CSCI-UA 480: Computer Networks Assignment 2

Kai Liao

October 16, 2020

1 Moving averages and retransmission timeouts

1.1 Moving averages

The equation for online estimation of mean RTT is

$$\mu \leftarrow (1-\alpha)\cdot\mu + \alpha\cdot RTT \tag{1}$$

Figure 1 shows the mean RTT estimated from EWMA using two different α and our synthetic RTT set. In the $\alpha = 0.5$ case, equation (1) decomposes into simple average of old estimate and current RTT. Therefore, the mean RTT becomes 0.5 when we have RTT $= 1$ for the first time, gradually increases to 1, and remains 1 since all of the subsequent RTTs are 1 in our synthetic set.

In the $\alpha = 0.01$ case, equation (1) becomes a weighted average of old estimate and current RTT, in which we assign significantly more weight (0.99) to the old estimate. It is the reason why our estimate of mean RTT increases much slower.

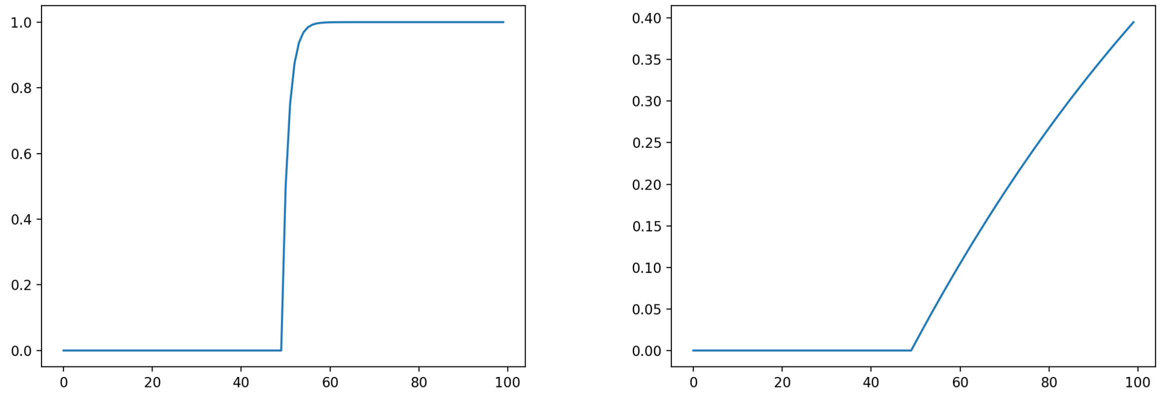


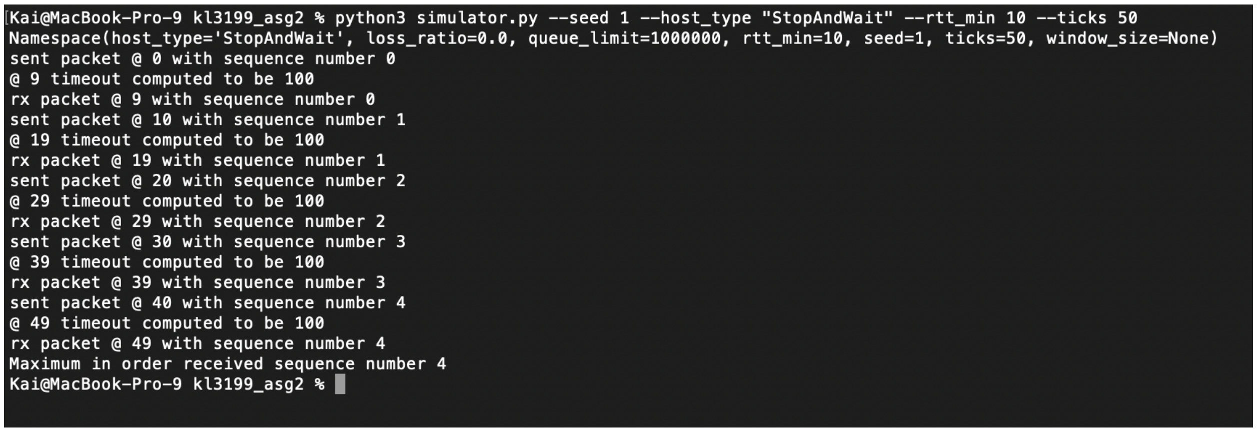
Figure 1: Mean RTT estimation for $\alpha = 0.5$ (left) and $\alpha = 0.01$ (right)

