



Computer Networks

Midsem Lab

CS F303

Susmit Wani

2018A7PS0116G

Midsem Lab Exam

Vinayak Naik • Mar 13 (Edited 1:07 PM)

Midsem • 18 points

Due Tomorrow, 6:59 PM

Notes

1. All the students will upload C code along with a PDF file containing the screenshots of the executed program on Google classroom
2. Submit a README file giving instruction for compilation and execution
3. Submit the captured pcap file
4. Write the Wireshark filter, measured values, and the justification in the PDF
5. We will use MOSS to detect cheating cases

Question

Write a TCP server and a client for the following.

1. The server as a command line argument accepts the port number to which it should bind. (2 marks)
2. The client, as command line arguments, accepts the IP address and the port number at which it will find the server. (2 marks)
3. After connecting to the server, the client reads a line from the standard input and sends it to the server. (2 marks)
4. The server prints the received line in the reverse order (2 marks) and reads a line from the standard input and sends it to the client. (2 marks)
5. The client prints the received line in the reverse order and exits. The server is ready to accept a new client. (2 marks)
6. Use Wireshark to capture the packets (2 marks) and write a filter that will find what were the sizes of the TCP segment sent from the client to the server. (2 marks) Compare the number of bytes in the line with those observed in the TCP segment. Justify how the values match. (2 marks)



Output of C codes.

1. server.c

```
lenovo@susmits-lenovo:~/Desktop/CN_Midsem_Lab$ gcc -o server.out server.c
lenovo@susmits-lenovo:~/Desktop/CN_Midsem_Lab$ ./server.out 8000
Attempting to create socket on port number 8000
Socket created successfully.
Binded to the server successfully
Server listening
server has accepted the client

olleH
Enter your message:Hello

Waiting for new client.

server has accepted the client

niaga olleH
Enter your message:Okay then

Waiting for new client.
```



Output of C codes.

2. client.c

```
lenovo@susmits-lenovo:~/Desktop/CN_Midsem_Lab$ gcc -o client.out client.c
lenovo@susmits-lenovo:~/Desktop/CN_Midsem_Lab$ ./client.out 127.0.0.1 8000
Connecting to IP address: 127.0.0.1
Attempting to connect to port: 8000
Socket created successfully.
connected to the server successfully
Enter your message:Hello
Message from server:

olleH
lenovo@susmits-lenovo:~/Desktop/CN_Midsem_Lab$ ./client.out 127.0.0.1 8000
Connecting to IP address: 127.0.0.1
Attempting to connect to port: 8000
Socket created successfully.
connected to the server successfully
Enter your message:Hello again
Message from server:

neht yak0
lenovo@susmits-lenovo:~/Desktop/CN_Midsem_Lab$
```

Wireshark packets from the pcapng file and the columns used

Apply a display filter ... <Ctrl-/>									
No.	Time	Source	Destination	Protocol	TCP Data Len	TCP Header Length	Source Port	Destination Port	Info
1	2021-03-14 17:37:06.403175503	192.168.0.6	142.250.18...	UDP					53185 → 443 Len=33
2	2021-03-14 17:37:06.427238888	142.250.18...	192.168.0.6	UDP					443 → 53185 Len=25
3	2021-03-14 17:37:07.730680289	192.168.0.6	224.0.0.251	IGMPv2					Membership Report group 224.0
4	2021-03-14 17:37:09.302995709	192.168.0.6	172.217.16...	TCP	0		32 53808	443	53808 → 443 [FIN, ACK] Seq=1
5	2021-03-14 17:37:09.317223763	172.217.16...	192.168.0.6	TCP	0		32 443	53808	443 → 53808 [FIN, ACK] Seq=1
6	2021-03-14 17:37:09.317272216	192.168.0.6	172.217.16...	TCP	0		32 53808	443	53808 → 443 [ACK] Seq=2 Ack=2
7	2021-03-14 17:37:09.775445733	31.13.79...	192.168.0.6	TLSv1.2	336		32 443	36952	Application Data
8	2021-03-14 17:37:09.775470842	192.168.0.6	31.13.79.53	TCP	0		32 36952	443	36952 → 443 [ACK] Seq=1 Ack=3
9	2021-03-14 17:37:12.828097839	192.168.0.6	142.250.18...	UDP					53185 → 443 Len=33
10	2021-03-14 17:37:12.849001575	142.250.18...	192.168.0.6	UDP					443 → 53185 Len=25
11	2021-03-14 17:37:13.656887162	74.125.2...	192.168.0.6	UDP					443 → 47559 Len=46
12	2021-03-14 17:37:13.675351608	192.168.0.6	74.125.200...	UDP					47559 → 443 Len=33
13	2021-03-14 17:37:15.872460073	127.0.0.1	127.0.0.1	TCP	0		40 48890	8000	48890 → 8000 [SYN] Seq=0 Win=
14	2021-03-14 17:37:15.872490570	127.0.0.1	127.0.0.1	TCP	0		40 8000	48890	8000 → 48890 [SYN, ACK] Seq=0
15	2021-03-14 17:37:15.872519291	127.0.0.1	127.0.0.1	TCP	0		32 48890	8000	48890 → 8000 [ACK] Seq=1 Ack=
▶ Frame 1: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0									
▶ Linux cooked capture									
▶ Internet Protocol Version 4, Src: 192.168.0.6, Dst: 142.250.182.238									
▶ User Datagram Protocol, Src Port: 53185, Dst Port: 443									
▶ Data (33 bytes)									
0000	00 04 00 01 00 06 70 c9	4e d2 91 d7 00 00 08 00p..N.....						
0010	45 00 00 3d 46 b8 40 00	40 11 ed 60 c0 a8 00 06	E...F.@. @.....						
0020	8e fa b6 ee cf c1 01 bb	00 29 06 d2 44 c1 f0 81f....).D....						
0030	5d 77 87 0c b7 53 61 a6	04 5f 89 72 0c ba b3 ba]w...Sa...r....						
0040	86 e3 48 86 5d a4 79 7f	f0 eb 99 02 b9	..H..]y.....						

