Homework 3 - Sliding Window

Temesgen Habte, James Kim, Brandon Hu, Bryant Nguyen

Table of Contents

Introduction

For this homework assignment, our goal was to implement a stop and wait algorithm as well as a sliding window algorithm for reliable transfer of data over UDP. Not only do we measure the time it takes for the test to complete, we also measure the number of packets that needed to be retransmitted. This document is split up into two parts: The documentation with detail about the algorithms and the discussion with the analysis of what the times mean.

Preliminary Notes

The linux machines given to us for this lab could not connect to each other even though we set the ports to be between 4000 and 6000. We even tried using the regular linux lab machines with no luck as well. We are unsure as to why this was the case but instead of endlessly trying to make it work, we decided to just run it on one of our local networks. The following tests were executed between a personal desktop and a virtual machine on **separate** hardware. They were linked by a router and each had a 1gbps connection.

The files can be compiled using the same commands given in the assignment description. hw3.cpp has not been altered in any way so everything else can be compiled with the hw3.cpp that was included in the zip folder given to us in the assignment description.

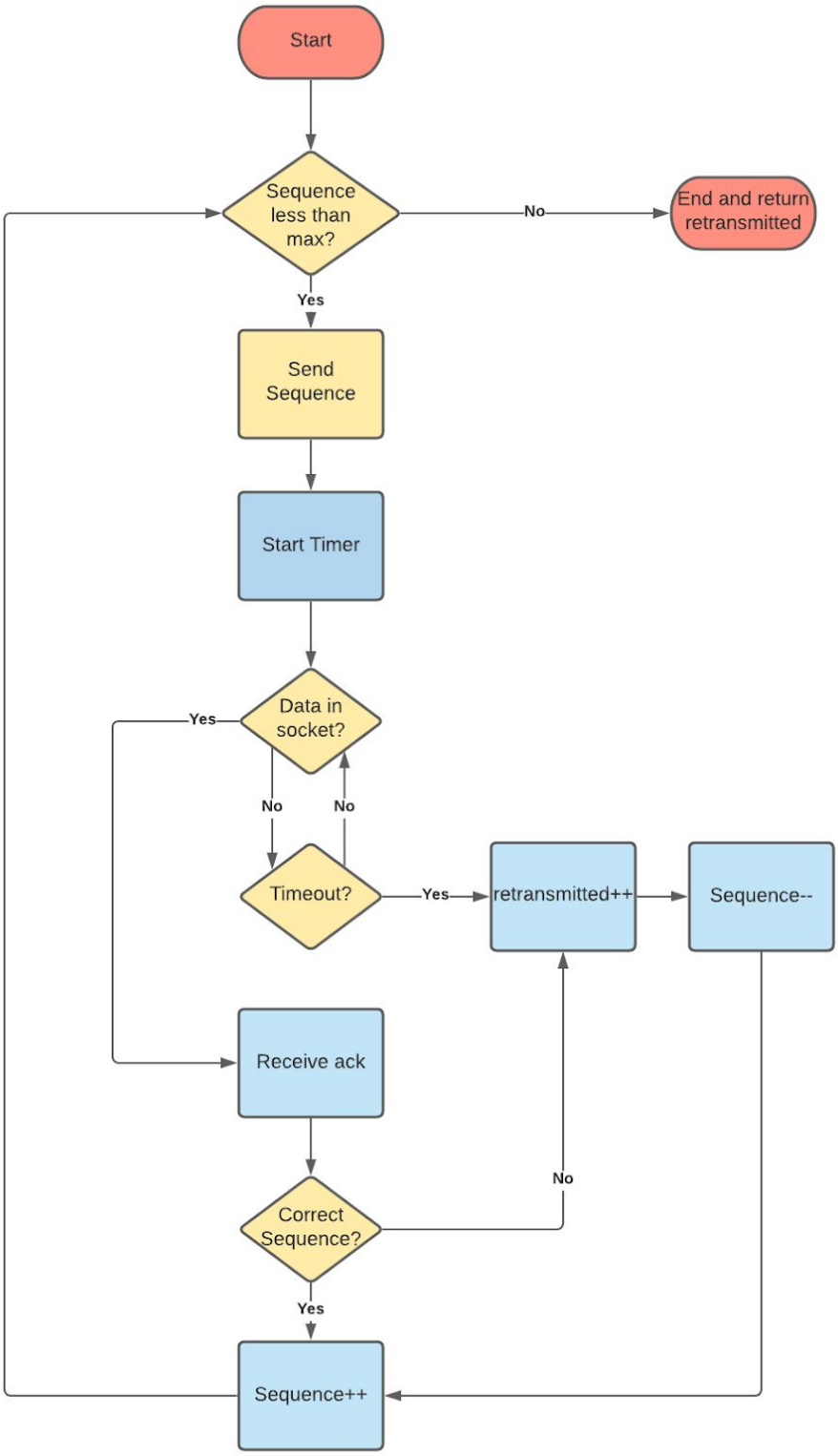
Documentation

Unreliable Transfer

Unreliable transfer is just the client sending everything to the server regardless of if there has been an acknowledgement from the server. This method was given to us with the assignment so we are not going to go into detail on how that is implemented.

