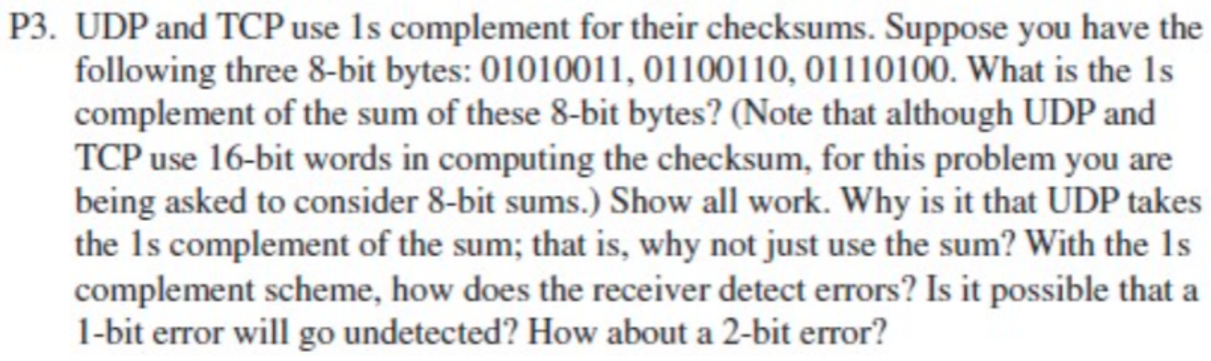
Julian Garcia
CSE 353: Hw3


P3. UDP and TCP use 1s complement for their checksums. Suppose you have the following three 8-bit bytes: 01010011, 01100110, 01110100. What is the 1s complement of the sum of these 8-bit bytes? (Note that although UDP and TCP use 16-bit words in computing the checksum, for this problem you are being asked to consider 8-bit sums.) Show all work. Why is it that UDP takes the 1s complement of the sum; that is, why not just use the sum? With the 1s complement scheme, how does the receiver detect errors? Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

Adding first two 8-bit bytes:

```
       01010011
   +   01100110
   =   10111001
```

Adding the 3rd byte to the result

```
       10111001
   +   01110100
   =   100101101
```

Wrapping around the extra bit.

```
 00101101
        1
  00101110
```

The sum of the three bytes is 00101110.

The 1's compliment of 00101110 is 11010001.
To get the check sum, we invert the bits, so the check sum is also 11010001

UDP uses the 1's complement of the sum because it turns out to be the same as the checksum.

The receiver detects errors by adding all frames of 8 bits with the checksum, then finding the one's compliment of the sum of data and checksum, then if the result contains any zeros, the receiver knows an error has occurred, if there are no zeros, then no error has occurred.

It isn't possible for a 1-bit error to go undetected using 1's complement, because if a single bit is changed from 0 to 1, then the sum changes from 1 to zero which results in an error.

It is, however, possible for a 2-bit error to go undetected using 1's complement, because if two bits are changed at once sometimes the sum can remain the same.

P7. In protocol `rdt3.0`, the ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence number of the packet they are acknowledging). Why is it that our ACK packets do not require sequence numbers?

The easiest way to answer this question is to look at what sequence numbers are used for. The sender needs sequence numbers so that the receiver can tell if a packet is a duplicate of an already received packet. With ACKs, the sender doesn't need that sequence number to detect a duplicate ACK. A duplicate ACK is just obvious to the rdt3.0 receiver, because when it received the original ACK it transitioned to the next state. The duplicate ACK is not the ACK that the sender needs and therefore is just ignored by the rdt3.0 sender, which is why it doesn't need a sequence number.

