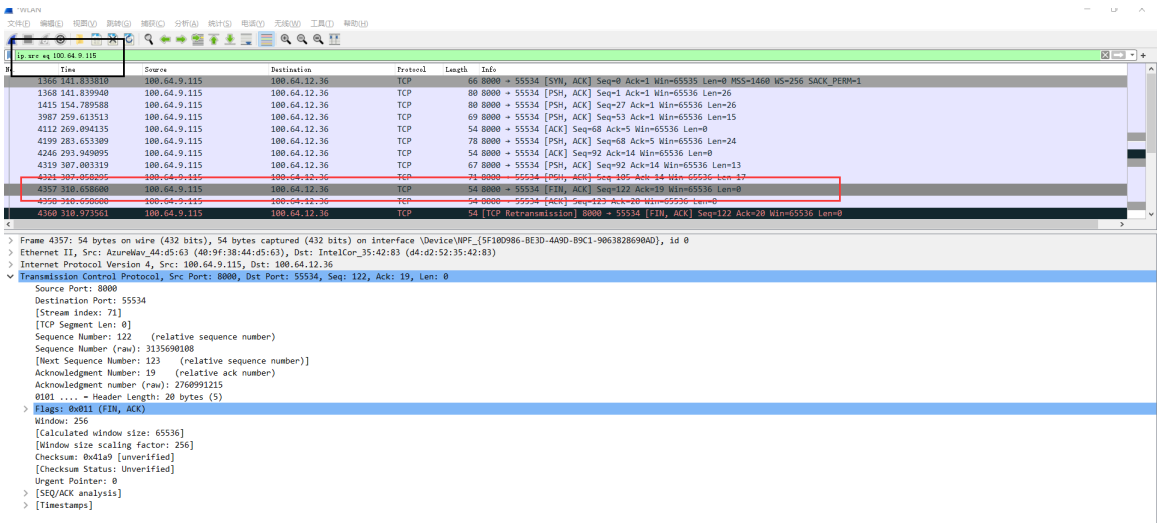
Within the socketServer, we create clientThread to server all the connected servers. The thread could help to achieve parallelism in Java program. In the clientThread, a client could read and send message and check if the user wants to leave. Meanwhile it could broadcast the information to all the other connected clients. Still, we use java.io.BufferedReader and java.io.PrintWriter to process read and write. We use Java sets to store connected users and clientThreads.

III. Analysis of transmitting packets using Wireshark.

1. Wireshark network protocol analyzer

Wireshark is a free and open source packet analyzer. It is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets.

2. Analysis of packets



I use a filter to find out all the packets whose source IP is 100.64.9.115 which is the IP address od my host computer. Then Wireshark captures all the packets that it sent from the host computer. We can easily find all the basic information of the packets.

Host IP address	100.64.9.115
Host side port number	8000
Server IP address	100.64.12.36
Server side port number	55534
Transmission protocol	TCP

Then we take the highlighted packet as a example to analysis one single packet. We can acquire detail information of one single packet.

Sequence number	122
Acknowledge number	19
Window size	256
Checksum	0x41a9
Checksum status	Unverified

Reference

[1] A. Leon-Garcia, I. Widjaja, Communications Networks: Fundamental Concepts and Key Architectures

[2] A. Tanenbaum, Computer Network

[3] W. Stalling, Data & Computer Communications

[4] Andrew Huang, TCP three-way handshakes and four-way handshakes explanation https://www.cnblogs.com/hnrainll/archive/2011/10/14/2212415.html

[5] Fabio Bussato, TCP Keepalive HOWTO https://tldp.org/HOWTO/html_single/TCP-Keepalive-HOWTO/

[6] Ya-fei Luo, The multi-thread communication of Socket based on TCP

[7] Socket program in Java https://www.geeksforgeeks.org/socket-programming-in-java/