1.2 Retransmission timouts

If we don't have a minimum timeout, it is possible that our timeout estimate may fall below RTT_{min}, which is the minimum time required for a round trip due to propagation and transmission delay. Packets would be incorrectly determined as lost in this case, leading to more retransmissions and lower throughput. Similarly, if we don't have a maximum timeout, our timeout estimate may continue to increase. In this case, we may end up waiting for an ACK in which the packet might have been lost for unreasonably long period of time and thus lower throughput.

2 Moving averages and retransmission timeouts

2.1



```
Kai@MacBook-Pro-9 kl3199_asg2 % python3 simulator.py --seed 1 --host_type "StopAndWait" --rtt_min 10 --ticks 50
Namespace(host_type='StopAndWait', loss_ratio=0.0, queue_limit=1000000, rtt_min=10, seed=1, ticks=50, window_size=None)
sent packet @ 0 with sequence number 0
@ 9 timeout computed to be 100
rx packet @ 9 with sequence number 0
sent packet @ 10 with sequence number 1
@ 19 timeout computed to be 100
rx packet @ 19 with sequence number 1
sent packet @ 20 with sequence number 2
@ 29 timeout computed to be 100
rx packet @ 29 with sequence number 2
sent packet @ 30 with sequence number 3
@ 39 timeout computed to be 100
rx packet @ 39 with sequence number 3
sent packet @ 40 with sequence number 4
@ 49 timeout computed to be 100
rx packet @ 49 with sequence number 4
Maximum in order received sequence number 4
Kai@MacBook-Pro-9 kl3199_asg2 %
```

Figure 2: Terminal output from running simulator.py (same parameters as those in sample output)

2.2

The throughput is close to the function derived in class $\frac{1}{RTT_{min}}$

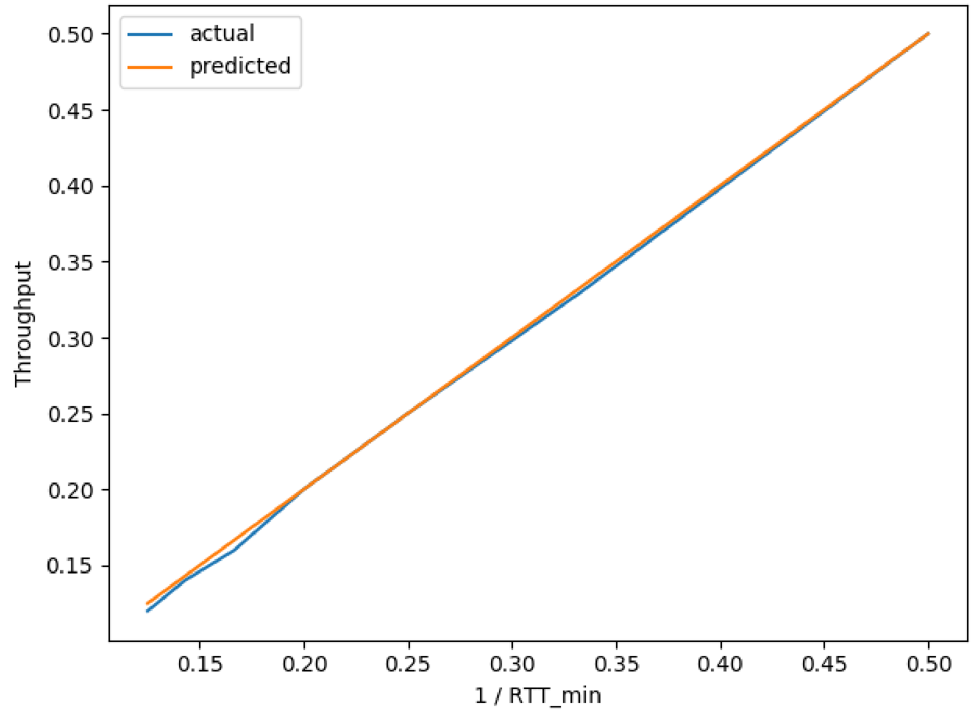
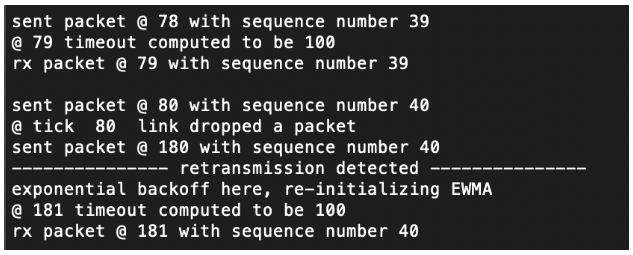