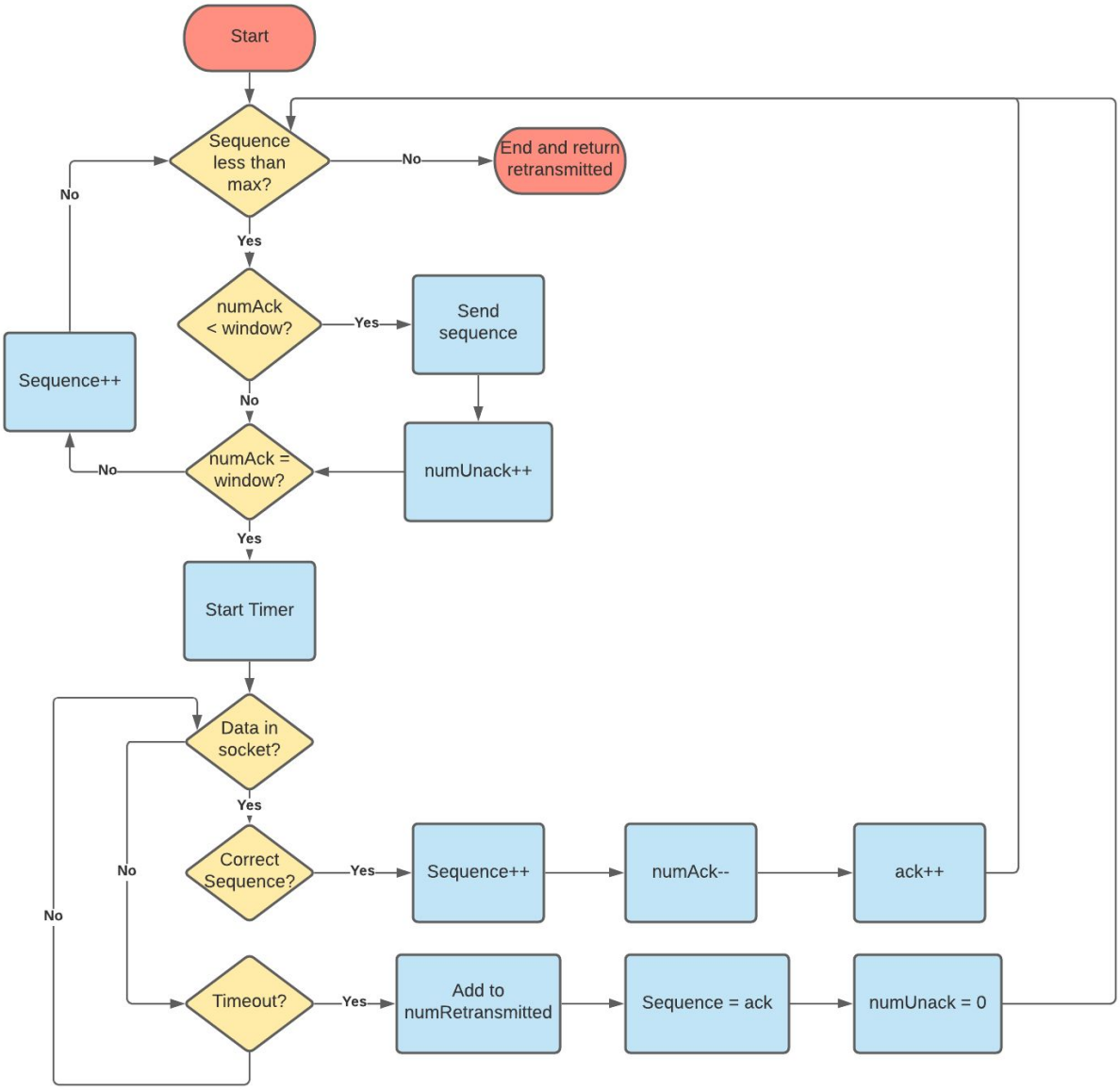
Client Stop and Wait

The stop and wait algorithm is quite simple. It sends the data, then waits for an ack before sending the next piece of data. If there is no ack after reaching the timeout threshold or if the ack is for the wrong data, the same piece of data will be sent again. It can also be seen as a sliding window algorithm with a window size of 1. A diagram of the logic is shown below.


