TCP/IP COMMUNICATION

Abstract

TCP provides for the recovery of packets that get lost, are damaged, duplicated or received out of their correct order. TCP is described as a reliable protocol because it attempts to recover from these errors.The sequencing is handled by labling every packet with a sequence number. These sequence numbers permit TCP to detect dropped packets. TCP also requires that an acknowledge message be returned after transmitting data.

Keywords

Socket,IP,port,TCP,stream

Socket

Sockets are communication points on the same or different computers to exchange data.You can think of sockets as the bus between computers. Sockets are supported by Unix, Windows, Mac, and many other operating systems.

Most used socket types are:

- Stream sockets
- Datagram sockets

Stream Sockets

There is no risk of packet loss.Packages are sent and received in order.These sockets use TCP for data transmission.It provides users with a connection-oriented environment.

Datagram Sockets

Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets. They use UDP (User Datagram Protocol).

*In this paper,we will talk about TCP/IP communication.

What is the IP and port?

IP address is the numerical name of a computer on the network.And we can define the port as virtual holes in a computer.For example,when you connect to a website, you connect to the 80 port of the website (is a computer).Because HTTP port is 80.

How is the connection between two computers?

Packets are transmitted securely in the TCP connection.This is due to a 3-way handshake.Shown in figure 1.

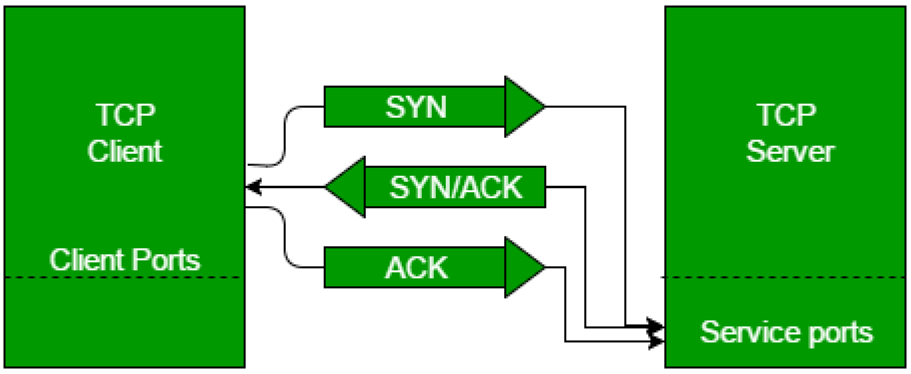


Figure 1

To go through the example,consider a server and client model.

Basicly Server/Client model (shown in figure 2)

<u>Server side:</u>

1.Create a Socket

We need a few features to open a socket on the computer.

- Address family(IPv4 or Ipv6)
- Socket Type
- Protocol Type(basicly TCP or UDP)

2.Bind the opened socket to the local ip address of the computer.

*Here, we need to specify which port the connection will come from. (Specify the port to enable)

3.Place the socket in listen mode(for incoming connections from clients.)

4.Accept an incoming connection on the socket.

<u>Client side:</u>

1.Create a socket

As on the server.

2.Connect the socket to the remote server address

*Here,we will be informed which computer will be connected and which port to connect to.

