

HW3 – Sliding Window

Table of Contents

Overview of homework 3.....	1
What is included in the submission?	1
Overview of hw3.cpp	2
Stop-And-Wait algorithm.....	2
Client side.....	2
Server side.....	3
Sliding window algorithm	4
Client side.....	4
Server side.....	5
Overview of hw3case4.cpp	6
How to compile	6
Execution output.....	7
Performance evaluation	9

Overview of homework 3

Within this assignment, I will implement a stop-and-wait and sliding algorithms and evaluate their performance when transferring 20000 packets over 1Gbps networks using the schools network

Common Error

If “cannot bind the local address to the UDP socket”, probably, you exited the program forcefully that made it to skip the ability to reuse the address. You can either wait it to time out, who know how long

Or you can edit the hw3.cpp or hw3case4.cpp file and change the port number

If somehow case 2 or 3 got stuck, try restarting it with different port number and ensure that you enter the correct order. Invoke server first, select the option in server. Invoke client with server name, select option in client

What is included in the submission?

Within the submission, I have included the following:

- Hw3.cpp: This is important since this one is different with the hw3 within the zip file on the assignment page. This one has the code to call the methods for the stop and wait and sliding algorithm

- UdpHw3 header and cpp file: Include the definition and source file for case 2 and 3. This will be called from hw3.cpp
- Hw3case4.cpp: Exactly like hw3.cpp. But this one include the test case 4. Note that test case 1 and 3 are similar to hw3.cpp. So this file should only be used for running test case 4
- UdpHw3case4: Only implement test case 4 (which modified case 3) and be used by hw3case4.cpp

Other files such as udpSocket and Timer are not included. I recommend you read the overview before compiling since it can be a little tricky. I wish I can improve this but the description required this

Overview of hw3.cpp

Hw3.cpp is the place where the multiple test cases will be used. There are 3 cases within it

- Client and server unreliable: Send the data all out and does not wait at all. This is already implemented so I won't talk about this
- Client stop and wait and server unreliable: Implement the stop and wait algorithm. I will talk about this
- Client sliding window and server early retransmission: Implement the sliding window algorithm. I will also talk about this

The stop and wait algorithm and the sliding window are implemented within udpHw3. The hw3.cpp class only call those methods. **Because of this, hw3.cpp is needed within the submission in order to compile and run**

Stop-And-Wait algorithm

Client side

The name of the algorithm pretty much explains what it do. The client will send out a package and wait until it receives a response. Only after it receive a response and confirm that it is what the client is waiting for, it will advance and move on. Or if the response is not correct or timeout, it will resend the message