Figure 3: Predicted and actual throughput

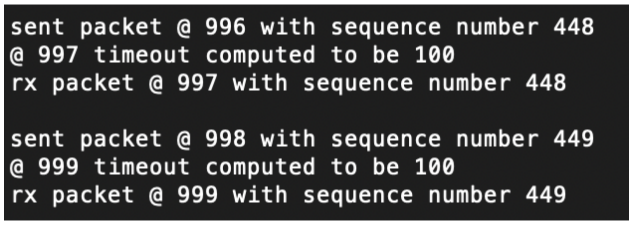
2.3

The difference between simulated and theoretical throughput is relatively large. In the experiment in Figure 4, the throughput is $\frac{450}{1000} = 0.45$, which is bit far from $\frac{1}{RTT_{min}} = 0.5$. The protocol continues to function



```
sent packet @ 78 with sequence number 39
@ 79 timeout computed to be 100
rx packet @ 79 with sequence number 39

sent packet @ 80 with sequence number 40
@ tick  80  link dropped a packet
sent packet @ 180 with sequence number 40
--------------- retransmission detected ---------------
exponential backoff here, re-initializing EWMA
@ 181 timeout computed to be 100
rx packet @ 181 with sequence number 40
```



```
sent packet @ 996 with sequence number 448
@ 997 timeout computed to be 100
rx packet @ 997 with sequence number 448

sent packet @ 998 with sequence number 449
@ 999 timeout computed to be 100
rx packet @ 999 with sequence number 449
```

Figure 4: Terminal output from running simulator.py (seed=1, rtt min=2, ticks=1000, loss ratio=0.01)

correctly in this case. The only packet drop occurs at tick 80 and retransmission is triggered at tick 180 according to the calculated timeout.

The divergence between theoretical and simulated throughput decreases as RTT_{min} increases. The figure bloew shows the experimental results. As RTT_{min} increases, packet loss is less likely to occur, making the simulation resembles the situation without packet loss more.