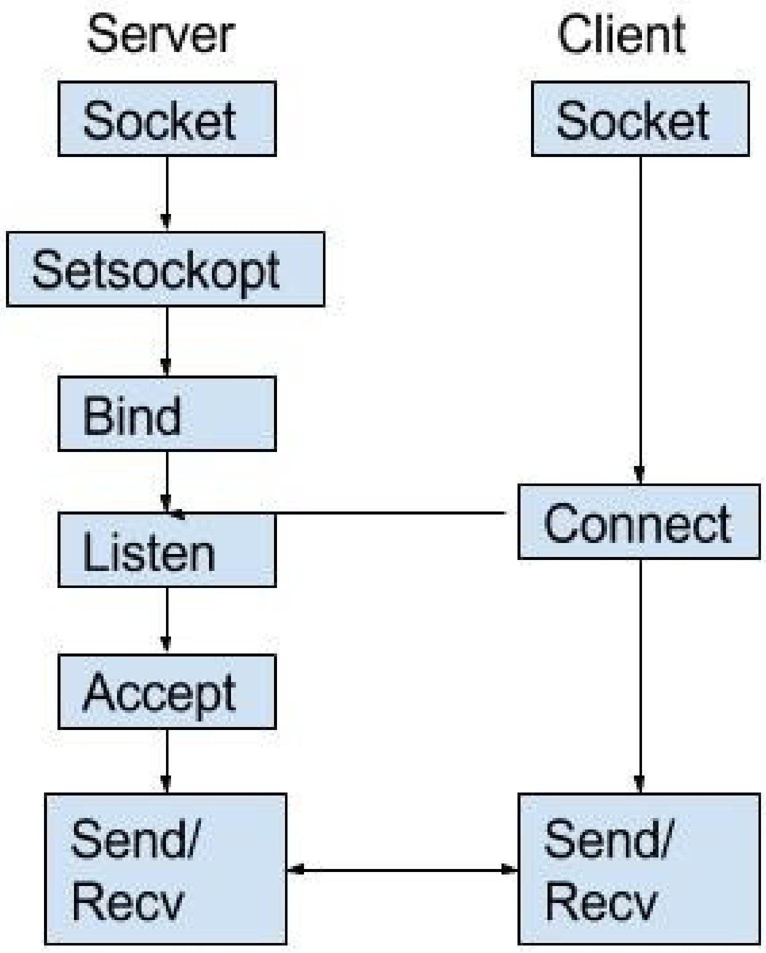


Figure 2

When tcp goes bad

Byte packets sent during communication are collected in the buffer respectively and the accumulated packets are read according to the rule.These rules depend on the imagination of the programmer. These programming techniques are the most commonly used.

- Always send fixed-sized messages
- Send the message size with each message
- Use a marker system to separate messages

In this paper we will talk about marker solution:

In the Marker solution, the sender adds a special character to the end of the packet and sends it.In this way, the receiving party can recognize that the part up to

the special character is 1 pack.Otherwise, since all packets are received in sequence, they do not know how to allocate them.
For example, the server may have timed out.And by the time the sender sent another packet, the server's buffer would be full.Server could not allocate packages.And this is not a desired scenario.In the use of the marker, even if the server's buffer is full, it can separate packets.Shown in Figure 3 the packet exchange between the server and the client (without using a marker).

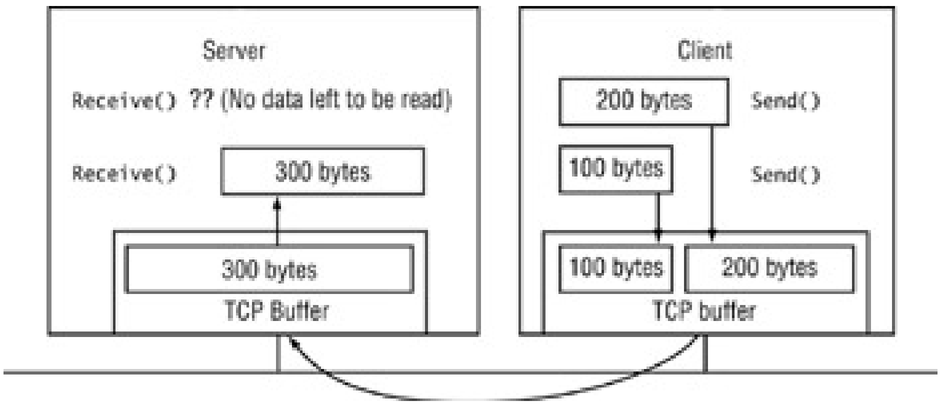
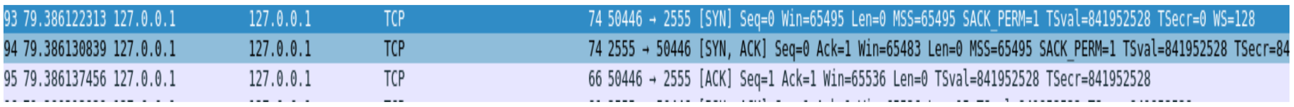


Figure 3

Let's see a sample ‘hello’ text message sent in this project.(with help of Wireshark).

First, the server and the client perform 3-way handshake.Shown figure 4.

*Server port 2555 and client port 50446 on localhost(127.0.0.1).

