EL5373
INTERNET ARCHITECTURE AND ROTOCOLS

Rueilong Wu

0464979

Rwu06@students.poly.edu

Workstation: Fenchi
MAC: 00:14:6c:2e:66:08
Lab Report 6
Due 25 November 2012

Exercise 1

TCP connection establishment:

```
21:09:46.404073 fenchi.32942 > 128.238.66.104.time: S 3562299761:3562299761(0) win 5840 <mss
1460,sackOK,timestamp 86740 0,nop,wscale 0> (DF) [tos 0x10]
21:09:46.404291 128.238.66.104.time > fenchi.32942: S 70557543:70557543(0) ack 3562299762 win 5792
<mss 1460,sackOK,timestamp 81115 86740,nop,wscale 0> (DF)
21:09:46.404319 fenchi.32942 > 128.238.66.104.time: . ack 70557544 win 5840 <nop,nop,timestamp 86740
81115> (DF) [tos 0x10]
```

1.My machine ,128.238.66.105, sent a SYN packet (seq# = 3562299761, mss = 1460, win = 5840) to remote host , 128.238.66.104, to request a TCP connection.

2.Remote Host, 128.238.66.104, replied packet with a SYN packet (seq# = 70557543, ack# = 3562299762, mss = 1460, win = 5792).

3.My machine acknowledged the byte with ack# = 70557544. Thus, the TCP connection was established

TCP connection termination:

```
21:09:46.405009 128.238.66.104.time > fenchi.32942: F 70557548:70557548(0) ack 3562299762 win 5792
<nop,nop,timestamp 81115 86740> (DF)
21:09:46.405366 fenchi.32942 > 128.238.66.104.time: . ack 70557549 win 5840 <nop,nop,timestamp 86740
81115> (DF) [tos 0x10]
21:09:46.406734 fenchi.32942 > 128.238.66.104.time: F 3562299762:3562299762(0) ack 70557550 win 5840
<nop,nop,timestamp 86740 81115> (DF) [tos 0x10]
21:09:46.406919 128.238.66.104.time > fenchi.32942: . ack 3562299763 win 5792 <nop,nop,timestamp 81115
86740> (DF)
```

1. Remote host,128.238.66.104, sent a FIN packet (seq# = 70557548, ack# = 3562299762) to my machine to indicate the termination of the one-way data flow.

2. My machine acknowledged the byte with ack# = 70557549. Thus, host (66.100) terminated the one-way data flow.

3. My machine sent a FIN packet (seqNo = 3562299762, ackNo =70557550) to host (66.100) to indicate the termination of the one-way data flow.

4.Remotw Host ,128.238.66.104, acknowledged the byte with ackNo =3562299763. Thus, the TCP connection was terminated.

Both hosts announced that the MSS = 1460, and the DF (Don't Fragment) flag was set.

If there is an intermediate network that has an MTU less than the MSS of each host:

1. If DF is set. The IP datagram will be dropped and an ICMP unreachable error will be sent to the source carrying the MTU of the next link. The MSS will then be re-calculated.