Below is a diagram that represent the flow of the client side for the algorithm

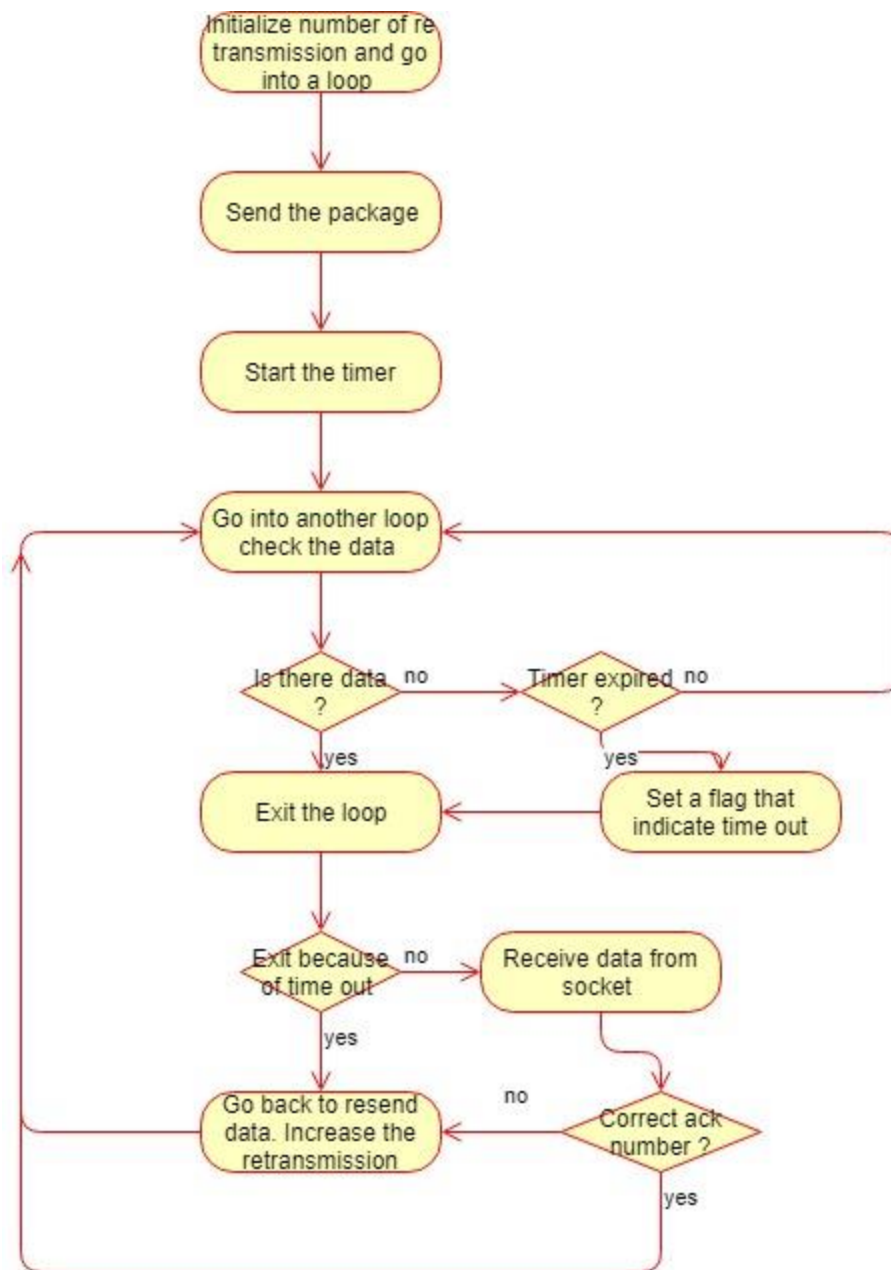


Figure 1 Client side illustration for stop and wait

Server side

The server is expecting a certain package from the client. It will only send the acknowledgement if the data received from the socket match with the data it is expecting and advance to receive next package. Otherwise, it will keep checking if there is anything and check the data

Below is a diagram that represent the flow of the server side

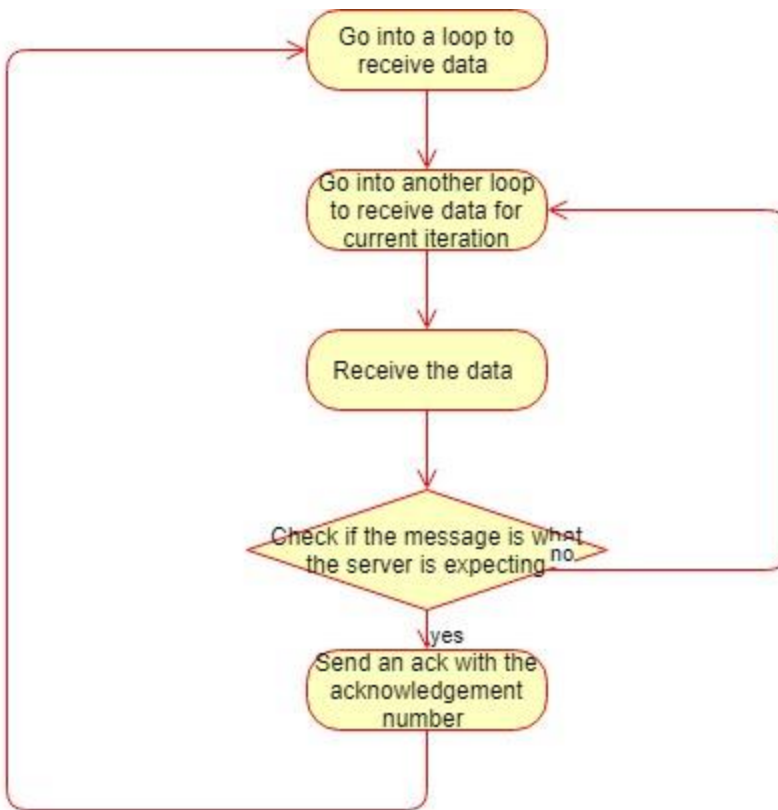


Figure 2 Server side illustration for stop and wait

Sliding window algorithm

Client side

Using the sliding window algorithm, the client will keep sending out message until it reach a window size. Once that happen, a timer will begin and if it time out and at that time, the number of un-acknowledgement message is still the window size, then it will start again and resend it again. Each time it receives a correct in order ack, then it will slide the window and reduce the number of un-acknowledgement