Start
Sequence less than max?
No
End and return retransmitted
Yes
Send Sequence
Start Timer
Data in socket?
Yes
No
No
Timeout?
Yes
retransmitted++
Sequence--
Receive ack
Correct Sequence?
No
Yes
Sequence++

Sliding Window

The sliding window algorithm is when there is a window of packets that are sent out and waiting for an ack. The window proceeds to slide over to the next series of data once the acks have been received. If a timeout occurs, the window resets to the last ack received and starts over from there. The test includes simulations for window sizes from 1 to 30. A diagram of the logic is shown below.



Discussion

Unreliable Transfer

After running the first case test several times over a 1gbps network, the results were mostly the same. We have not had an instance of it not finishing completely, even on the server side. The amount of time it took to complete the test was also consistent. Below is a screenshot of unreliable transfer output.

```
message = 19994
message = 19995
message = 19996
message = 19997
message = 19998
message = 19999
Elasped time = 4535676
finished
bhu413@Brandon-DT:/mnt/c/Users/Brandon/Documents/GitHub/css432/Homework/HW 3/hw3$
```

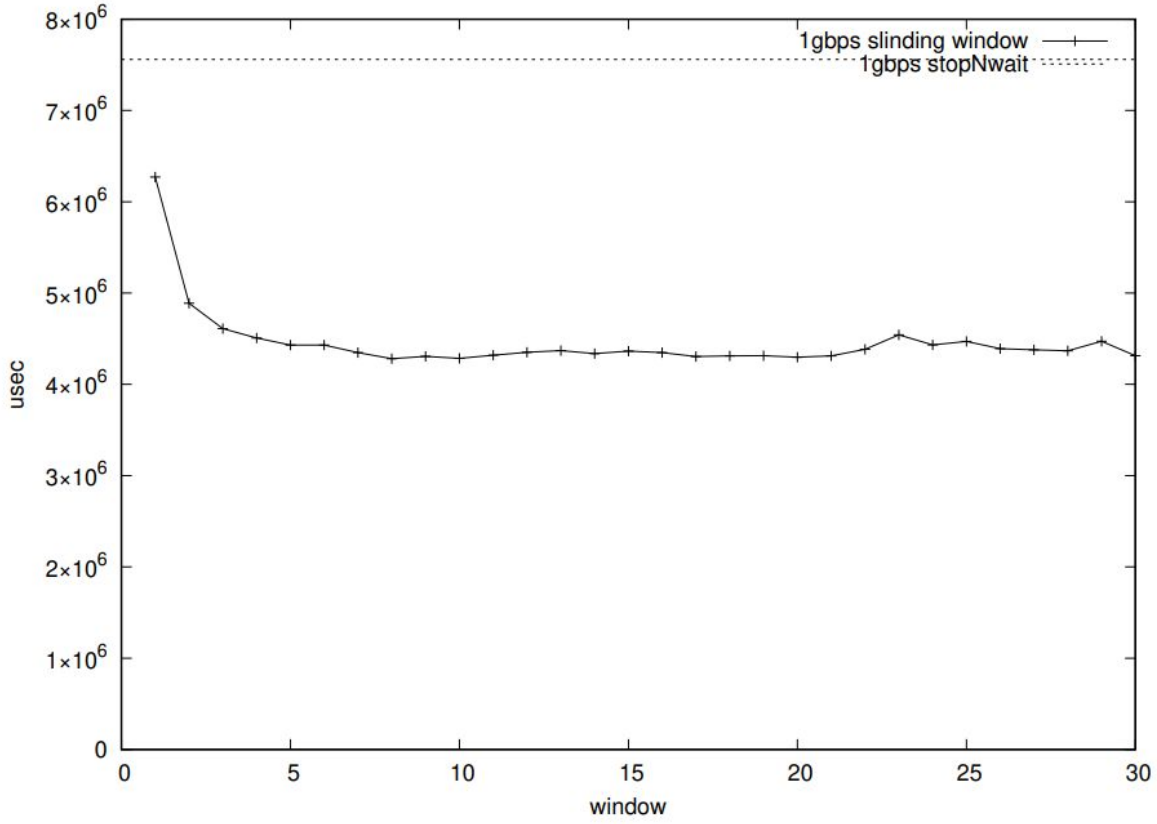
Stop and Wait

While only 4 packets needed to be transmitted on this particular run, we can see that performance is definitely worse than just pure UDP with no acknowledgement as seen in test case 1. Since we got **7559061** usec as our elapsed time for this test, this is the number we are inputting in the "TTTTT" section of the udp.plt file. This would be the baseline to compare with the sliding window performance. After doing this test multiple times, it is safe to say that the results are also consistent, taking the same time and not dropping more than 10 packets at a time.

```
19997
19998
19999
Elasped time = 7559061
retransmits = 4
finished
bhu413@Brandon-DT:/mnt/c/Users/Brandon/Documents/GitHub/css432/Homework/HW 3/hw3$
```