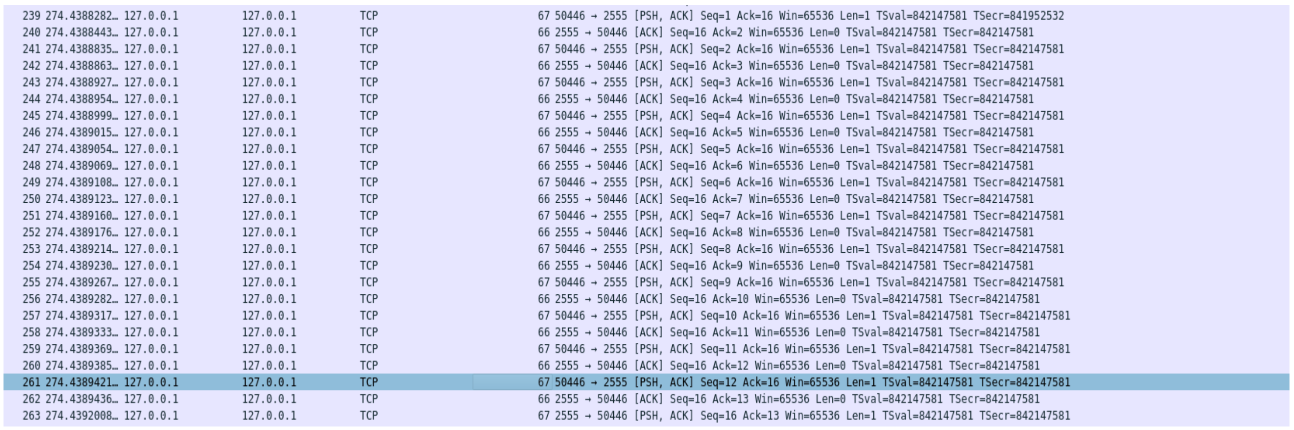


```
93 79.386122313 127.0.0.1      127.0.0.1      TCP      74 50446 → 2555 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=841952528 TSecr=0 WS=128
94 79.386130839 127.0.0.1      127.0.0.1      TCP      74 2555 → 50446 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=841952528 TSecr=84
95 79.386137456 127.0.0.1      127.0.0.1      TCP      66 50446 → 2555 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=841952528 TSecr=841952528
```

Figure 4

Then the client sends a 'hello' message from the console to the server.(byte to byte).

In figure 5,a hello message between 239-261 with marker <##!##>.

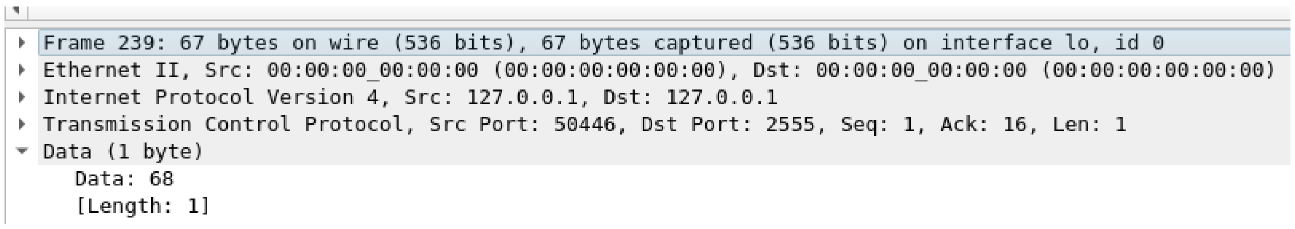


```
239 274.4388282… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=1 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=841952532
240 274.4388443… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=2 Win=65536 Len=0 TSval=842147581 TSecr=842147581
241 274.4388835… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=2 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
242 274.4388863… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=3 Win=65536 Len=0 TSval=842147581 TSecr=842147581
243 274.4388927… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=3 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
244 274.4388954… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=4 Win=65536 Len=0 TSval=842147581 TSecr=842147581
245 274.4388999… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=4 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
246 274.4389015… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=5 Win=65536 Len=0 TSval=842147581 TSecr=842147581
247 274.4389054… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=5 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
248 274.4389069… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=6 Win=65536 Len=0 TSval=842147581 TSecr=842147581
249 274.4389108… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=6 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
250 274.4389123… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=7 Win=65536 Len=0 TSval=842147581 TSecr=842147581
251 274.4389160… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=7 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
252 274.4389176… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=8 Win=65536 Len=0 TSval=842147581 TSecr=842147581
253 274.4389214… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=8 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
254 274.4389230… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=9 Win=65536 Len=0 TSval=842147581 TSecr=842147581
255 274.4389267… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=9 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
256 274.4389282… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=10 Win=65536 Len=0 TSval=842147581 TSecr=842147581
257 274.4389317… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=10 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
258 274.4389333… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=11 Win=65536 Len=0 TSval=842147581 TSecr=842147581
259 274.4389369… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=11 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
260 274.4389385… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=12 Win=65536 Len=0 TSval=842147581 TSecr=842147581
261 274.4389421… 127.0.0.1      127.0.0.1      TCP      67 50446 → 2555 [PSH, ACK] Seq=12 Ack=16 Win=65536 Len=1 TSval=842147581 TSecr=842147581
262 274.4389436… 127.0.0.1      127.0.0.1      TCP      66 2555 → 50446 [ACK] Seq=16 Ack=13 Win=65536 Len=0 TSval=842147581 TSecr=842147581
263 274.4392008… 127.0.0.1      127.0.0.1      TCP      67 2555 → 50446 [PSH, ACK] Seq=16 Ack=13 Win=65536 Len=1 TSval=842147581 TSecr=842147581
```