Below is a diagram that represent the flow for the client side of the sliding window

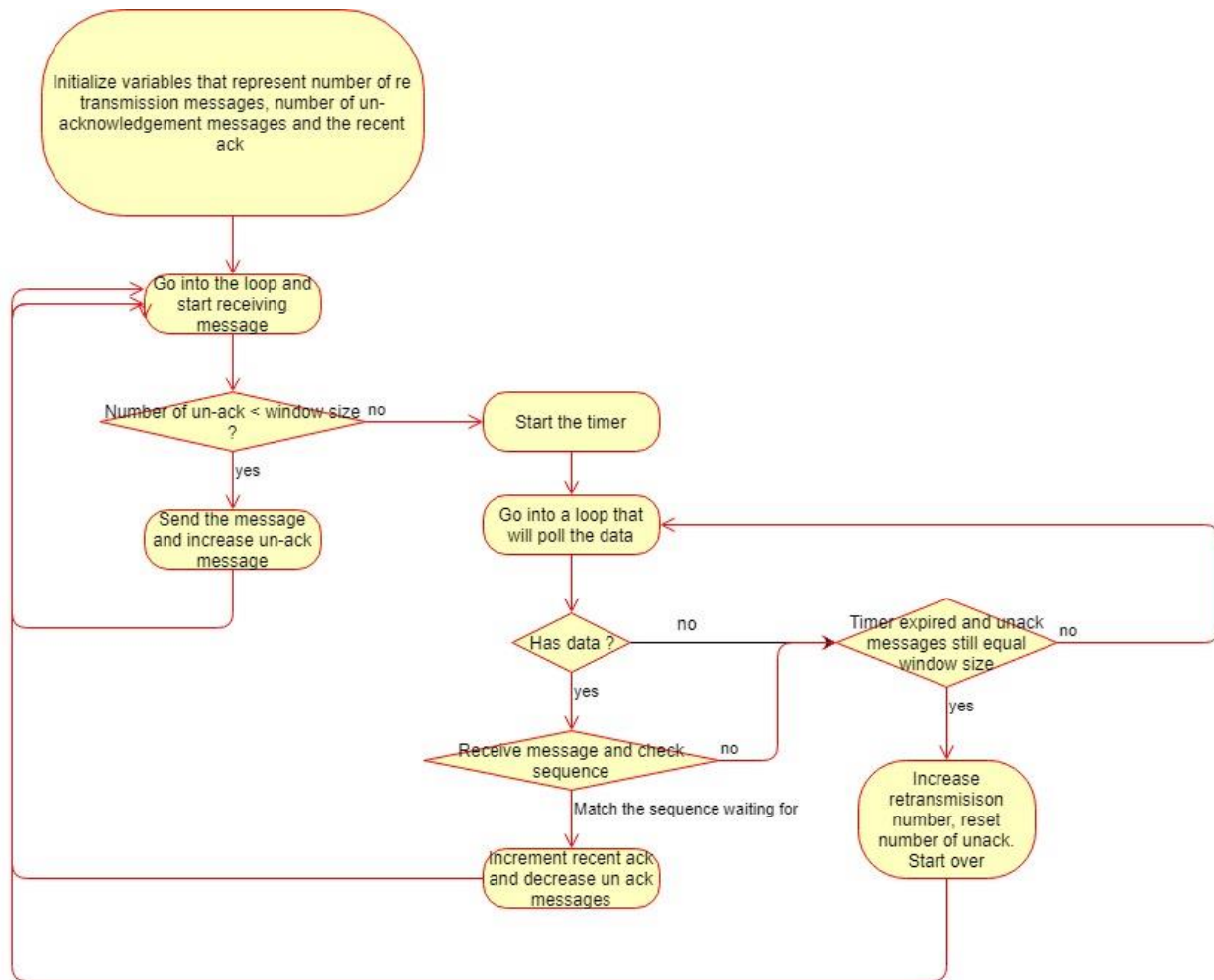


Figure 3 Client side illustration for sliding window

Server side

There are some interesting thing for the server side. Professor Peng has redesigned the problem description compared to professor Fukuda. In which, the server will discard any ack that it out of order and will send back ack with the recent number, which is in order. This made the window size and the array to recognize which ack have received is not necessary anymore. At first, I thought that this will follow selective repeat, but professor Peng has clarified this.