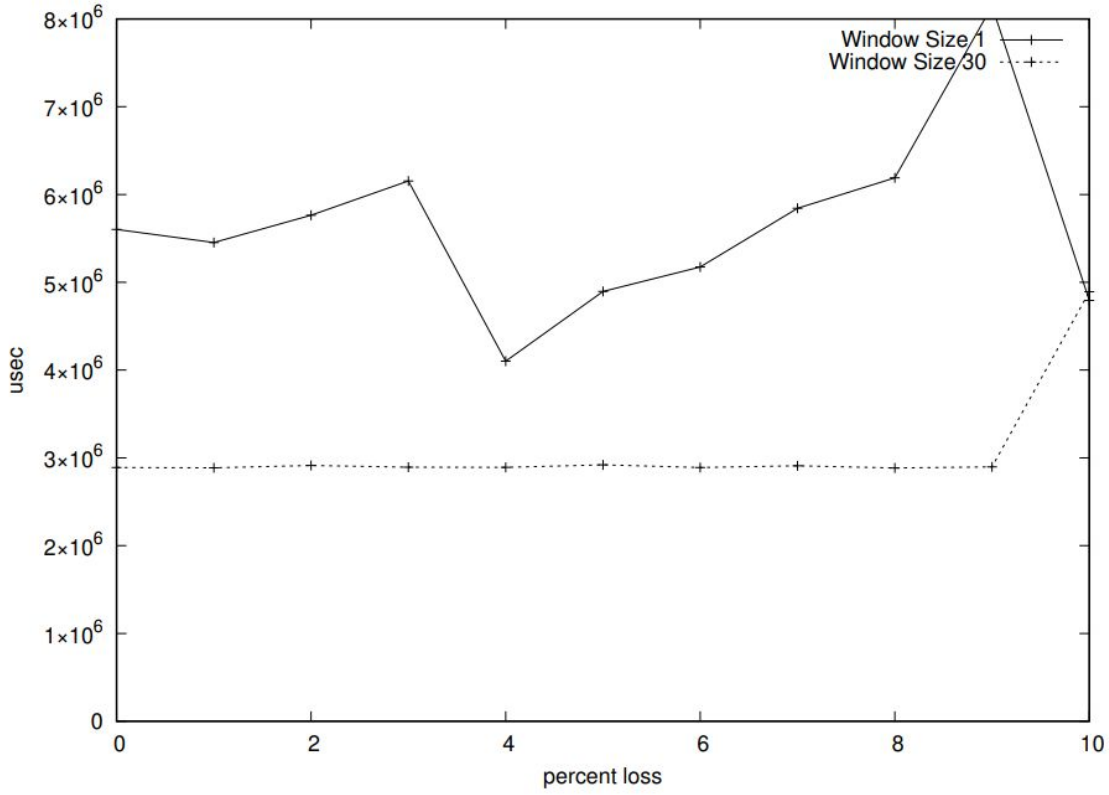
Sliding Window

As we can see with the graph below, a window size of 1 is quite similar in performance to the stop and wait algorithm. This makes sense because the stop and wait algorithm can be seen as a sliding window with a window size of one, as we mentioned before. After the single window test, performance increases dramatically and then stays relatively constant all the way until the end. Overall, the performance of the sliding window is much better than the stop and wait algorithm and seems to be consistent even as the window size increases.



Case 4

The sliding window with packet loss definitely gave the most interesting results. We can see from the graph below that the window size of 1 had worse performance and was inconsistent with the time it took to finish the test. On the other hand, a window size of 30 made it extremely consistent and had better performance. The performance gain was expected as it is sliding window vs stop and wait but we did not expect it to be that consistent, especially considering the increase in packet loss. At the end, there seems to be a dramatic increase for 10 percent loss and we are unsure why this was the case. Perhaps it was just unlucky and a high amount of packets were lost.


Window Size 1
Window Size 30
usec
percent loss
8×10^6
7×10^6
6×10^6
5×10^6
4×10^6
3×10^6
2×10^6
1×10^6
0
0
2
4
6
8
10