Homework 3

Due: 11:59 PM on Thursday, March 11, 2021.

Please answer in your own words. Show your work.

1. (15 points) Both TCP and UDP use 1’s complement for their checksum values:
   1. Suppose that you have the following 8-bit bytes: 01010101, 01110000, and 01001100. Calculate the 1’s complement of the sum of all of these 8-bit bytes (that is, add them all together and then determine the 1’s complement from that sum), being sure to handle the overflow as described in the text in a circular fashion. Note that TCP and UDP use 16-bit words in computing the checksum, but only 8-bits are used here for simplicity.

Answer:

Lets first calculate the sum of two 8 bytes, then we will add the result to the third to get the final result:

01010101

+ 01110000

11000101

Lets add this to third byte:

11000101

+ 01001100

100010001

Lets add the carry on to 8 bit to resultant sum

00010001

+ 1 Wraparound add 9th bit

00010010 This is final resultant sum

|  |
| --- |
| **11101101 One’s complement** |

* 1. Is it possible that a 1-bit error will go undetected by the checksum? If yes, explicitly state how this can happen.

Answer: No 1-bit error will not go undetected; the receiver adds all the 8 bits bytes segment, including checksum(one’s complement). When it adds all the 4 bytes, if the resultant value contains 0 in any bit, then receiving side(receiver) knows there is an error in the segment.

Let's demonstrate with an example

So suppose from above part (a) one bit in the first 8 bytes segment is changed when received, that is receiver received 01010001 rest all received as it is. At receiver at it will add all the 4 bytes segment:

01010001

+ 01110000

11000001

Lets add this to third byte:

11000001

+ 01001100

00001101

+ 1(carry)

00001110

Finally, add this with complement with have received

00001110

+ 11101101

11111011

|  |
| --- |
| **Clearly, we can see here that there is one bit which is zero in the resultant sum, which concludes there is a one-bit error. This implies one-bit error won’t go undetected.** |

* 1. Is it possible that a 2-bit error will go undetected by the checksum? If yes, explicitly state how this can happen.

Answer:

Yes, with a 2-bit error will go undetected. Let's explain it with an example

So suppose from above part (a) one of the bits in first and second 8 bytes segment changed i:e it receives 01010001 and 01110100 but, rest all received as it is at the receiver end. At receiver at it will add all the 4 bytes segment including checksum:

01010001

+ 01110100

11000101

Lets add this to third byte:

11000101

+ 01001100

00010001

+ 1(carry)

00001010

Finally, add this with complement with have received

00001010

+ 11101101

11111111

|  |
| --- |
| **Clearly, we can see here that all the bits of the final results are 1. If the receiver receives any zero in the final resultant sum (adding all the received bytes with checksum), only then receiver detects there is an error. Hence the 2-bit error goes undetected.** |

1. (12 points) Consider the scenario where we have implemented a reliable data transfer protocol that uses only negative acknowledgements (NAKs).
   1. Suppose that you send data very infrequently. Would our NAK-only protocol be preferable over an ACK-only protocol? Justify your answer.

Answer: The feedback of the message received or not is sent to sender from receiver in the form of positive (ACK) and negative (NAK) acknowledgments.

As mentioned in the question, since the sender's data transmission is infrequent, NAK -only protocol is not preferable over ACK-only protocol. Following are the reasons for same:

* + NAK-only packet lost is only identified when the next packet is received.
  + In our case, the transmission is infrequent; the receiver has to wait for too long for the next packet to check if the packet is received in sequence or not to detect packet loss and send a response accordingly.
  + Once the receiver identifies the packet lost, it will send NAK to the sender, and then the sender needs to retransmit the lost and next packet both, which takes a long time because transmission is not frequent.
  + The NAK-only protocol is preferred if the sender sends the data frequently. The protocol that uses ACKs is not preferred as it requires sending a greater number of acknowledgements.
  1. Now suppose that we now have a lot of data to send and our end-to-end connection experiences very few losses. Would our NAK-only protocol be preferable over an ACK-only protocol in this case? Justify your answer.

Answer: Yes, in this case, the NAK-only protocol is preferable over ACK-only:

* Since the data transmission here is not infrequent, the packet transmitted from sender to receiver with no long delay, so as soon as receive identify the packet received is out of sequence, it will send NAK to sender.
* Another advantage, in this case, is very few loss experience as mentioned in the question, so the receiver needs to send very few NAK to the sender.

