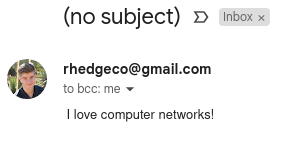
Group Rust

CMPE 148 HW3

LAB GITHUB LINK: <https://github.com/CMPE-148-Sec3-FALL2022/homework3>

SCREENSHOT OF RECEIVED EMAIL FROM SMTP PYTHON CODE:



P1 - P4

P1

*Suppose Client A initiates a Telnet session with Server S. At about the same time, Client B also initiates a Telnet session with Server S. Provide possible source and destination port numbers for a. The segments sent from A to S. b. The segments sent from B to S. c. The segments sent from S to A. d. The segments sent from S to B. e. If A and B are different hosts, is it possible that the source port number in the segments from A to S is the same as that from B to S? f. How about if they are the same host*

1. Possible port numbers of segment from A - S

Source: 1030, Destination: 23

1. Possible port numbers of segment from B - S

Source: 1040, Destination: 23

1. Possible port numbers of segment from S - A

Source: 23, Destination: 1030

1. Possible port numbers of segment from S - B

Source: 23, Destination: 1040

1. **It is possible** for the source port number to be the same between A - S and B - S if A and B are different hosts
2. **It is not possible** for the source port number to be the same if A and B are the same host.

P2

*Consider Figure 3.5 . What are the source and destination port values in the segments flowing from the server back to the clients’ processes? What are the IP addresses in the network-layer datagrams carrying the transport-layer segments?*

1. Source and destination port values in segments flowing from server back to client:

Server B to web client host C (left port): **Source: 80, Destination: 7532**

Server B to web client host C (right port): **Source: 80, Destination: 26145**

Server B to web client host A: **Source: 80, Destination: 26145**

1. IP addresses in datagrams carrying transport-layer segments

Server B to web client host C (left port): **Source: B, Destination: C**

Server B to web client host C (right port): **Source: B, Destination: C**

Server B to web client host A: **Source: B, Destination: A**

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?*

1. 1s complement of the sum of 01010011, 01100110, and 01110100

First, add 01010011 and 01100110:

01010011

* 01100110

= 10111001

Second, add the sum to the third 8-bit byte, 01110100

10111001

* 01110100

= 00101101 (exclude left overflowed bit)

Third, take sum and invert to get **one’s complement**

00101101

inverted = **11010010**

1. Why does UDP take the 1s complement rather than just the sum?

By having the sender compute and include the checksum as a 1s complement, the receiver is able to more quickly check for an error by adding the sum of the bytes and the checksum and checking if the result contains no zeros (all ones).

If UDP takes just the sum instead of the sum’s complement, it would result in longer computations for the receiver. This is because it would require a comparison operation, checking each bit of the calculated sum and the provided sum one-by-one. With a 1’s complement checksum however, the receiver needs to just add one more byte (the checksum) and perform a faster operation to see if any of the bits are 0.

1. How does the receiver detect errors with the one’s complement scheme?

The receiver calculates the sum of the bytes and then adds the checksum. If the result contains anything other than all 1’s (meaning 1 or more 0s), then there must be an error.

1. A 1-bit error **will be detected**
2. It is possible for a 2-bit error to go **undetected**

P4

*Suppose you have the following 2 bytes: 01011100 and 01100101. What is the 1s complement of the sum of these 2 bytes? b. Suppose you have the following 2 bytes: 11011010 and 01100101. What is the 1s complement of the sum of these 2 bytes? c. For the bytes in part (a), give an example where one bit is flipped in each of the 2 bytes and yet the 1s complement doesn’t change.*

1. Sum: 10000001 -> 1’s complement of sum: **01111110**
2. Sum: 00111111 -> 1’s complement of sum: **11000000**
3. If both bytes have their leftmost bit flipped from 0 to 1 (**11011100 and 11100101**), the sum and the complement will be the same. In this case, the error will go undetected

P51 - P56

P51

*Consider the network described in the previous problem. Now suppose that the two TCP connections, C1 and C2, have the same RTT of 100 msec. Suppose that at time t , C1’s congestion window size is 15 segments but C2’s congestion window size is 10 segments. a. What are their congestion window sizes after 2200 msec? b. In the long run, will these two connections get about the same share of the bandwidth of the congested link? c. We say that two connections are synchronized, if both connections reach their maximum window sizes at the same time and reach their minimum window sizes at the same time. In the long run, will these two connections get synchronized eventually? If so, what are their maximum window sizes? d. Will this synchronization help to improve the utilization of the shared link? Why? Sketch some idea to break this synchronization.*

1. Congestion windows of C1 and C2 after 2200 ms

**C1: 2 segments, C2: 2 segments**

Reasoning:

At 100 ms, after the first RTT for C1 and C2, the combined data-sending rate is 250 segments/second (15 / 0.1 + 10 / 0.1). So, the window size for both C1 and C2 should be decreased by half. C1’s window becomes 7 (15 // 2) segments and C2’s window becomes 5 (10 // 2) segments.