The columns TCP Data Len and TCP Header Len are custom added columns.

TCP Data Len: tcp.len

TCP Header Length: tcp.hdr_len

Filter to be used to display packets we want:

`tcp.dstport==8000 && ip.src==127.0.0.1 && ip.dst==127.0.0.1`

tcp.dstport==8000 && ip.src==127.0.0.1 && ip.dst==127.0.0.1										
No.	Time	Source	Destination	Protocol	TCP Data Len	TCP Header Length	Source Port	Destination Port	Info	
13	2021-03-14 17:37:15.872460073	127.0.0.1	127.0.0.1	TCP	0	40	48890	8000	48890 → 8000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SAC	
15	2021-03-14 17:37:15.872519291	127.0.0.1	127.0.0.1	TCP	0	32	48890	8000	48890 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2	
16	2021-03-14 17:37:19.242784377	127.0.0.1	127.0.0.1	TCP	13	32	48890	8000	48890 → 8000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=13 T	
23	2021-03-14 17:37:22.796278382	127.0.0.1	127.0.0.1	TCP	0	32	48890	8000	48890 → 8000 [ACK] Seq=14 Ack=14 Win=65536 Len=0 TSval	
24	2021-03-14 17:37:22.796698174	127.0.0.1	127.0.0.1	TCP	0	32	48890	8000	48890 → 8000 [FIN, ACK] Seq=14 Ack=14 Win=65536 Len=0	
26	2021-03-14 17:37:24.428497087	127.0.0.1	127.0.0.1	TCP	0	40	48892	8000	48892 → 8000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SAC	
28	2021-03-14 17:37:24.428558266	127.0.0.1	127.0.0.1	TCP	0	32	48892	8000	48892 → 8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2	
69	2021-03-14 17:37:31.972628396	127.0.0.1	127.0.0.1	TCP	17	32	48892	8000	48892 → 8000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=17 T	
76	2021-03-14 17:37:38.385163451	127.0.0.1	127.0.0.1	TCP	0	32	48892	8000	48892 → 8000 [ACK] Seq=18 Ack=16 Win=65536 Len=0 TSval	
77	2021-03-14 17:37:38.385527619	127.0.0.1	127.0.0.1	TCP	0		32	48892	8000	48892 → 8000 [FIN, ACK] Seq=18 Ack=16 Win=65536 Len=0

▶ Frame 16: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface 0

▶ Linux cooked capture

▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

▶ Transmission Control Protocol, Src Port: 48890, Dst Port: 8000, Seq: 1, Ack: 1, Len: 13

▶ Data (13 bytes)

0000	00 00 03 04 00 06 00 00 00 00 00 00 00 08 00
0010	45 00 00 41 cc df 40 00 40 06 6f d5 7f 00 00 01	E..A..@..@o..
0020	7f 00 00 01 be fa 1f 40 b2 9c a3 6b 0e a3 60 ae@..k..
0030	80 18 02 00 fe 35 00 00 01 01 08 0a 79 ef f7 dc5.....y..
0040	79 ef ea b2 48 65 6c 6c 6f 20 73 65 72 76 65 72	y..Hell o server
0050	0a	.

tcp.dstport allows us to get packets which were sent to port 8000. This along with the ip.dst and ip.src allow us to capture exactly those packets which we want, i.e. those packets which were sent from client to server.

The file in the submission contains only these tcp dumps. The previous slide is only an example of the pcap output.



Comparison between bytes in line and TCP segment

3.5.2 TCP Segment Structure

Having taken a brief look at the TCP connection, let's examine the TCP segment structure. The TCP segment consists of header fields and a data field. The data field contains a chunk of application data. As mentioned above, the MSS limits the

We can know that the TCP segment is made up of header+data. So as we can see in the TCP header length column, the length is 32 bytes for ACK message and 40 for SYN message. This is because in SYN we need to synchronise the sequence numbers, hence the extra size.

Now if we check the data messages, the first message is "Hello server\n" which is 13 bytes long. The second data message is 17 bytes long and it says "this is client 2\n".

The TCP data length column denotes the length of data sent via the TCP message. It is 0 in case of ACK and SYN and 13 and 17 where actual data is being sent.

Thus we can clearly see that the TCP segment size is equal to 32 plus the data length sent which is nothing but bytes in line sent.