P12

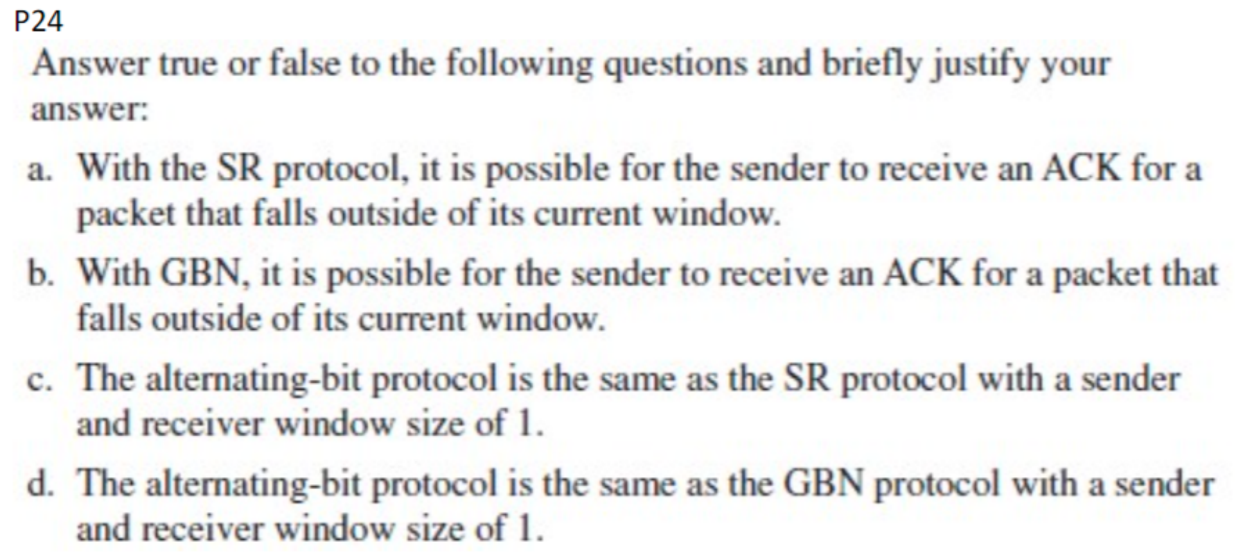
The sender side of `rdt3.0` simply ignores (that is, takes no action on) all received packets that are either in error or have the wrong value in the `acknum` field of an acknowledgment packet. Suppose that in such circumstances, `rdt3.0` were simply to retransmit the current data packet. Would the protocol still work? (*Hint*: Consider what would happen if there were only bit errors; there are no packet losses but premature timeouts can occur. Consider how many times the *n*th packet is sent, in the limit as *n* approaches infinity.)

This probably wouldn't work efficiently.
rtd3.0 transfers data to a receiver from a sender, as soon as the receiver receives the packet, an ACK is given as response to the sender so that sender can confirm the receiver has gotten it.
The receiver will not send any acknowledgement if the packet contains an error.
After that timeout, the packet will be re-transmitted by the sender.
That's where the inefficiency arises, in the case of a packet being sent many times, since other packets will have to wait to be sent until the current packet is sent successfully.

P14

Consider a reliable data transfer protocol that uses only negative acknowledgments. Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why? Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

If the sender sends the data infrequently, then the NAK-only protocol wouldn’t really be preferred. It’d be better to use the protocol that uses ACKs. This is because the biggest disadvantage of a NAK-only protocol is that when the packet has been lost it can only detect that it has been lost when the next packet is received by the receiver. The NAK-only protocol would realize the loss of the packet after far too much time had passed in this case, since the sender in this case sends packets occasionally. Then, when the receiver would realize the packet loss, it would send a NAK to to the sender and the sender would then have to retransmit both the lost packet and the next packet. However, if the sender were sending the data frequently with very few losses, the NAK-only protocol would be preferred since the ACK protocol would have to send a far greater number of acknowledgements than the NAK-only protocol would have to send.



P24

Answer true or false to the following questions and briefly justify your answer:

a. With the SR protocol, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.

b. With GBN, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.

c. The alternating-bit protocol is the same as the SR protocol with a sender and receiver window size of 1.

d. The alternating-bit protocol is the same as the GBN protocol with a sender and receiver window size of 1.