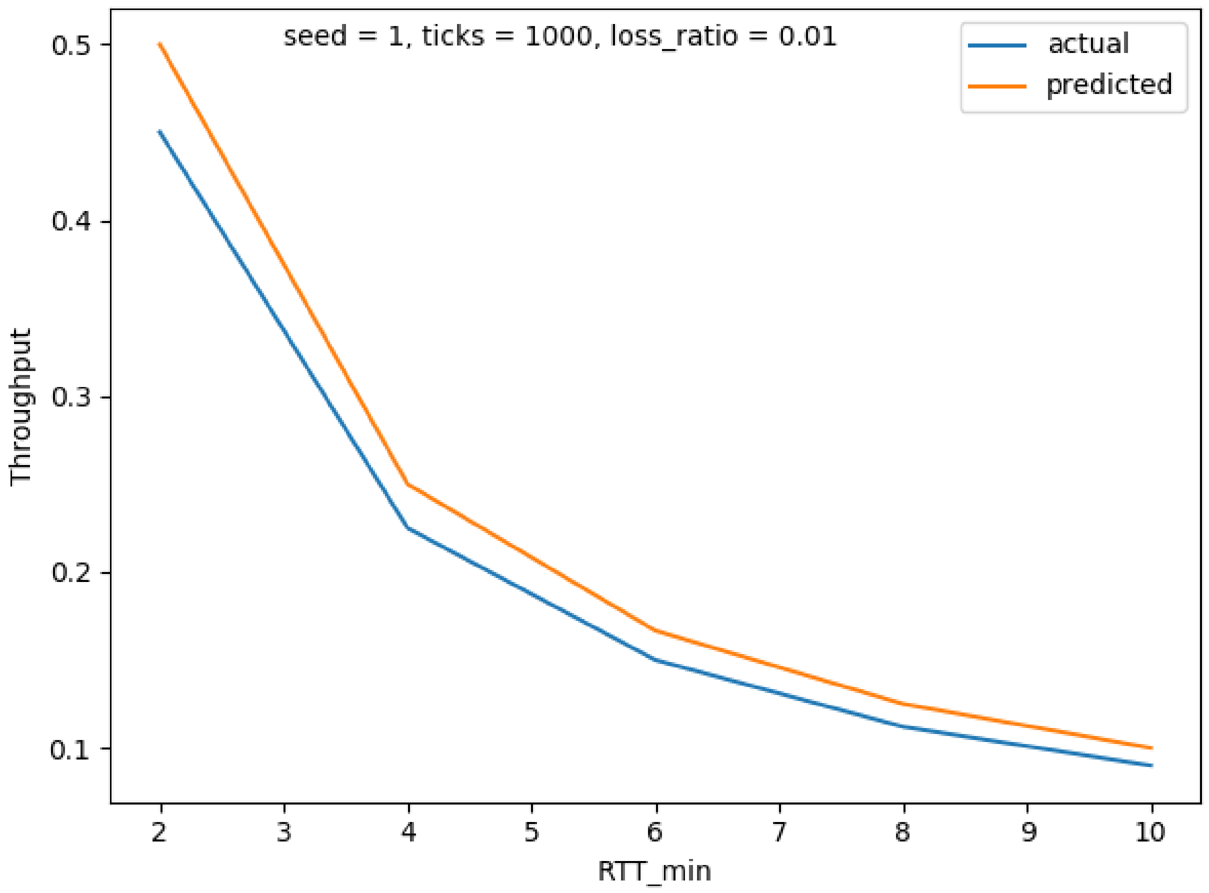
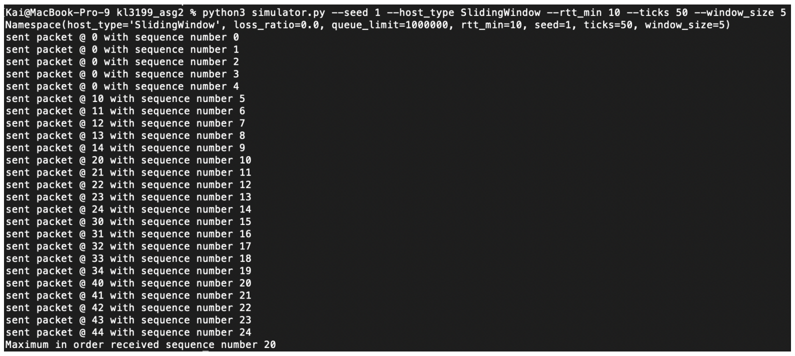