2. If DF is not set. The IP datagram will be fragmented by the router in the intermediate network.

Exercise 2

For UDP, when the port is not available, the remote host will send an ICMP unreachable error message.

```
[guest@fenchi rueilongwu]$ sudo tcpdump -nx host 128.238.66.105 and 128.238.66.104
tcpdump: listening on eth0
21:14:47.032040 128.238.66.105.32771 > 128.238.66.104.8888: udp 1024 (DF)
                        4500 041c c843 4000 4011 e7df 80ee 4269
                        80ee 4268 8003 22b8 0408 c849 2021 2223
                        2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                        3435 3637 3839 3a3b 3c3d 3e3f 4041 4243
                        4445 4647 4849 4a4b 4c4d 4e4f 5051 5253
                        5455
21:14:47.033479 128.238.66.104 > 128.238.66.105: icmp: 128.238.66.104 udp port 8888 unreachable [tos
0xc0]
                        45c0 0240 8b1d 0000 4001 6632 80ee 4268
                        80ee 4269 0303 29a4 0000 0000 4500 041c
                        c843 4000 4011 e7df 80ee 4269 80ee 4268
                        8003 22b8 0408 c849 2021 2223 2425 2627
                        2829 2a2b 2c2d 2e2f 3031 3233 3435 3637
                        3839
21:14:52.025226 arp who-has 128.238.66.105 tell 128.238.66.104
                        0001 0800 0604 0001 0009 5b0a ea03 80ee
                        4268 0000 0000 0000 80ee 4269 ffff ffff
                        ffff ffff ffff ffff ffff ffff ffff
21:14:52.025252 arp reply 128.238.66.105 is-at 0:14:6c:2e:66:8
                        0001 0800 0604 0002 0014 6c2e 6608 80ee
                        4269 0009 5b0a ea03 80ee 4268
```

For TCP, when the port is not available, the remote host will send a RST packet to the host machine and the TCP connection is need to reset.

```
[guest@fenchi rueilongwu]$ sudo tcpdump -nx host 128.238.66.105 and 128.238.66.104
tcpdump: listening on eth0
21:16:04.081878 128.238.66.105.33019 > 128.238.66.104.8888: S 3945879715:3945879715(0) win 5840 <mss
1460,sackOK,timestamp 124508 0,nop,wscale 0> (DF)
                        4500 003c beff 4000 4006 f50e 80ee 4269
                        80ee 4268 80fb 22b8 eb31 58a3 0000 0000
                        a002 16d0 dca2 0000 0204 05b4 0402 080a
                        0001 e65c 0000 0000 0103 0300
21:16:04.082102 128.238.66.104.8888 > 128.238.66.105.33019: R 0:0(0) ack 3945879716 win 0 (DF)
                        4500 0028 0000 4000 4006 b422 80ee 4268
                        80ee 4269 22b8 80fb 0000 0000 eb31 58a4
                        5014 0000 4199 0000 ffff ffff ffff
21:16:09.074621 arp who-has 128.238.66.105 tell 128.238.66.104
                        0001 0800 0604 0001 0009 5b0a ea03 80ee
                        4268 0000 0000 0000 80ee 4269 ffff ffff
                        ffff ffff ffff ffff ffff ffff ffff
21:16:09.074648 arp reply 128.238.66.105 is-at 0:14:6c:2e:66:8
                        0001 0800 0604 0002 0014 6c2e 6608 80ee
                        4269 0009 5b0a ea03 80ee 4268
```

IN the nonexisting sever cases

The host both use broadcast to find another host where it is.

Exercise 3

(1)host 2, receiving a data segment, TCP delays sending the ACK to allow the host, let's say host 1, on the other end to collect new data from its application layer. Thus, by the time it receives the delayed ACK from host 2, host 1 will send the ACK piggybacked with the new data segment. The delayed ACK reduces the number of small segments; therefore, improve the efficiency of the bandwidth.

(2)Yes, the delayed ACK was observed. When keystrokes are fast enough, which means host 1 collects new data before the delayed acknowledgment timer expires; host 1 sends the ACK piggybacked with the new data.

(3)Each TCP connection can have only one small segment outstanding that has not been acknowledged. For interactive data flows, TCP sends one byte and buffer all subsequent bytes until an acknowledgment for the byte is received. Thus, all buffered bytes will be sent in a single segment. Higher bandwidth efficiency is achieved but delay increases. From the tcpdump output, Nagle algorithm is enabled. This can be seen that for every byte sent, there is an acknowledgement and the sequence number is always incremented by one byte since telnet always transmits the keystroke immediately which consists of one byte per keystroke.