1. (12 points) Suppose we are sending a large file from host A to host B over a TCP connection with a transmission rate of 1 Gbps where the size of a packet is 1,500 bytes, including both header fields and data. The speed-of-light round-trip propagation delay between these two end systems, RTT, is approximately 20 milliseconds. Assume, for simplicity, that ACK packets are extremely small and can be ignored.
   1. How big would the window size have to be (in packets) for the channel utilization to be greater than 80%?

Answer: Transmission rate = 1 Gbps

packet size L = 1500bytes

RTT = 20 ms

The time required for transmitting the packet over a 1Gbps link

=

=

= 0.000012 sec = 0.012ms

So total time = 20 + 0.012 = 20.012 ms

For finding window size having utilization > 80%

0.80

0.80

window size

window size > 1334 packets

|  |
| --- |
| **The window size should be greater than 1334 packets for utilization greater than 80%** |

* 1. Assuming that we have an infinite initial threshold with no losses nor competing traffic, approximately how long in seconds would it take for a normal slow start mechanism to achieve this 80% utilization?

Answer: Number of RTT for 1334 packets = [ = 11

The time required to transmit over 1Gbps link (RTT) = 20.012

11

|  |
| --- |
| **sec will be required to normal flow to start** |

1. (12 points) Suppose that we want to transfer a very large file with size of *L* bytes from Host A to Host B and that we have a maximum segment size (MSS) of 1,460 bytes.
   1. What is the largest size of *L* that is supported so that the TCP sequence numbers are not exhausted (i.e., to avoid duplicate data being accepted as new), recalling that the TCP sequence number field has 4 bytes?

Answer:

File Size = L bytes

MSS = 1460 bytes

The size of TCP sequence number field = 4 bytes = 4\* 8 bits = 32 bits

Number of possible sequence is 232  = 4,294,967,296

Sequence number increase with the number of bytes of data sent. It does not increment by 1. Therefore MSS size is irrelevant, and maximum file size does not depend on MSS.

|  |
| --- |
| **Host A and host B can share file having a maximum size 232** |

* 1. Assume that 26 bytes of transport header, 20 bytes of network header, and 20 bytes of data link header are added to each segment before each resulting packet is sent over a 10 Mbps link. Ignoring flow and congestion control so that Host A can send segments back continuously, use the value of *L* computed in the (a) to compute how long it takes to transmit the file.

Answer:

Header = 26 byte transport + 20 byte network +20 byte data-link

= 66 bytes

Data size = 4.29 Gbytes

MSS = 1460 bytes

Number of segment = Data size / MSS

= 4.29Gbytes/ 1460 bytes = 2,941,758 bytes

After adding the header to each segment, the total number of bytes to be transferred 1460 +66 = 1526 bytes \* Number of segments

1526 \* 2941758 bytes = 4,489,122,708 bytes

To transfer 4,489,122,708 bytes file over 10 Mbps link time required = 4,489,122,708bytes / 10 Mbps = 3591.2 seconds

|  |
| --- |
| **So the time required to transmit the file is 3591 seconds** |

1. (24 points) Consider the method to estimate round trip time (RTT) for TCP using and .
   1. Using initial values of ms and ms (taken before any of the values were obtained), compute the , , and values after values ms, ms, and ms are taken.

|  |  |  |  |
| --- | --- | --- | --- |
| ***SampleRTT(ms)*** | ***EstimatedRTT(ms)*** | ***DevRTT(ms)*** | ***TimeOutInterval*** |
|  | 20 | 30 | 140 |
| 10 | 18.75 | 24.69 | 117.51 |
| 25 | 19.53 | 19.88 | 99.05 |
| 30 | 20.84 | 17.2 | 89.64 |

Lets Calculate:

**For First Time:**

TimeoutInterval = EstimatedRTT + 4\*DevRTT

= 20 + 4\* 30

= 140

**For Second Time:**

EstimatedRTT= (1-)\*EstimatedRTT+ \*SampleRTT

= (1-0.125) \*20 + 0.125 \*10

= 17.5 + 1.25

DevRTT = (1-)\*DevRTT + \*|SampleRTT-EstimatedRTT|

= (1-0.25) \*30 + 0.25\* |10-18.75|

= 22.5 + 2.1875

= 24.69

TimeoutInterval = EstimatedRTT + 4\*DevRTT

= 18.75 + 4\* 24.69

= 117.51

**For third Time:**

EstimatedRTT= (1-)\*EstimatedRTT+ \*SampleRTT

= (1-0.125) \*18.75 + 0.125 \*25

= 16.40625 + 3.125

= 19.53

DevRTT = (1-)\*DevRTT + \*|SampleRTT-EstimatedRTT|

= (1-0.25) \*24.69 + 0.25\* |25-19.53|

= 18.5175+ 1.3675

= 19.88

TimeoutInterval = EstimatedRTT + 4\*DevRTT

= 19.53 + 4\* 19.88

= 99.05

**For fourth Time:**

EstimatedRTT= (1-)\*EstimatedRTT+ \*SampleRTT

= (1-0.125) \*19.53 + 0.125 \*30

= 17.08875 + 3.75

= 20.84

DevRTT = (1-)\*DevRTT + \*|SampleRTT-EstimatedRTT|

= (1-0.25) \*19.88 + 0.25\* |30-20.84|

= 14.91+ 2.29

= 17.2

TimeoutInterval = EstimatedRTT + 4\*DevRTT

= 20.84 + 4\* 17.2

* 1. Based on the results from (a), derive the formula for for *n* sample RTTs using induction with .

Answer:

Let's say *EstimatedRTT*n is estimated after n sample RTT.

Since no Samples were taken, let us suppose *SampleRTT* and *EstimatedRTT* are the same. Therefore:

When n=1

=

Now,

When n= 2

=

When n = 3

=

=

=

Now, for the nth iteration:

=

|  |
| --- |
| **= 0.125** |

* 1. Based on your formula for *n* sample RTTs derived in (a) above, what happens when ? Write the resulting formula for .

Answer:

If n

=

=

1. (15 points) Consider that Host A and Host B are communicating over TCP. Host B has already received from Host A all bytes through 146. Now, suppose that Host A sends Host B two segments back-to-back, where the first and second segments contain 80 and 40 bytes of data respectively. In the first segment, the sequence number is 147, with a source port number of 282 and destination port number of 80. Host B sends an acknowledgement whenever it receives a segment from Host A.
   1. Identify the sequence number, source port number, and destination port number in the second segment sent from Host A to Host B.

Answer:

Sequence number for second segment = 147 + 80 = 227

Source port number would be =282

Destination port number = 80

* 1. Suppose that the first segment arrives at Host B before the second segment. Identify the ACK number, source port number, and destination port number in the acknowledgement of the first arriving segment.

Answer:

ACK number for first arriving segment = 227

Source port number is = 80

Destination port number is = 282

* 1. Suppose that the second segment arrives at Host B before the first segment. Identify the ACK number, source port number, and destination port number in the acknowledgement of the first arriving segment.

Answer:

In this case, if the second segment reaches host B before the first segment arrival, the ACK number would be 147, which means Host B will wait for the 147 bits segment.

Source port number = 80

Destination port number = 282

1. (5 points) The designers of TCP chose to have TCP wait until it has received three duplicate ACKs before performing a fast retransmit. Why do you think that they chose not to perform a fast retransmit after the first duplicate ACK for a segment is received?

Answer:

TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments; it waits for a small number of duplicate ACKs to be received. It is assumed that if there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed, which will then generate a new ACK. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. TCP then performs a retransmission of what appears to be the missing segment without waiting for a retransmission timer to expire.[2]

Now suppose we sender sent three packets k-1, k, and k+1, and now packet k-1 is received and acknowledged, now as explained above, if packet k and k+1 reordered along the path and k+1 is received before k, then the receiver will generate duplicate ack for packet k-1, and sender sent them immediately instead of waiting for two more duplicate packet (while packet k is along the way) increase the packets number in the buffer (which Is not lost but reordered) this increased overhead for transmission, hence to avoid this overhead caused due to reordering designer has chosen to wait for three duplicate acknowledgment.

1. (5 points) Of the two protocols discussed, Go-Back-N or Selective-Repeat, which one makes more efficient use of network bandwidth? Justify your answer.

Answer:

As we know, Go-Back-N and selective repeat are both overcome the channel utilization problem of stop and wait protocol. They both fill the pipeline for trying to utilize the bandwidth at their maximum.

Although both have come up with a solution for stop and wait network bandwidth waste, the most effective between them is a **selective repeat**. Since Selective Repeat protocol keeps the out-of-sequence packets in its buffer at receiver end waiting for packet arrival, whereas this has not been followed by Go-Back-N protocol, “here single-packet error can cause GBN to retransmit a large number of packets, many unnecessarily. As the probability of channel errors increases, the pipeline can become filled with these unnecessary retransmissions”.[1]

References:

[1]: Computer Networking: A Top-Down Approach featuring Internet 7th edition, Kurose and Ross, Addison Wesley

[2] : stack overflow