BLG 433E Computer Communications Project 2

Due: Monday, May 20, 2024

Lecturer: Prof. Abdül Halim Zaim CRN: 21358

Fatih Baskın
150210710

May 20, 2024

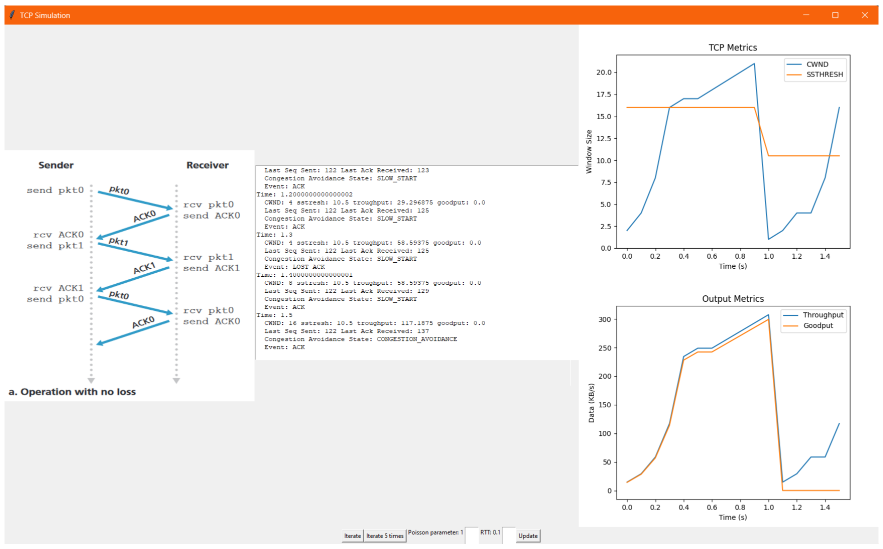
Contents

Simulation Program

How to run?

To be able to run the simulation program, first, you should create a Python virtual environment. Then, you should install the packages specified in the `requirements.txt` file in the virtual environment. After you do all the preparations, you should run the `run.py` file in the virtual environment.

UI Explanation



```
  Last Seq Sent: 122 Last Ack Received: 123
  Congestion Avoidance State: SLOW_START
  Event: ACK
Time: 1.2000000000000002
  CWND: 4 sstresh: 10.5 troughput: 29.296875 goodput: 0.0
  Last Seq Sent: 122 Last Ack Received: 125
  Congestion Avoidance State: SLOW_START
  Event: ACK
Time: 1.3
  CWND: 4 sstresh: 10.5 troughput: 58.59375 goodput: 0.0
  Last Seq Sent: 122 Last Ack Received: 125
  Congestion Avoidance State: SLOW_START
  Event: LOST ACK
Time: 1.4000000000000001
  CWND: 8 sstresh: 10.5 troughput: 58.59375 goodput: 0.0
  Last Seq Sent: 122 Last Ack Received: 129
  Congestion Avoidance State: SLOW_START
  Event: ACK
Time: 1.5
  CWND: 16 sstresh: 10.5 troughput: 117.1875 goodput: 0.0
  Last Seq Sent: 122 Last Ack Received: 137
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
```

Figure 1: Screenshot of the UI

After starting the `run.py` the screen will look like this. In below, you can iterate the simulation for 1 RTT or 5 RTTs, and you can modify the simulation parameters, RTT, and Poisson distribution mean.

On the left, there is a representing image showing the sequence diagram of the current event. in the middle, there is a text representation of what is going on, what the last sequence number sent, what is the last ack number received, cwnd, sstresh, throughput, goodput, current event, and current congestion avoidance algorithm running in the TCP simulation.

On the right, there are two graphs showing the last 20 iterations' performance metrics: cwnd, sstresh, throughput and goodput.

Simulation Program Logic

The simulation program does not run the simulation in real-time, on the contrary, it runs the segments iteratively, using the Poisson distribution. For the current problem, a random number is selected and it is compared with a set of random numbers.