Figure 5: Difference between actual and predicted throughput

3 The sliding window protocol

3.1



```
Kai@MacBook-Pro-9 kl3199_asg2 % python3 simulator.py --seed 1 --host_type SlidingWindow --rtt_min 10 --ticks 50 --window_size 5
Namespace(host_type='SlidingWindow', loss_ratio=0.0, queue_limit=1000000, rtt_min=10, seed=1, ticks=50, window_size=5)
sent packet @ 0 with sequence number 0
sent packet @ 0 with sequence number 1
sent packet @ 0 with sequence number 2
sent packet @ 0 with sequence number 3
sent packet @ 0 with sequence number 4
sent packet @ 10 with sequence number 5
sent packet @ 11 with sequence number 6
sent packet @ 12 with sequence number 7
sent packet @ 13 with sequence number 8
sent packet @ 14 with sequence number 9
sent packet @ 20 with sequence number 10
sent packet @ 21 with sequence number 11
sent packet @ 22 with sequence number 12
sent packet @ 23 with sequence number 13
sent packet @ 24 with sequence number 14
sent packet @ 30 with sequence number 15
sent packet @ 31 with sequence number 16
sent packet @ 32 with sequence number 17
sent packet @ 33 with sequence number 18
sent packet @ 34 with sequence number 19
sent packet @ 40 with sequence number 20
sent packet @ 41 with sequence number 21
sent packet @ 42 with sequence number 22
sent packet @ 43 with sequence number 23
sent packet @ 44 with sequence number 24
Maximum in order received sequence number 20
```

Figure 6: Terminal output from running simulator.py (same parameters as those in sample output)

3.2

Setting: seed=1, ticks=10000. In each plot, one variable is fixed, the other one is varied. All digits are kept. The throughputs we observe are close to the function derived in class [1].

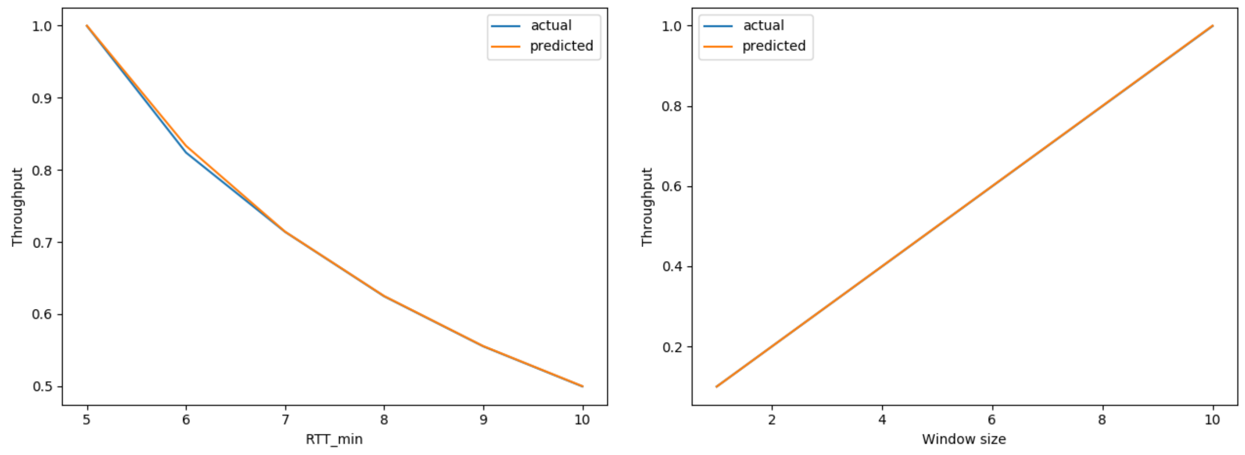


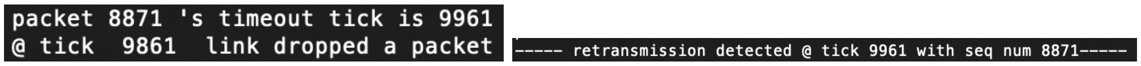
Figure 7: (left) fix window size = 5, (right) fix rtt_min = 10