When typing very rapidly, there is no observation more than one character packet going from client to host.

```
[guest@fenchi rueilongwu]$ sudo tcpdump -nx host 128.238.66.105 and 128.238.66.1
04
tcpdump: listening on eth0
21:25:33.113454 128.238.66.105.33136 > 128.238.66.104.telnet: S 251075461:251075
461(0) win 5840 <mss 1460,sackOK,timestamp 181411 0,nop,wscale 0> (DF) [tos 0x10
]
21:25:33.113691 128.238.66.104.telnet > 128.238.66.105.33136: S 1067273944:10672
73944(0) ack 251075462 win 5792 <mss 1460,sackOK,timestamp 175787 181411,nop,wsc
ale 0> (DF)

21:25:33.113719 128.238.66.105.33136 > 128.238.66.104.telnet: . ack 1 win 5840 <
nop,nop,timestamp 181411 175787> (DF) [tos 0x10]

21:25:33.116305 128.238.66.105.33136 > 128.238.66.104.telnet: P 1:28(27) ack 1 w
in 5840 <nop,nop,timestamp 181411 175787> (DF) [tos 0x10]

21:25:33.116533 128.238.66.104.telnet > 128.238.66.105.33136: P 1:13(12) ack 1 w
in 5792 <nop,nop,timestamp 175787 181411> (DF) [tos 0x10]

21:25:33.116552 128.238.66.105.33136 > 128.238.66.104.telnet: . ack 13 win 5840
<nop,nop,timestamp 181411 175787> (DF) [tos 0x10]

21:25:33.116604 128.238.66.104.telnet > 128.238.66.105.33136: . ack 28 win 5792
<nop,nop,timestamp 175787 181411> (DF) [tos 0x10]

21:25:33.117297 128.238.66.104.telnet > 128.238.66.105.33136: P 13:52(39) ack 28
 win 5792 <nop,nop,timestamp 175787 181411> (DF) [tos 0x10]

21:25:33.117305 128.238.66.105.33136 > 128.238.66.104.telnet: . ack 52 win 5840
<nop,nop,timestamp 181411 175787> (DF) [tos 0x10]
21:25:33.128920 128.238.66.105.33136 > 128.238.66.104.telnet: P 28:141(113) ack
52 win 5840 <nop,nop,timestamp 181413 175787> (DF) [tos 0x10]

21:25:33.214639 128.238.66.104.telnet > 128.238.66.105.33136: P 52:55(3) ack 141
 win 5792 <nop,nop,timestamp 175797 181413> (DF) [tos 0x10]

21:25:33.214977 128.238.66.105.33136 > 128.238.66.104.telnet: P 141:144(3) ack 5
5 win 5840 <nop,nop,timestamp 181421 175797> (DF) [tos 0x10]

21:25:33.215563 128.238.66.104.telnet > 128.238.66.105.33136: P 55:126(71) ack 1
44 win 5792 <nop,nop,timestamp 175797 181421> (DF) [tos 0x10]

21:25:33.215817 128.238.66.105.33136 > 128.238.66.104.telnet: P 144:147(3) ack 1
26 win 5840 <nop,nop,timestamp 181421 175797> (DF) [tos 0x10]

21:25:33.216008 128.238.66.104.telnet > 128.238.66.105.33136: P 126:133(7) ack 1
47 win 5792 <nop,nop,timestamp 175797 181421> (DF) [tos 0x10]

21:25:33.248547 128.238.66.105.33136 > 128.238.66.104.telnet: . ack 133 win 5840
```

```
 <nop,nop,timestamp 181425 175797> (DF) [tos 0x10]

21:25:35.117985 128.238.66.105.33136 > 128.238.66.104.telnet: P 147:148(1) ack 1
33 win 5840 <nop,nop,timestamp 181611 175797> (DF) [tos 0x10]

21:25:35.118534 128.238.66.104.telnet > 128.238.66.105.33136: P 133:134(1) ack 1
48 win 5792 <nop,nop,timestamp 175987 181611> (DF) [tos 0x10]

21:25:35.118566 128.238.66.105.33136 > 128.238.66.104.telnet: . ack 134 win 5840
 <nop,nop,timestamp 181612 175987> (DF) [tos 0x10]
```

Exercise 4

Every time the sender send a TCP message, the receiver acknowledge the TCP message with the *acknowledge number* which is set to the same value as the *sequence number plus the bytes in the data segment* in the received message.