Figure 5

Notice that there is a ‘acknowledgement’ between the server and the client each time a packet is sent.

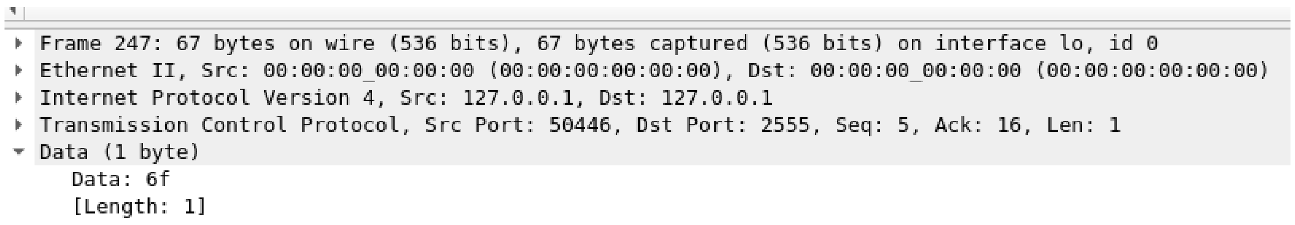
Start ‘hello’ text message with first characher ‘h’ shown figure 6.In data section,data 68 because ‘h’ is hexadecimal ‘68’.



```
▸ Frame 239: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 50446, Dst Port: 2555, Seq: 1, Ack: 16, Len: 1
▾ Data (1 byte)
    Data: 68
    [Length: 1]
```

Figure 6

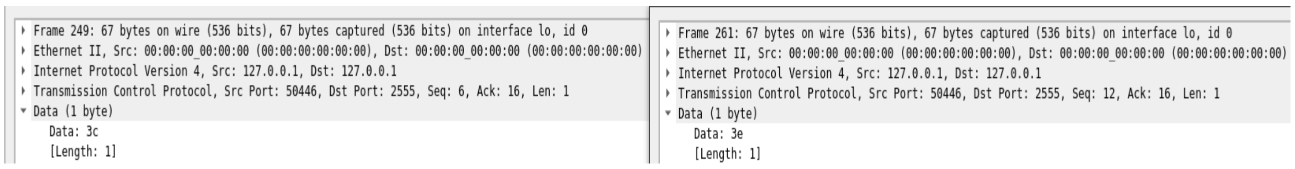
End ‘hello’ text message with last character ‘o’ (hexadecimal 6f) shown figure 7.



```
▸ Frame 247: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 50446, Dst Port: 2555, Seq: 5, Ack: 16, Len: 1
▾ Data (1 byte)
    Data: 6f
    [Length: 1]
```

Figure 7

And finally,start ‘<##!##>’ (marker) first character ‘<’(hexadecimal 3c) and last character ‘>’(hexadecimal 3e) shown figure 8.(between frame 249-261)



```
▸ Frame 249: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 50446, Dst Port: 2555, Seq: 6, Ack: 16, Len: 1
▾ Data (1 byte)
    Data: 3c
    [Length: 1]
```

```
▸ Frame 261: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 50446, Dst Port: 2555, Seq: 12, Ack: 16, Len: 1
▾ Data (1 byte)
    Data: 3e
    [Length: 1]
```

Figure 8

Thank you for reading…

AUTHOR
Abdüssamed Güzey

References:

[1]https://www.tutorialspoint.com/unix_sockets/

[2]https://medium.com/@gokhansengun/tcp-nas%C4%B1l-%C3%A7al%C4%B1%C5%9F%C4%B1r-1-484612c5264f

[3]https://pymotw.com/2/socket/addressing.html

[4]https://tutorialspoint.dev/language/cpp/socket-programming-cc

[5]https://www.geeksforgeeks.org/tcp-3-way-handshake-process/

[6]C# Network Programming by Richard Blum,Chapter 5