3.3

Left figure setting: ticks = 10000, rtt_min = 10, loss_ratio = 0.01
Right figure setting: ticks = 10000, window size = 5, loss_ratio = 0.01

It continues to function correctly if we examine the timeout and retransmission time tick. Refer to the setting $RTT_{min} = 10$, $window\_size = 10$, $loss\_ratio = 0.01$, $ticks = 10000$. Breakpoint print functoin lines are added, and we have the following terminal output for the last retransmission detected. Packet 8871 is dropped and retransmitted at the correct time tick.

The difference between predicted and actual throughput is relatively large. Refer to the setting in the above discussion, the difference is around 10 percent. More than 90 packets are retransmitted in this setting, which can be verified running the script. Clearly, those retransmissions contribute to the difference in predicted and actual throughput.



```
packet 8871 's timeout tick is 9961
@ tick  9861  link dropped a packet
```

```
----- retransmission detected @ tick 9961 with seq num 8871-----
```

Figure 8: terminal outline catch points

Similar to the situation in stop and wait ptotocol, the divergence between theoretical and simulated throughput decreases as RTT_{min} increases if we hold window size fixed. As RTT_{min} increases, we have less packet loss, making it resembles the situation without packet loss more.

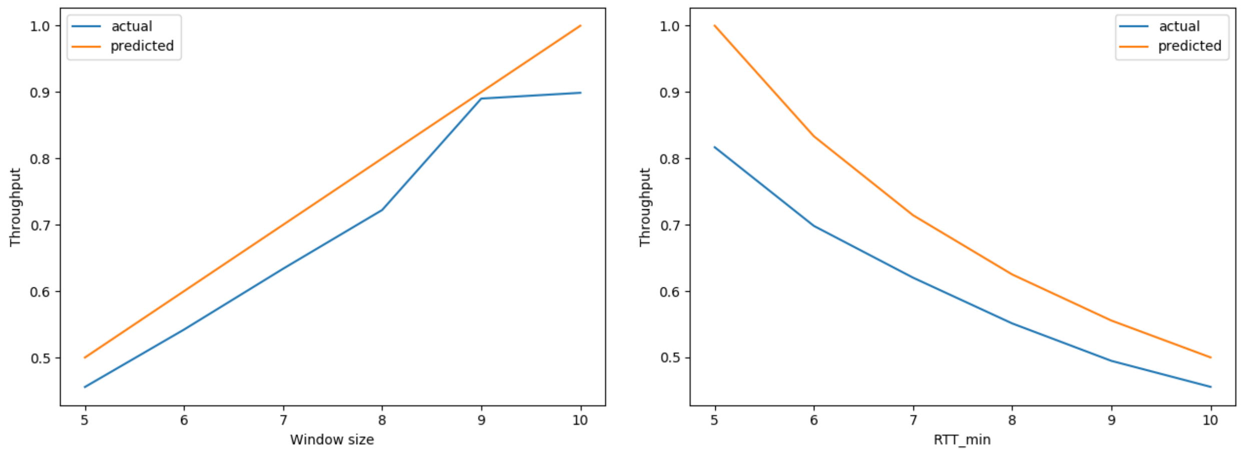


Figure 9: (left) fix rtt_min = 10, (right) fix window size = 5

4 Congestion collapse

Setting: MIN_TIMEOUT=100, MAX_ TIMEOUT=1000, seed=1, rtt_min=10, ticks=100000, queue_limit=10000 The throughput is recorded with a step of size 10. Notice that the throughput does not reach a plateau starting from window size 600. Overall, it is decreasing in that range, which can be verified in the file congestion_collapse.txt that records max in order received sequence number.

