# HW3 Solutions

### Problem 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Source IP | Destination IP | source port# | destination port# |
| P1P2 | A | S | a | s |
| P3P2 | B | S | b1 | s |
| P4P2 | B | S | b2 | s |
| P2P4 | S | B | s | b2 |

1. Yes, a and b1 could be the same.
2. No, b2 and b1 cannot be the same.

1. HTTP runs over TCP, and hence the sockets are TCP sockets. As TCP socket is uniquely addressed by the 4-typle (source IP, destination IP, source port number, destination port number), the server needs to create three sockets, one for each client process.

### Problem 2

1. The 8-bit ASCII codes for “Hello world!” is:

|  |  |  |  |
| --- | --- | --- | --- |
| Value | Binary | Value | Binary |
| H | 0100 1000 | w | 0111 0111 |
| e | 0110 0101 | o | 0110 1111 |
| l | 0110 1100 | r | 0111 0010 |
| l | 0110 1100 | l | 0110 1100 |
| o | 0110 1111 | d | 0110 0100 |
| space | 0010 0000 | ! | 0010 0001 |

We can compute checksum as:

0100 1000 0110 0101 “He”

+0110 1100 0110 1100 “ll”

1011 0100 1101 0001

+0110 1111 0010 0000 “o ”

10010 0011 1111 0001

Wrap around

0010 0011 1111 0010

+0111 0111 0110 1111 “wo”

1001 1011 0110 0001

+0111 0010 0110 1100 “rl”

10000 1101 1100 1101

Wrap around

0000 1101 1100 1110

+0110 0100 0010 0001 “d!”

0111 0001 1110 1111

So the checksum is: 1000 1110 0001 0000

1. No. As long as there is a 1-bit error, it’ll be detected.
2. Yes, a 2-bit error could be undetected. For example, if there are two bit errors 16 bits apart, then the errors will cancel out when summing up the 16-bit binary numbers, and the errors will not be detected.

### Problem 3

Since we are currently assuming a channel that does not lose packets, we know that the reason that receiver receives the wrong sequence number is because the sender retransmitted the packet due to a corrupted acknowledgement. Therefore, if the receiver still responds NAK, the sender will continue to retransmit the same packet, which will result in a deadlock.

Problem 4

(7) rdt\_rcv(rcvpkt)&&from\_B(rcvpkt)

sndpkt=make\_pkt(ACK, 0, checksum)

udt\_send(B,sndpkt)

Note: It is also correct to have empty action (i.e., Λ).

(8) rdt\_rcv(rcvpkt) && (corrupt(rcvpkt) || has\_seq0(rcvpkt)) && from\_A(rcvpkt)

sndpkt=make\_pkt(ACK, 0, checksum)

udt\_send(A,sndpkt)

(9) rdt\_rcv(rcvpkt)&&not\_corrupt(rcvpkt)&&has\_seq1(rcvpkt)&&from\_A(rcvpkt)

extract(rcvpkt,data)

deliver\_data(data)

sndpkt=make\_pkt(ACK, 1, checksum)

udt\_send(A,sndpkt)

(10) rdt\_rcv(rcvpkt)&&from\_A(rcvpkt)

sndpkt=make\_pkt(ACK, 1, checksum)

udt\_send(A,sndpkt)

Note: It is also correct to have empty action (i.e., Λ).

(11) rdt\_rcv(rcvpkt) && (corrupt(rcvpkt) ||has\_seq0(rcvpkt)) && from\_B(rcvpkt)

sndpkt=make\_pkt(ACK, 0, checksum)

udt\_send(B,sndpkt)

(12) rdt\_rcv(rcvpkt)&&not\_corrupt(rcvpkt)&&has\_seq1(rcvpkt)&&from\_B(rcvpkt)

extract(rcvpkt,data)

deliver\_data(data)

sndpkt=make\_pkt(ACK,1,checksum)

udt\_send(B,sndpkt)

### Problem 5

Consider the one-to-two rdt. Host A wants to simultaneously send packets to host B and C.

The sender and receiver FSM are shown as follows. In this problem, the sender state indicates whether the sender has received an ACK from B (only), from C (only) or from neither C nor B.

The receiver state indicates which sequence number the receiver is waiting for.

(9) rdt\_rcv(rcvpkt)

Λ

(10) rdt\_send(data)

sndpkt=make\_pkt(1,data,checksum)

udt\_send(sndpkt)

start\_timer

(11) timeout

udt\_send(sndpkt)

start\_timer

(12) rdt\_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0,B) || isACK(rcvpkt,0,C))

Λ

(13) rdt\_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1,B))

Λ

(14) rdt\_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1,C))

Λ

(15) timeout

udt\_send(sndpkt)

start\_timer

(16) rdt\_rcv(rcvpkt) && (corrupt(rcvpkt) || !isACK(rcvpkt,1,C))

Λ

(17) rdt\_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1,C))

Λ

(18) timeout

udt\_send(sndpkt)

start\_timer

(19) rdt\_rcv(rcvpkt) && (corrupt(rcvpkt) || !isACK(rcvpkt,1,B))

Λ

(20) rdt\_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1,B))

Λ

### Problem 6

1. In-order and successful:

Sequence number: 201, 301, 361;

ACK number: 301, 361, 481.