There is no difference between the number of data segments and their acknowledgements

There is *no difference* among all three tcpdump outputs.

```
[guest@fenchi rueilongwu]$ sudo tcpdump -nx host 128.238.66.105 and 128.238.66.104
tcpdump: listening on eth0
21:37:58.959971 128.238.66.105.33291 > 128.238.66.104.7777: S 1049557841:1049557841(0) win 5840 <mss
1460,sackOK,timestamp 255996 0,nop,wscale 0> (DF)
                        4500 003c 1cad 4000 4006 9761 80ee 4269
                        80ee 4268 820b 1e61 3e8e fb51 0000 0000
                        a002 16d0 e83c 0000 0204 05b4 0402 080a
                        0003 e7fc 0000 0000 0103 0300
21:37:58.960178 128.238.66.104.7777 > 128.238.66.105.33291: R 0:0(0) ack 1049557842 win 0 (DF)
                        4500 0028 0000 4000 4006 b422 80ee 4268
                        80ee 4269 1e61 820b 0000 0000 3e8e fb52
                        5014 0000 4ed5 0000 ffff ffff ffff
21:37:59.691082 128.238.66.104.33045 > 128.238.66.105.telnet: P 702156587:702156589(2) ack 4195152692
win 5840 <nop,nop,timestamp 250445 255233> (DF) [tos 0x10]
                        4510 0036 28ef 4000 4006 8b15 80ee 4268
                        80ee 4269 8115 0017 29da 0f2b fa0c f334
                        8018 16d0 6d6c 0000 0101 080a 0003 d24d
                        0003 e501 0d00
21:37:59.691330 128.238.66.105.telnet > 128.238.66.104.33045: P 1:3(2) ack 2 win 5792
<nop,nop,timestamp 256069 250445> (DF) [tos 0x10]
                        4510 0036 af0c 4000 4006 04f8 80ee 4269
                        80ee 4268 0017 8115 fa0c f334 29da 0f2d
                        8018 16a0 6a4c 0000 0101 080a 0003 e845
                        0003 d24d 0d0a
21:37:59.691528 128.238.66.104.33045 > 128.238.66.105.telnet: . ack 3 win 5840 <nop,nop,timestamp
250445 256069> (DF) [tos 0x10]
                        4510 0034 28f0 4000 4006 8b16 80ee 4268
```

```
80ee 4269 8115 0017 29da 0f2d fa0c f336
8010 16d0 772e 0000 0101 080a 0003 d24d
0003 e845
```

There are ***4 different flags*** as can be seen from the tcpdump output:
SYN----Synchronize sequence numbers.
ACK----Acknowledge the previous TCP message.
PSH----The receiver should pass all the data received to the application as soon as possible.
FIN----No more data from sender.
There are ***5 different options*** as can be seen from the tcpdump output:

Maximum segment size----Maximum segment size.
SACK----Selective acknowledgement, followed by a list of 1-4 blocks being selectively acknowledged, specified as 32-bit begin/end pointers.
Timestamps----Timestamp and echo of previous timestamp.
NOP----No operation (padding), used to align option fields on 32-bit boundaries for better performance.
Window scale----window scaling factor to expand window size.

Exercise 5

The default value of the TCP keepavive timer is 7200.

The maximum number of the TCP keepalive probes a host can send is 9.

```
[guest@fenchi rueilongwu]$ sysctl -a | grep keepalive
error: permission denied on key 'net.unix.max_dgram_qlen'
error: permission denied on key 'kernel.pid_max'
error: permission denied on key 'kernel.cad_pid'
error: permission denied on key 'kernel.cap-bound'
net.ipv4.tcp_keepalive_intvl = 75
net.ipv4.tcp_keepalive_probes = 9
net.ipv4.tcp_keepalive_time = 7200
```

