CS 421 ~ Computer Networks
FALL 2020

Programming Assignment 2 Report

Berrak Taşkınsu - 21602054
Irmak Demir - 21602603

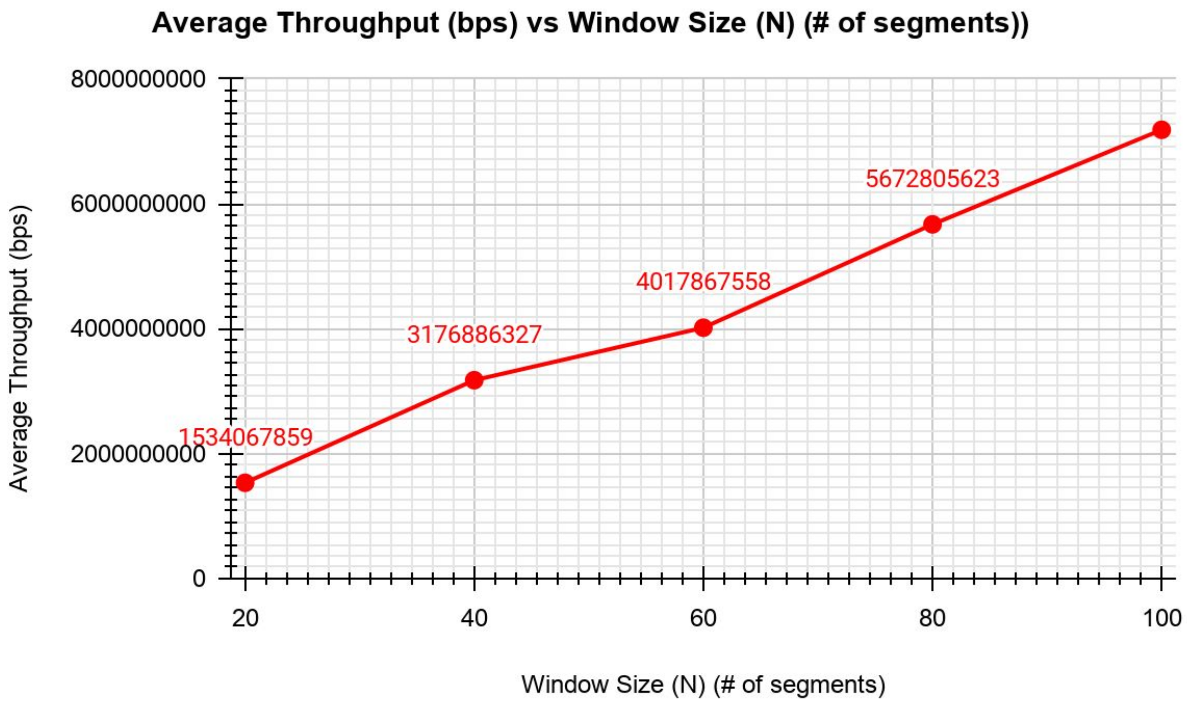
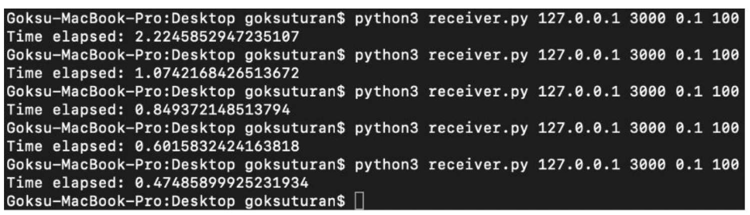
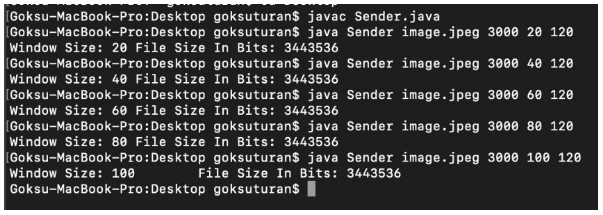