- if $rand \leq 2$ then successful ack
- if $2 < rand \leq 4$ then duplicate ack (singe duplicate ack is assumed as lost ack)
- if $4 < rand$ then timeout (premature timeout)

If three double acks come next to each other then it is assumed package loss (triple duplicate ack) and **fast retransmit** protocol is used. Cwmd is set to $\frac{cwmd}{2}$ and mode is switched to additive increase (**congestion avoidance**).

If the timeout occurs then the sstresh is set to $\frac{cwmd}{2}$ and cwmd is set to 1 and mode is switched to **slow start**.

If $cwmd > sstresh$ in successful acks, then mode is switched to **congestion avoidance**.

Performance Metric Calculation

Calculation of cwmd and sstresh was given above. For calculating the throughput, in each iteration, the number of sent segments are given by the simulation. It is assumed that packages are 1500 bytes in size, including 40 bytes of TCP headers.

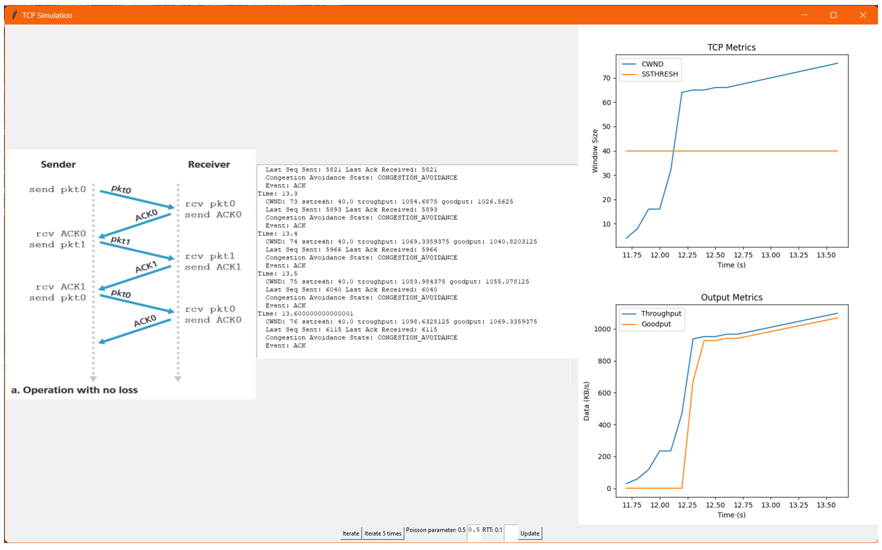
For calculating TCP troughput, basically $\frac{1500 \times num\_segments\_sent}{RTT}$ is used.

But for goodput, we should exclude the headers and retransmissions. We should always focus on the fresh data. Therefore there must be a mechanism to keep track of acked and not-acked segments. Therefore simulations keep track of these. The simulation also keeps track of which packets are retransmitted or not.

It is assumed that raw data in the packages are 1460 bytes and goodput is calculated like this: $goodput = \frac{num\_sent\_non-transmitted\_data \times 1460}{RTT}$.

TCP Events

Successful Ack Congestion Control



```
  Last Seq Sent: 5821 Last Ack Received: 5821
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
Time: 13.3
  CWND: 73 sstresh: 40.0 troughput: 1054.6875 goodput: 1026.5625
  Last Seq Sent: 5893 Last Ack Received: 5893
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
Time: 13.4
  CWND: 74 sstresh: 40.0 troughput: 1069.3359375 goodput: 1040.8203125
  Last Seq Sent: 5966 Last Ack Received: 5966
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
Time: 13.5
  CWND: 75 sstresh: 40.0 troughput: 1083.984375 goodput: 1055.078125
  Last Seq Sent: 6040 Last Ack Received: 6040
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
Time: 13.600000000000001
  CWND: 76 sstresh: 40.0 troughput: 1098.6328125 goodput: 1069.3359375
  Last Seq Sent: 6115 Last Ack Received: 6115
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
```