Exercise 6

The cable was disconnected since time : 1.691021

When the cable was disconnected, my computer retransmitted every double time.There were two data segments after connection was reestablished.

```
    244 1.468856     128.238.66.104        128.238.66.105        TCP       8888 > 33674 [ACK]
Seq=3784263786 Ack=2980198498 Win=2896 Len=0
    245 1.468862     128.238.66.105        128.238.66.104        TCP       33674 > 8888 [PSH, ACK]
Seq=2980199946 Ack=3784263786 Win=5840 Len=1448
    246 1.691021     128.238.66.105        128.238.66.104        TCP       33674 > 8888 [ACK]
Seq=2980198498 Ack=3784263786 Win=5840 Len=1448
```

```
    247 2.151031    128.238.66.105        128.238.66.104        TCP      33674 > 8888 [ACK]
Seq=2980198498 Ack=3784263786 Win=5840 Len=1448
    248 3.071026    128.238.66.105        128.238.66.104        TCP      33674 > 8888 [ACK]
Seq=2980198498 Ack=3784263786 Win=5840 Len=1448
    249 4.911038    128.238.66.105        128.238.66.104        TCP      33674 > 8888 [ACK]
Seq=2980198498 Ack=3784263786 Win=5840 Len=1448
    250 8.591033    128.238.66.105        128.238.66.104        TCP      33674 > 8888 [ACK]
Seq=2980198498 Ack=3784263786 Win=5840 Len=1448
    251 15.951039   128.238.66.105        128.238.66.104        TCP      33674 > 8888 [ACK]
Seq=2980198498 Ack=3784263786 Win=5840 Len=1448
    252 15.952428   128.238.66.104        128.238.66.105        TCP      8888 > 33674 [ACK]
Seq=3784263786 Ack=2980201394 Win=50680 Len=0
    253 15.952461   128.238.66.105        128.238.66.104        TCP      33674 > 8888 [ACK]
Seq=2980201394 Ack=3784263786 Win=5840 Len=1448
    254 15.952464   128.238.66.105        128.238.66.104        TCP      33674 > 8888 [ACK]
Seq=2980202842 Ack=3784263786 Win=5840 Len=1448
```

Exercise 7

There is no IP fragmentation. Because the MSS = 1460. The IP datagram = 1460 + 20 (TCP header) + 20 (IP header) = 1500 bytes = MTU of the Ethernet

I think the reason is that TCP is designed to provide upper layer user with a byte stream service, while internally TCP treats data as TCP segments, and their sizes are bounded by the MSS so that IP fragmentation is not performed. Whenever a user uses TCP to send something, it sends data byte by byte to TCP, so there is always something that has been sent till the maximum size constraint is reached.

```
[guest@fenchi rueilongwu]$ sudo tcpdump src host 128.238.66.105
tcpdump: listening on eth0
22:21:12.905203 fenchi.33830 > 128.238.66.104.echo: S 3779544419:3779544419(0) w
in 5840 <mss 1460,sackOK,timestamp 515390 0,nop,wscale 0> (DF)
22:21:12.905473 fenchi.33830 > 128.238.66.104.echo: . ack 292310398 win 5840 <no
p,nop,timestamp 515390 509769> (DF)
22:21:12.905948 fenchi.33830 > 128.238.66.104.echo: . 0:1448(1448) ack 1 win 584
0 <nop,nop,timestamp 515390 509769> (DF)
22:21:12.905956 fenchi.33830 > 128.238.66.104.echo: . 1448:2896(1448) ack 1 win
5840 <nop,nop,timestamp 515390 509769> (DF)
22:21:12.908563 fenchi.33830 > 128.238.66.104.echo: P 2896:4344(1448) ack 1 win
5840 <nop,nop,timestamp 515391 509769> (DF)
22:21:12.908567 fenchi.33830 > 128.238.66.104.echo: . 4344:5792(1448) ack 1 win
5840 <nop,nop,timestamp 515391 509769> (DF)
22:21:12.909414 fenchi.33830 > 128.238.66.104.echo: . ack 1025 win 7168 <nop,nop
,timestamp 515391 509769> (DF)
```