Figure 1: Average Throughput vs Window Size

The plot provided above shows how the average throughput changes as window size (N) increases. Here, the average throughput is in bps and it is calculated by dividing the file size (in bits) by the total elapsed time (in seconds). Window size is taken to be 20, 40, 60, 80 and 100 segments respectively. Note that the file size is constant throughout the experiment. Also throughout the experiment, the maximum delay, the retransmission timeout and ACK loss probability are taken to be 100 ms, 120 ms and 0.1, respectively. The plot is made according to the output below.



```
Goksu-MacBook-Pro:Desktop goksuturan$ python3 receiver.py 127.0.0.1 3000 0.1 100
Time elapsed: 2.2245852947235107
Goksu-MacBook-Pro:Desktop goksuturan$ python3 receiver.py 127.0.0.1 3000 0.1 100
Time elapsed: 1.0742168426513672
Goksu-MacBook-Pro:Desktop goksuturan$ python3 receiver.py 127.0.0.1 3000 0.1 100
Time elapsed: 0.849372148513794
Goksu-MacBook-Pro:Desktop goksuturan$ python3 receiver.py 127.0.0.1 3000 0.1 100
Time elapsed: 0.6015832424163818
Goksu-MacBook-Pro:Desktop goksuturan$ python3 receiver.py 127.0.0.1 3000 0.1 100
Time elapsed: 0.47485899925231934
Goksu-MacBook-Pro:Desktop goksuturan$ 
```



```
Goksu-MacBook-Pro:Desktop goksuturan$ javac Sender.java
Goksu-MacBook-Pro:Desktop goksuturan$ java Sender image.jpeg 3000 20 120
Window Size: 20 File Size In Bits: 3443536
Goksu-MacBook-Pro:Desktop goksuturan$ java Sender image.jpeg 3000 40 120
Window Size: 40 File Size In Bits: 3443536
Goksu-MacBook-Pro:Desktop goksuturan$ java Sender image.jpeg 3000 60 120
Window Size: 60 File Size In Bits: 3443536
Goksu-MacBook-Pro:Desktop goksuturan$ java Sender image.jpeg 3000 80 120
Window Size: 80 File Size In Bits: 3443536
Goksu-MacBook-Pro:Desktop goksuturan$ java Sender image.jpeg 3000 100 120
Window Size: 100        File Size In Bits: 3443536
Goksu-MacBook-Pro:Desktop goksuturan$ 
```

An increase in window size results in a significant increase in throughput of file transfer. This is due to the fact that the sender does not stop transmitting packets one after another until the last packet in the window is sent. As the size of the window increases the sender module becomes more independent on the incoming acknowledgements, and eventually, file transfer is more rapid and smooth.

Yet acceleration of the throughput decreases as window size increases. This is due to the fact that when the window is larger, then in the case of a retransmission timeout, since the whole window has to be retransmitted, elapsed time increases more and more gradually. The reason behind the whole window being retransmitted in case of timeout is due to Go Back N Protocol.