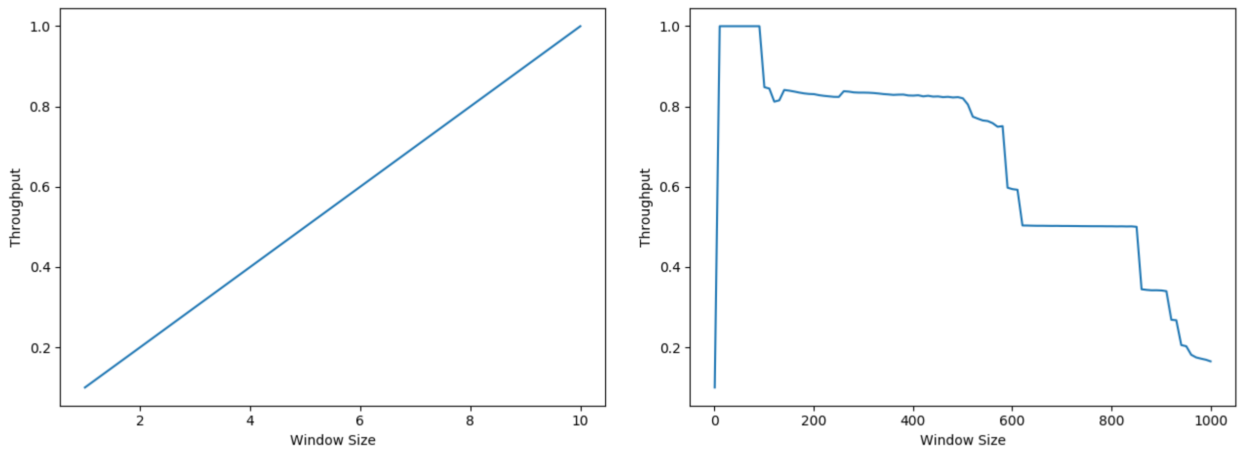


Figure 10: Simulation of congestion collapse with the above setting

Throughput increases until we reach window size of 10. In that phase, we have $\frac{W}{RTT_{min}} < C$, and therefore throughput increases linearly. In the second phase, we have throughput remaining the same, in which $RTT_{min} < \frac{W}{C} < T$ where T denotes estimated timeout. The packet latency is $\frac{W}{C}$ is less than timeout, and no packet gets transmitted. After that, we have $\frac{W}{C} > T$, and timeout is detected. The sender retransmits packets that are in the queue, making the throughput less than capacity [1].

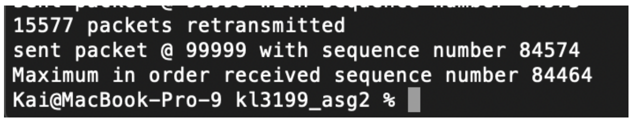

```
15577 packets retransmitted
sent packet @ 99999 with sequence number 84574
Maximum in order received sequence number 84464
Kai@MacBook-Pro-9 kl3199_asg2 %
```

Figure 11: Results with window size = 111

Numerically, let's look at the case with window size 111. To keep track of the number of packets retransmitted, I added a set in the SlidingWindowHost class variable. The sender sends out 84575 packets and receives 84465 packets. 15577 packets are retransmitted. A considerable amount of time is consumed in retransmitting those packets. Similar analysis applies to other cases with large window size. Therefore, throughput gets strictly less than capacity.

5 AIMD

5.1

Please see implementations in py files

5.2

Assume we have a relatively large window, it's likely we have multiple unacked packets at current time tick. If we don't wait, we would decrease the window size by half for every unacked packet, which approaches 0 quickly. Therefore, we should wait for RTT_{min}, since halving the window size once might be sufficient and decreasing it sharply means underutilizaiton.