Poisson parameter: 0.5 0.5 RTT: 0.1

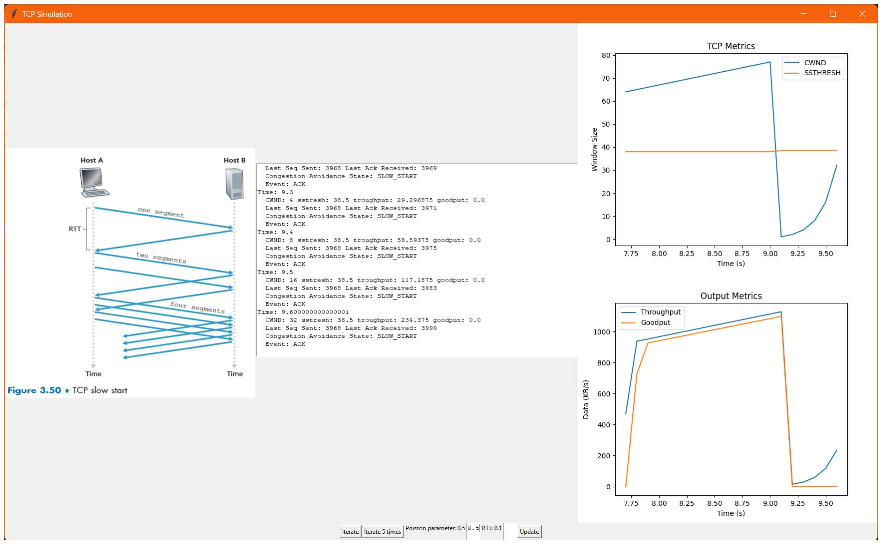
Figure 2: Successful acknowledgment and additive increase in congestion control.

In the figure above, the effects of successive successful acknowledgments are shown. Here, after $cwnd > sstresh$ and the TCP congestion control switches into congestion control mode and starts additive increase, it increases maximum segment size by 1 in each successive acks. This is very important in congestion control because connection limits are not reached that fast while increasing the connection's throughput.

TCP Events continued on next page…

Slow Start after Premature Timeout

Slow start occurs when a connection is newly established or on timeout events. The maximum segment size is doubled in each ack, therefore the increase is logarithmic and throughput is also logarithmic.



```
  Last Seq Sent: 3968 Last Ack Received: 3969
  Congestion Avoidance State: SLOW_START
  Event: ACK
Time: 9.3
  CWND: 4 sstresh: 38.5 troughput: 29.296875 goodput: 0.0
  Last Seq Sent: 3968 Last Ack Received: 3971
  Congestion Avoidance State: SLOW_START
  Event: ACK
Time: 9.4
  CWND: 8 sstresh: 38.5 troughput: 58.59375 goodput: 0.0
  Last Seq Sent: 3968 Last Ack Received: 3975
  Congestion Avoidance State: SLOW_START
  Event: ACK
Time: 9.5
  CWND: 16 sstresh: 38.5 troughput: 117.1875 goodput: 0.0
  Last Seq Sent: 3968 Last Ack Received: 3983
  Congestion Avoidance State: SLOW_START
  Event: ACK
Time: 9.600000000000001
  CWND: 32 sstresh: 38.5 troughput: 234.375 goodput: 0.0
  Last Seq Sent: 3968 Last Ack Received: 3999
  Congestion Avoidance State: SLOW_START
  Event: ACK
```

Poisson parameter: 0.5 0.5 RTT: 0.1

Figure 3: Slow start after timeout.

In the figure above, a slow start after a premature timeout is given. Since it is required to retransmit un-acked packages, goodput is initially 0. After un-acked packages are sent, then the goodput rebounds back to its normal ratio.