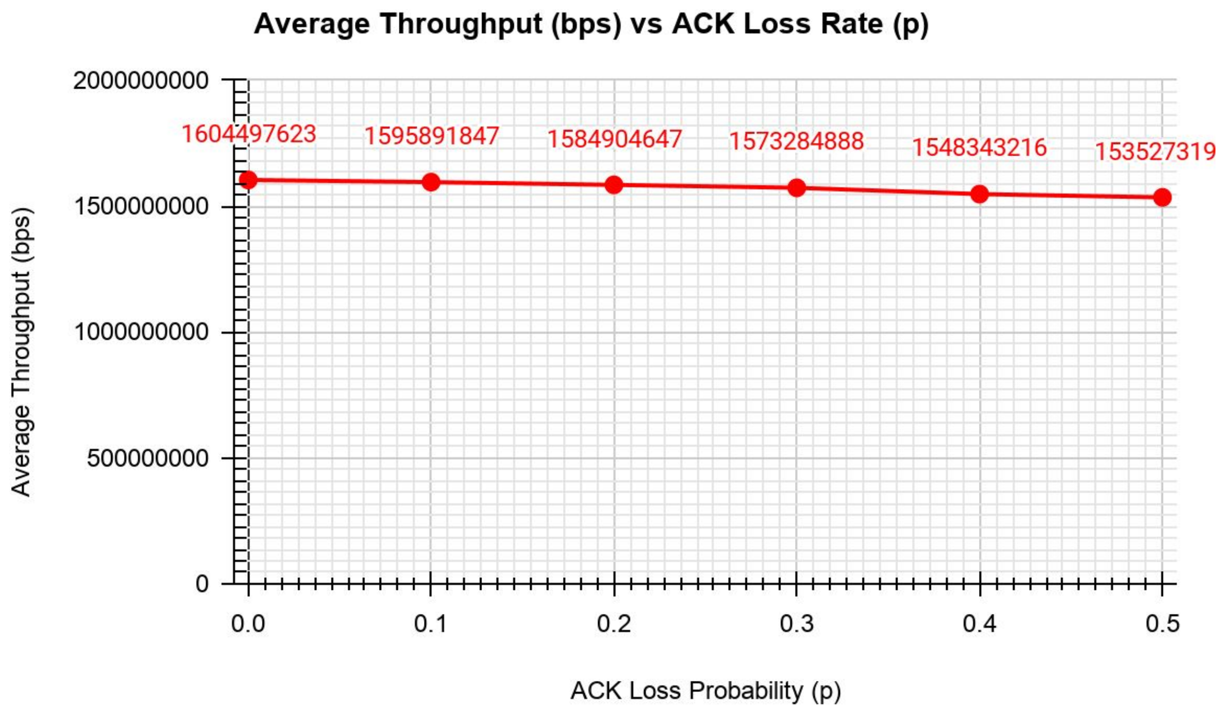
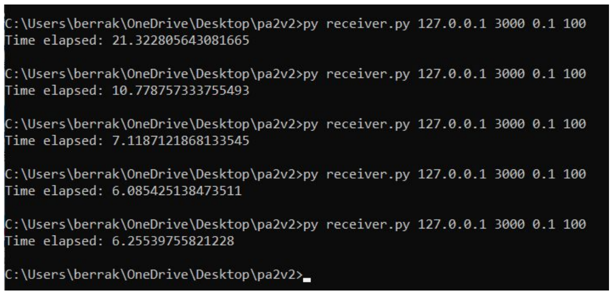
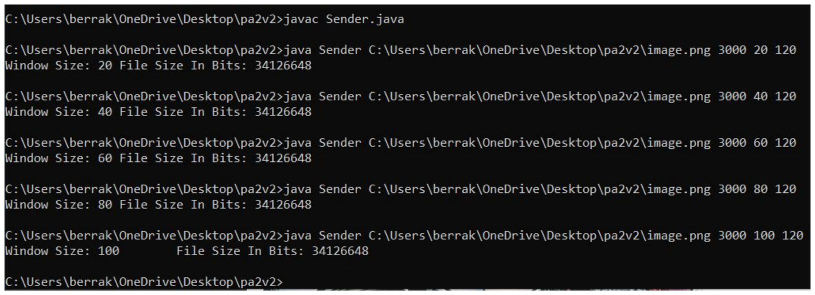