5.3

Setting: seed=1, rtt_min=20, ticks=10000, queue_limit=10
Clearly, we have additive increase and multiplicative decrease. The period is around 800 (a rough estimate by inspection). Throughput is 0.7724

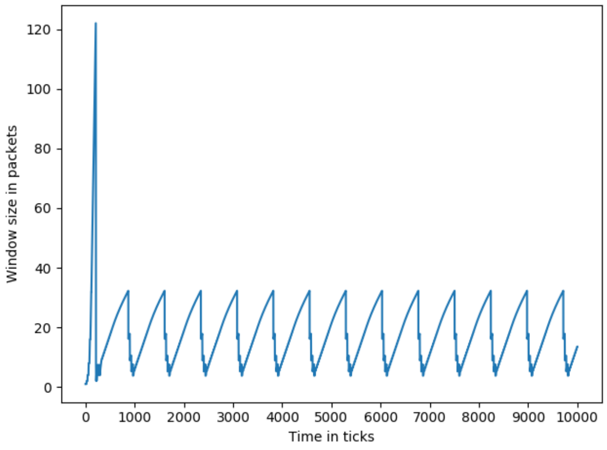
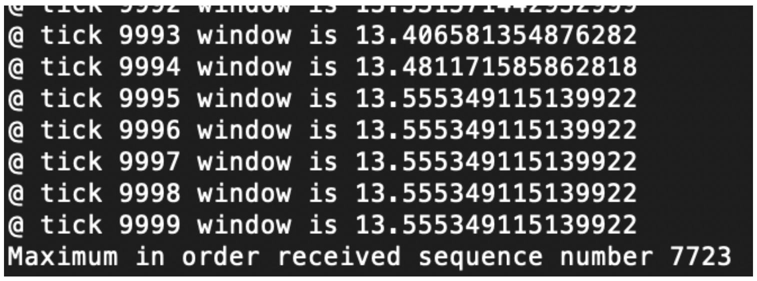


Figure 12: Window size with respect to time tick

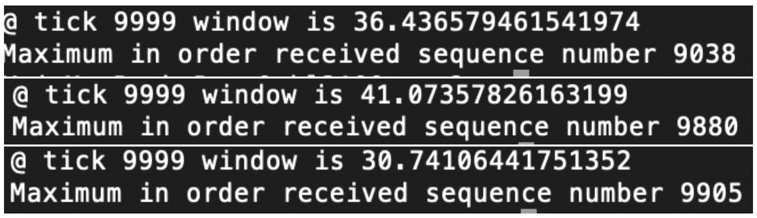


```
@ tick 9993 window is 13.406581354876282
@ tick 9994 window is 13.481171585862818
@ tick 9995 window is 13.555349115139922
@ tick 9996 window is 13.555349115139922
@ tick 9997 window is 13.555349115139922
@ tick 9998 window is 13.555349115139922
@ tick 9999 window is 13.555349115139922
Maximum in order received sequence number 7723
```

Figure 13: Command line output

5.4

Setting remains the same except varying queue limit
Throughputs are 0.9039, 0.9881, and 0.9906. Larger queue can hold more packets before it starts dropping packets. Therefore, we have less retransmission and higher throughput with larger queue.



```
@ tick 9999 window is 36.436579461541974
Maximum in order received sequence number 9038
```

```
@ tick 9999 window is 41.07357826163199
Maximum in order received sequence number 9880
```

```
@ tick 9999 window is 30.74106441751352
Maximum in order received sequence number 9905
```

Figure 14: Throughput for different queue limit (1 BDP, 2 BDP, 3 BDP)

5.5

The queue serves like a buffer that holds packets and reduces the number of retransmission, making the process of handling bursty arrivals smoother.

References

[1] Sivaraman Anirudh. *Lecture 6: Congestion collapse*. 2020. URL: `https://cs.nyu.edu/~anirudh/CSCI-UA.0480-062/lectures/lec6.pdf`.