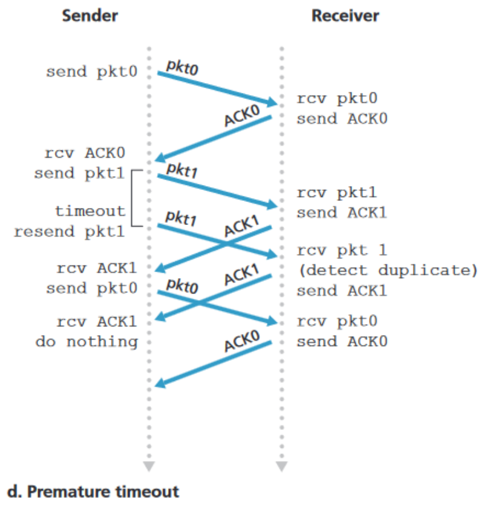


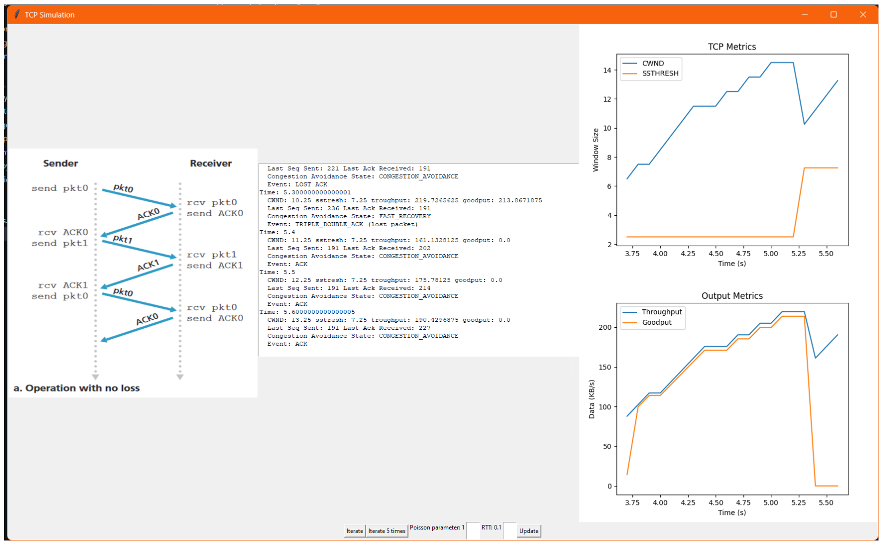
Figure 4: Sequence diagram of premature timeout

Lost Packet, Triple Duplicate Acknowledgement and Fast Retransmission

In Triple Duplicate Acknowledgement, the congestion control mechanism applies **s fast retransmission** to recover from and avoid congestion without hampering the performance. What happens is the maximum

TCP Events [Lost Packet, Triple Duplicate Acknowledgement and Fast Retransmission] continued on next page. . .

segment size is halved, so the sstresh is set to the same value and the algorithm continues in **congestion control** mode. Also goodput is negatively affected since retransmission occurs.



```
  Last Seq Sent: 221 Last Ack Received: 191
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: LOST ACK
Time: 5.300000000000001
  CWND: 10.25 sstresh: 7.25 troughput: 219.7265625 goodput: 213.8671875
  Last Seq Sent: 236 Last Ack Received: 191
  Congestion Avoidance State: FAST_RECOVERY
  Event: TRIPLE_DOUBLE_ACK (lost packet)
Time: 5.4
  CWND: 11.25 sstresh: 7.25 troughput: 161.1328125 goodput: 0.0
  Last Seq Sent: 191 Last Ack Received: 202
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
Time: 5.5
  CWND: 12.25 sstresh: 7.25 troughput: 175.78125 goodput: 0.0
  Last Seq Sent: 191 Last Ack Received: 214
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
Time: 5.6000000000000005
  CWND: 13.25 sstresh: 7.25 troughput: 190.4296875 goodput: 0.0
  Last Seq Sent: 191 Last Ack Received: 227
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
```

Figure 5: Lost Packet, Triple Duplicate Ack

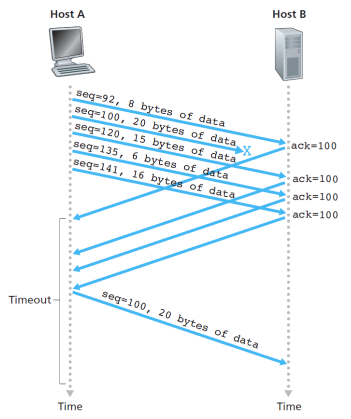


Figure 3.37 ♦ Fast retransmit: retransmitting the missing segment before the segment's timer expires