Figure 1: Average Throughput vs ACK Loss Probability

The plot provided above shows how the average throughput changes as ACK loss probability (p) increases. Here, the average throughput is in bps and it is calculated by dividing the file size (in bits) by the total elapsed time (in seconds). ACK loss probability is taken to be 0, 0.1, 0.2, 0.3, 0.4 and 0.5 respectively. Note that the file size is constant throughout the experiment. Also throughout the experiment, the maximum delay, the retransmission timeout and window size (N) are taken to be 100 ms, 120 ms and 20, respectively. The plot is made according to the output provided below.



```
C:\Users\berrak\OneDrive\Desktop\pa2v2>py receiver.py 127.0.0.1 3000 0.1 100
Time elapsed: 21.322805643081665

C:\Users\berrak\OneDrive\Desktop\pa2v2>py receiver.py 127.0.0.1 3000 0.1 100
Time elapsed: 10.778757333755493

C:\Users\berrak\OneDrive\Desktop\pa2v2>py receiver.py 127.0.0.1 3000 0.1 100
Time elapsed: 7.1187121868133545

C:\Users\berrak\OneDrive\Desktop\pa2v2>py receiver.py 127.0.0.1 3000 0.1 100
Time elapsed: 6.085425138473511

C:\Users\berrak\OneDrive\Desktop\pa2v2>py receiver.py 127.0.0.1 3000 0.1 100
Time elapsed: 6.25539755821228

C:\Users\berrak\OneDrive\Desktop\pa2v2>
```



```
C:\Users\berrak\OneDrive\Desktop\pa2v2>javac Sender.java

C:\Users\berrak\OneDrive\Desktop\pa2v2>java Sender C:\Users\berrak\OneDrive\Desktop\pa2v2\image.png 3000 20 120
Window Size: 20 File Size In Bits: 34126648

C:\Users\berrak\OneDrive\Desktop\pa2v2>java Sender C:\Users\berrak\OneDrive\Desktop\pa2v2\image.png 3000 40 120
Window Size: 40 File Size In Bits: 34126648

C:\Users\berrak\OneDrive\Desktop\pa2v2>java Sender C:\Users\berrak\OneDrive\Desktop\pa2v2\image.png 3000 60 120
Window Size: 60 File Size In Bits: 34126648

C:\Users\berrak\OneDrive\Desktop\pa2v2>java Sender C:\Users\berrak\OneDrive\Desktop\pa2v2\image.png 3000 80 120
Window Size: 80 File Size In Bits: 34126648

C:\Users\berrak\OneDrive\Desktop\pa2v2>java Sender C:\Users\berrak\OneDrive\Desktop\pa2v2\image.png 3000 100 120
Window Size: 100        File Size In Bits: 34126648

C:\Users\berrak\OneDrive\Desktop\pa2v2>
```

As ACK loss probability (p) increases, the possibility of the Sender.java module not to receive the expected acknowledgement increases.Yet, the sender module still updates the window when an acknowledgment with a higher sequence number than expected is received. This is due to the cumulative acknowledgement property of Go Back N Protocol. The Go Back N Protocol is intelligent in the sense that it can understand whether an acknowledgement is lost or simply the package has not reached the receiver yet. This is the case since the receiver discards the out of order packages instead of buffering them as in Selective Repeat algorithm. Therefore, if a package is lost, the receiver keeps sending the same acknowledgement. As a result, when the sender receives an acknowledgement of a package with higher sequence number, it understands that indeed an acknowledgement is late or lost and pretends as if the skipped acknowledgement is also received.

This is not the case in Selective Repeat algorithm since it uses buffering instead, and therefore sends corresponding acknowledgements even if packets are out of order. Therefore, the sender is never intelligent in determining whether a delay is due to an ACK loss or a packet loss. That is the reason for the Selective Repeat algorithm to work more efficiently when compared to GBN when there are only packet losses. Yet, efficiency of Go Back N algorithm does not highly depend on acknowledgement losses, but instead on packet losses. Since in this homework, there are only acknowledgement losses, an increase in the number of losses does not significantly increase the elapsed time.

The increase in elapsed time is more due to timeouts. When an acknowledgement is lost, the possibility of retransmission timeout to occur also increases. Therefore, the average throughput decreases as ACK loss probability increases, but slowly due to previously discussed reasons.