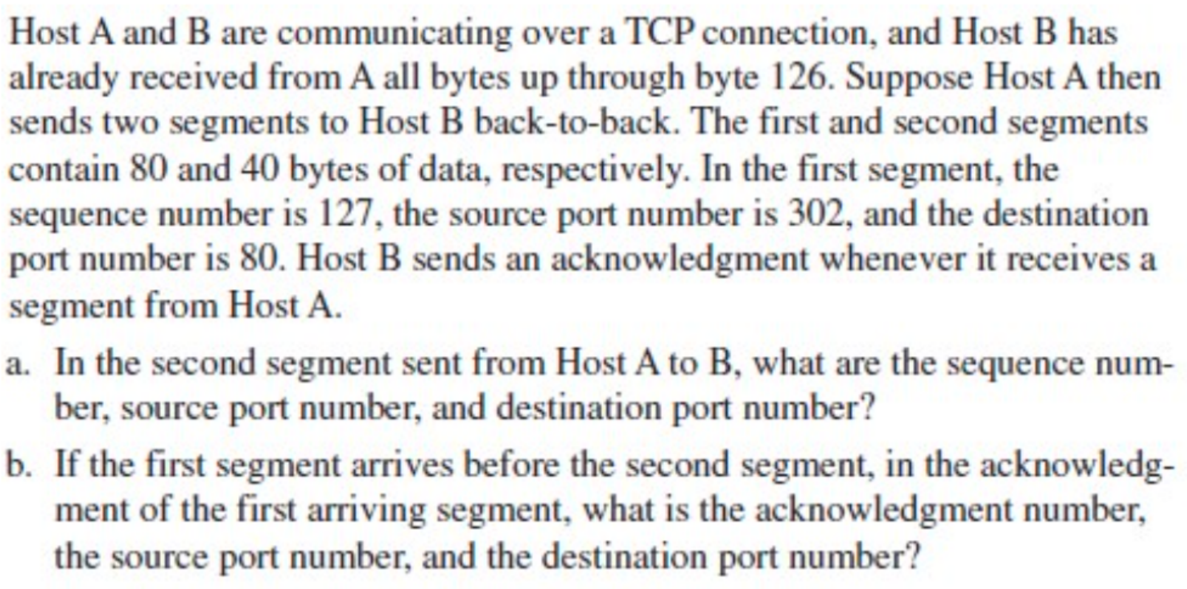
A) True - When you send some packages to receiver, the receiver receives the packages and sends the acknowledgement packages but the sender timeout occurs before receiving the packages and retransmits all the packets and now those delayed acknowledgements are received and the sender transmits the next window of packets but receives the acknowledgements for the packets it retransmitted for the first window, and hence the sender receives an acknowledgement which isn’t for the current window.

B) True - If we suppose all the ACKs are received after the timeout on the sender side, we receive ACKs for retransmitted packages in the next window of packages.

C) True - for a window size of 1, alternating bit protocol and SR protocols behave similarly since out of order packages do not occur.

D) True -They behave similarly since cumulative and individual acknowledgements are the same for the sender and receiver with a window size of 1.