1. The first segment sent by A is lost:

Sequence number: 201, 301, 361, 201;

ACK number: 201, 201, 481.

1. The first acknowledgement sent by B is lost:

Sequence number: 201, 301, 361, 201;

ACK number: 361, 481, 481.

1. The second segment sent by A arrives after the third segment:

Sequence number: 201, 301, 361;

ACK number: 301, 301, 481.

### Problem 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SampleRTT(ms) | 70 | 110 | 102 | 90 | 115 |
| EstimatedRTT(ms) | 78.8 | 82.5 | 84.9 | 85.5 | 89.0 |

1. If EstimatedRTT is updated first (upon receiving each SampleRTT),

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SampleRTT(ms) | 70 | 110 | 102 | 90 | 115 |
| DevRTT(ms) | 17.4 | 19.7 | 19.1 | 15.8 | 18.1 |

If DevRTT is updated first,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SampleRTT(ms) | 70 | 110 | 102 | 90 | 115 |
| DevRTT(ms) | 17.7 | 20.8 | 20.5 | 17.0 | 19.8 |

1. If EstimatedRTT is updated first,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SampleRTT(ms) | 70 | 110 | 102 | 90 | 115 |
| TimeoutInterval(ms) | 148.5 | 161.5 | 161.4 | 148.6 | 161.5 |

If DevRTT is updated first,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SampleRTT(ms) | 70 | 110 | 102 | 90 | 115 |
| TimeoutInterval(ms) | 149.6 | 165.8 | 166.9 | 153.3 | 168.4 |

Note: As the textbook did not specify which variable should be updated first, both versions of solutions will be considered correct.

### Problem 8

1. GBN:

Host A will send 12 segments in total. Host B will send 11 ACKs in total.

SR:

Host A will send 7 segments in total. Host B will send 6 ACKs in total.

TCP:

Host A will send 7 segments in total. Host B will send 6 ACKs in total.

1. GBN:

Sequence number of A:1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6. ACK number of B: 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6..

SR:

Sequence number of A:1, 2, 3, 4, 5, 6, 1. ACK number of B: 2, 3, 4, 5, 6, 1.

TCP:

Sequence number of A:.1, 2, 3, 4, 5, 6, 1. ACK number of B: 1, 1, 1, 1, 1, 7.

1. TCP. This is because TCP uses fast retransmit without waiting until time out.

### Problem 9 (optional)

1. TCP slowstart is operating in the intervals [1,6) and [23,26].
2. TCP congestion avoidance is operating in the intervals [6,16] and [17,22]. Note: During the 16th round, there must be a triple duplicate ACK which moves TCP temporarily into fast recovery state. But the new ACK must have arrived before timeout, which moves TCP back into congestion avoidance state.
3. After the 16th transmission round, packet loss is recognized by a triple duplicate ACK. If there was a timeout, the congestion window size would have dropped to 1.
4. After the 22nd transmission round, segment loss is detected due to timeout, and hence the congestion window size is set to 1.
5. The threshold is initially 32 MSS, since it is at this window size that slow start stops and congestion avoidance begins.
6. The ssthresh is set to half of the value of the congestion window when packet loss is detected. When loss is detected during transmission round 16, the congestion windows size is 42 MSS. Hence the threshold is 21 MSS during the 18th transmission round.
7. The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 22, the congestion windows size is 26 MSS. Hence the threshold is 13 MSS during the 24th transmission round.
8. During the 1st transmission round, packet 1 is sent; packet 2-3 are sent in the 2nd transmission round; packets 4-7 are sent in the 3rd transmission round; packets 8-15 are sent in the 4th transmission round; packets 16-31 are sent in the 5th transmission round; packets 32-63 are sent in the 6th transmission round; packets 64 – 96 are sent in the 7th transmission round. Thus packet 70 is sent in the 7th transmission round.
9. Upon receiving three duplicate ACKs, ssthresh will be set to half of the current value of the congestion window, which is 8 MSS, and congestion window will be set to the new threshold value + 3 MSS. Thus the new values of the threshold and window size will be 4 MSS and 7 MSS respectively.
10. Threshold is 21 MSS, and congestion window size is 4 MSS.
11. Round 17, 1 packet; round 18, 2 packets; round 19, 4 packets; round 20, 8 packets; round 21, 16 packets; round 22, 21 packets. So, the total number is 52. Note: In round 21, the sender will receive 16 ACKs, out of which the first 5 ACKs will increase cwnd to 21 MSS (that equals ssthresh) and move TCP to congestion avoidance state, and the remaining 11 ACKs will only increase cwnd by less than 1 MSS. Thus, round 22 only sends 21 packets.

### Problem 10 (optional)

1. It takes W/(2MSS) rounds to grow from W/2 to W. Note: It is also correct to say W/(2MSS)+1 rounds if including the first round with window size W/2.
2. The number of packets sent in a period is: 









So the total number of bytes transmitted in a period is: 

1. One segment is lost per period, so the loss rate is:
2. For  large,

.

Thus  or .

From the text, we therefore have

average throughput 