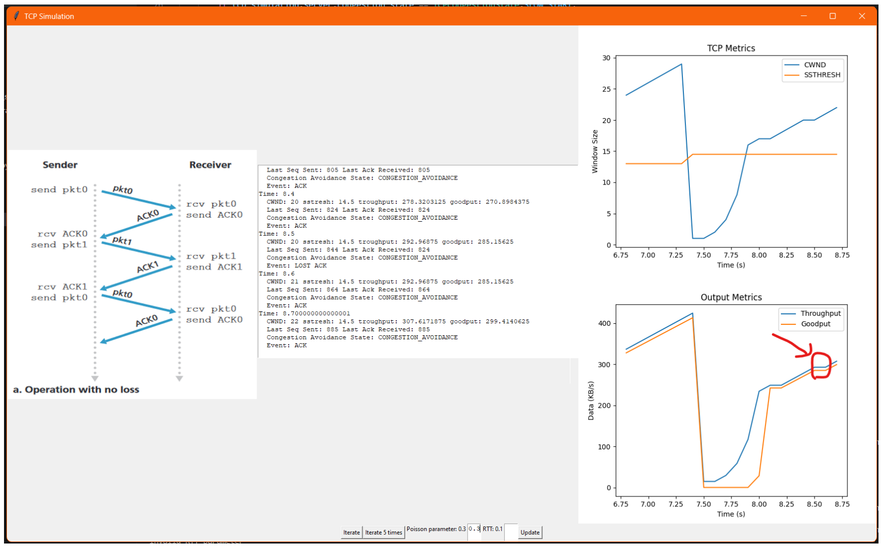
Figure 6: Sequence diagram of fast retransmission

Lost Ack

Lost acknowledgment occurs when an acknowledgment package can't reach the sender. Either a timeout occurs or another acknowledgment from pipelined packages acknowledges the lost packet. This simulation only simulates the latter case since it is hard to differentiate between premature timeout and timeout due to lost-ack.

With lost ack, additive increase in the congestion control stops abruptly and continues with new packages. It is important to note that this occurs since maximum segment size increases with each acknowledgment and if acknowledgments are lost, additive increase is delayed.

TCP Events [Lost Ack] continued on next page. . .



```
  Last Seq Sent: 805 Last Ack Received: 805
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
Time: 8.4
  CWND: 20 sstresh: 14.5 troughput: 278.3203125 goodput: 270.8984375
  Last Seq Sent: 824 Last Ack Received: 824
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
Time: 8.5
  CWND: 20 sstresh: 14.5 troughput: 292.96875 goodput: 285.15625
  Last Seq Sent: 844 Last Ack Received: 824
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: LOST ACK
Time: 8.6
  CWND: 21 sstresh: 14.5 troughput: 292.96875 goodput: 285.15625
  Last Seq Sent: 864 Last Ack Received: 864
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
Time: 8.700000000000001
  CWND: 22 sstresh: 14.5 troughput: 307.6171875 goodput: 299.4140625
  Last Seq Sent: 885 Last Ack Received: 885
  Congestion Avoidance State: CONGESTION_AVOIDANCE
  Event: ACK
```

Figure 7: Simulation of lost ack, highlighted with red

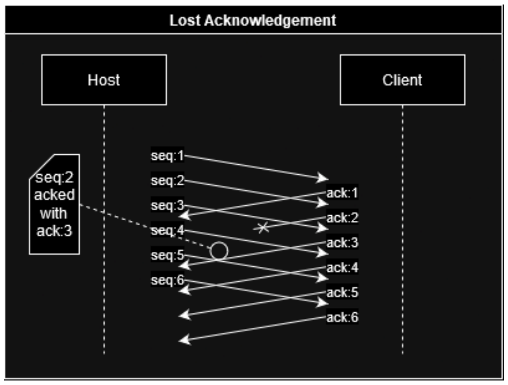


Figure 8: Sequence diagram of lost ack.

In the sequence diagram above, it is shown that pipelined packages and acknowledgment of later packages negate the risk of timeout due to lost acknowledgment but it temporarily stops the additive increase in the congestion control.