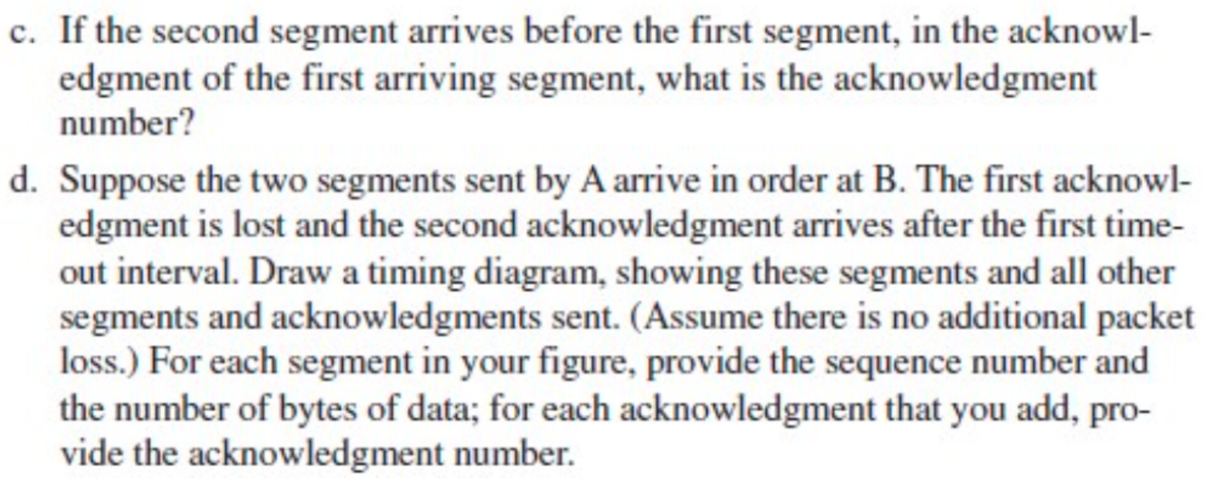
P27



Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 126. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 80 and 40 bytes of data, respectively. In the first segment, the sequence number is 127, the source port number is 302, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.

a. In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?

b. If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number?



c. If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number?

d. Suppose the two segments sent by A arrive in order at B. The first acknowledgment is lost and the second acknowledgment arrives after the first timeout interval. Draw a timing diagram, showing these segments and all other segments and acknowledgments sent. (Assume there is no additional packet loss.) For each segment in your figure, provide the sequence number and the number of bytes of data; for each acknowledgment that you add, provide the acknowledgment number.

a)

Sequence number = first segment of sequence number + destination port number

=127+80 =207

Sequence number=207

Source port number = 302

Destination port number= 80

b)

Acknowledgement number= 207

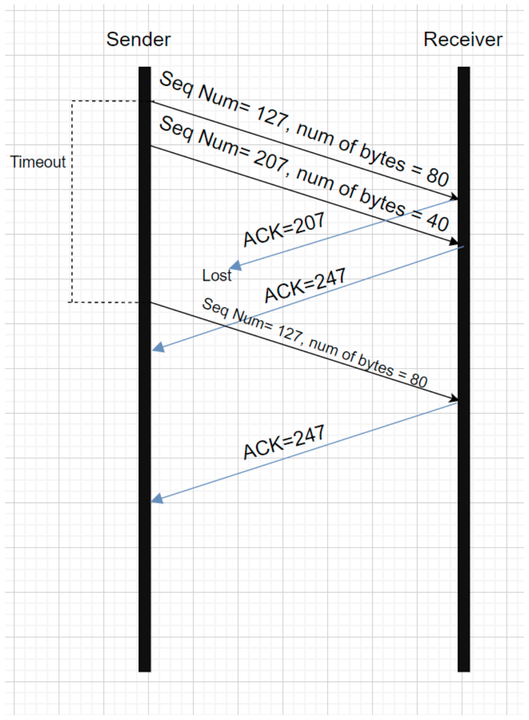
Source port number = 80

Destination port number= 302

c)

Acknowledgement number=127

If the second segment arrives before the first segment, in the acknowledgement of the first arriving segment, the acknowledgement number will be 127, indicating that it is still waiting for bytes 127 and onwards.

d. 
Sender
Receiver
Seq Num= 127, num of bytes = 80
Seq Num= 207, num of bytes = 40
Timeout
ACK=207
Lost
ACK=247
Seq Num= 127, num of bytes = 80
ACK=247

P44

Consider sending a large file from a host to another over a TCP connection that has no loss.

a. Suppose TCP uses AIMD for its congestion control without slow start. Assuming cwnd increases by 1 MSS every time a batch of ACKs is received and assuming approximately constant round-trip times, how long does it take for cwnd increase from 1 MSS to 6 MSS (assuming no loss events)?

b. What is the average throughout (in terms of MSS and RTT) for this connection up through time = 5 RTT?

a) It takes 1 RTTs to get it to 2 MSS, 2 RTTs to get it to 3 MSS, 3 RTTs to get it to 4 MSS, 4 RTTs to get it to 5 MSS, and 5 RTTs to get it to 6MSS
b) Average throughput = (1 + 2 + 3 + 4 + 5) / 6 = 2.5 MSS/RTT