At 200 ms, combined data-sending rate is 120, so window sizes become 3 and 2.

At 300 ms, combined data-sending rate is 50, so window sizes become 1 and 1.

At 400 ms, combined data-sending rate is 20, so windows sizes become 2 and 2.

At 500 ms, combined data-sending rate is 40, so windows sizes become 1 and 1.

At 600 ms, combined data sending rate is 20, so window sizes become 2 and 2.

From the last 400 ms, it can be observed that the rates of C1 and C2 have synchronized and fluctuate between 20 segments/s and 40 segments/s. The window sizes after 2200 ms will be 2 segments and 2 segments for C1 and C2.

1. In the long run, these two connections **will get the same share of bandwidth** as they quickly synchronize, with no way for the synchronization to break with the current algorithm.
2. These two **do synchronize**, reaching the minimum window size at 300 ms and the maximum window size at 400 ms. The max window size is **2 segments**.
3. Synchronization **does not help** improve the utilization of the shared link. The shared link is being underutilized around 50% of the time, with an actual rate of just 20 segments/second every other RTT. One way to break this synchronization is to intentionally not change the window size after random RTT (for example, C1 window size does not increase or decrease at 2300 ms. This can break the synchronization and allow the rates potentially allow the shared link to be utilized better).

P52

*Consider a modification to TCP’s congestion control algorithm. Instead of additive increase, we can use multiplicative increase. A TCP sender increases its window size by a small positive constant ‘a’ ( 0 < a < 1) whenever it receives a valid ACK. Find the functional relationship between loss rate L and maximum congestion window W. Argue that for this modified TCP, regardless of TCP’s average throughput, a TCP connection always spends the same amount of time to increase its congestion window size from W/2 to W*

S = (W/2) + (W/2) \* (1 + a) + (W/2) \* (1 + a)2 + (W/2) \* (1 + a)3 + … + (W/2) \* (1 + a)k

Where, k = log(1 + a) 2

S = W \* ((2a + 1) / (2a))

Loss Rate is inverse of total segments: L = (2a) / (W \* (2a + 1))

The time that it will take to increase the window size from W/2 to W is

(log(1 + a) 2) \* RTT

Which is independent of the average throughput of the TCP connection.

Average throughput = MSS \* S / ((k + 1) \* RTT)

= MSS / (L \* (k + 1) \* RTT)

In this case, the TCP spends an identical amount of time to increase the congestion window.

P53

*In our discussion of TCP futures in Section 3.7 , we noted that to achieve a throughput of 10 Gbps, TCP could only tolerate a segment loss probability of (or equivalently, one loss event for every 5,000,000,000 segments). Show the derivation for the values of (1 out of 5,000,000) for the RTT and MSS values given in Section 3.7 . If TCP needed to support a 100 Gbps connection, what would the tolerable loss be?*

1. Show the derivation for the 1 / 5,000,000,000 loss of segment probability:

TCP throughput (T) = (1.22 \* MSS) / (RTT \* sqrt(L))

RTT \* sqrt(L) = (1.22 \* MSS) / T

RTT2 \* L = ((1.22 \* MSS) / T)2

L = ((1.22 \* MSS) / T)2 / RTT2

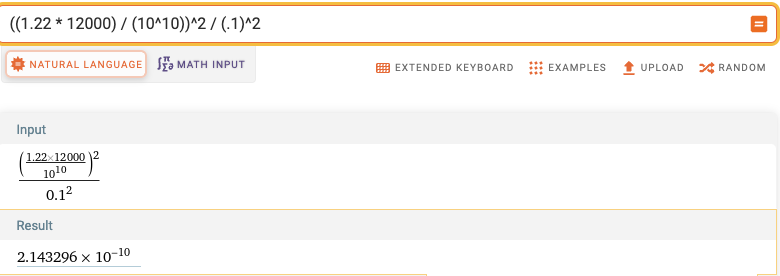
We are given that:

MSS = 1500 bytes -> 12000 bits

RTT = 100 ms -> 0.1 s

T = 10 Gbps -> 1 \* 10^ 10

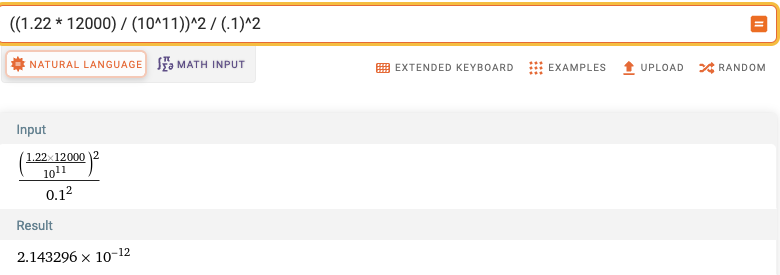
Plugging these values in to solve for L, we get:



*Calculation shown on Wolfram Alpha*

**2 \* 10^-10 (rounded) equates to 1 / 5,000,000,000**

1. If TCP needed to support a 100 Gbps connection, the tolerable loss can be found by plugging in 1 \* 1011 for T (our TCP throughput).



With L = 2 \* 10-12, **this means a loss tolerance of 1 / 500,000,000,000 segments lost**

P54

*In our discussion of TCP congestion control in Section 3.7 , we implicitly assumed that the TCP sender always had data to send. Consider now the case that the TCP sender sends a large amount of data and then goes idle (since it has no more data to send) at t . TCP remains idle for a relatively long period of time and then wants to send more data at t . What are the advantages and disadvantages of having TCP use the cwnd and ssthresh values from t when starting to send data at t ? What alternative would you recommend? Why?*

1. Advantages and disadvantages of reusing cwnd and ssthresh values from t1

Advantage: Starting with the same values from t1 can save many RTTs getting to the cwmd that is most optimal for the TCP link. Given that it had sent a large amount of data, the cwnd has likely approached an effective value.

Disadvantage: Given that the TCP remains idle for a relatively long period of time, there is a good chance that the TCP’s congestion is either much greater or much smaller than at time t1. For example, if our cwnd was very small because the TCP was very congested, but is no longer congested at t2, then the link is going to be underutilized for many RTTs. This would not be the case if it started at the default value.

1. Alternative I would recommend:

Assuming we are using an additive increase and multiplicative decrease algorithm, I would propose to choose the maximum between cwnd and ssthresh at t1 and the default values. This is because it will take many more RTTs to increase to the optimal cwnd compared to decreasing to it, so it is better to overshoot rather than undershoot (assuming that there is an equal chance of the TCP link being either less or more congested after a long idle period).

P55

*In this problem we investigate whether either UDP or TCP provides a degree of end-point authentication. a. Consider a server that receives a request within a UDP packet and responds to that request within a UDP packet (for example, as done by a DNS server). If a client with IP address X spoofs its address with address Y, where will the server send its response? b. Suppose a server receives a SYN with IP source address Y, and after responding with a SYNACK, receives an ACK with IP source address Y with the correct acknowledgment number. Assuming the server chooses a random initial sequence number and there is no “man-in-the-middle,” can the server be certain that the client is indeed at Y (and not at some other address X that is spoofing Y)?*

1. The server will send its response to the spoofed address Y
2. No, the client cannot be certain the client is indeed at Y. There is still a very small possibility that the spoofer guesses the random sequence number, and is able to send the ACK successfully.

P56

*In this problem, we consider the delay introduced by the TCP slow-start phase. Consider a client and a Web server directly connected by one link of rate R. Suppose the client wants to retrieve an object whose size is exactly equal to 15 S, where S is the maximum segment size (MSS). Denote the round-trip time between client and server as RTT (assumed to be constant). Ignoring protocol headers, determine the time to retrieve the object (including TCP connection establishment)*

1. 4 S/R>S/R+RTT>2S/R

(Including TCP connection establishment)

2RTT + S/R + RTT + S/R + RTT + 12/R

**4RTT + 14S/R**

1. S/R+RTT>4 S/R

2RTT + S/R + RTT + S/R + RTT + S/R + RTT + 8S/R

**5RTT + 11S/R**

1. S/R>RTT

2RTT + S/R + RTT + 14S/R

**3RTT + 15S/R**