Below is a diagram that represent the flow of the server side. Note that it is similar to stop and wait but only has the difference in which it will always ack but with the recent ack number

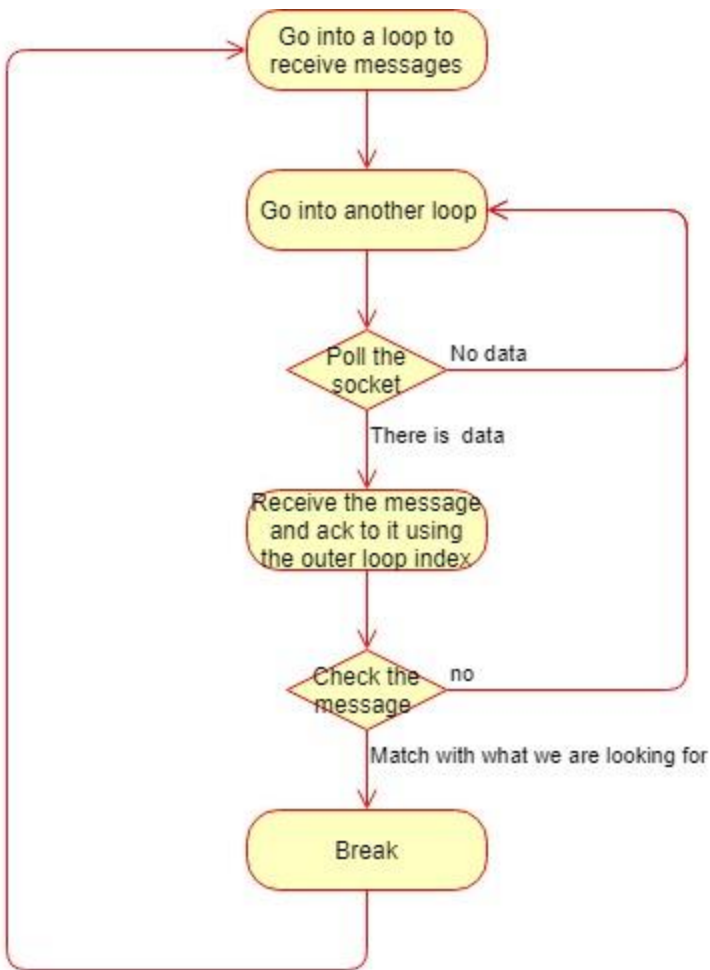


Figure 4 Server side illustration for sliding window

Overview of hw3case4.cpp

This file just implement the case 4 where packet are randomly dropped, which is based in case 3. Every time there is data to read , it will generate a random number from 0 – 100 and check if it should drop or not

This is the same as the server side for case 3, except now it has the ability to drop and also take in a drop rate as a parameter

Case 4 use both udphw3 and udphw3case4, in which the client method is reused.

Note that since case 4 is based on a modified case 3, hw3case4.Cpp should only be used for case 4 testing. Since the instruction required me to overwrite the method, then case 3 is no longer applicable

How to compile

For the general case of client stop and wait and sliding window (case 2 and 3), you should need both the timer and udpsocket class, which are not included that you can download from the assignment page

You also need hw3.cpp that is included (this is different compared to hw3.cpp within the assignment page), header and class udphw3

And remember to use `-std=c++14` to compile them

For the case 4, replace the `hw3.cpp` with `hw3case4.cpp`. And use `udphw3case4` as well

Either you compile them manually where you list out the file. Or after you tested the general case, delete `hw3.cpp` and add in `hw3case4.cpp` and use `g++ -std=c++14 *.cpp`

To sum up, to run case 2 and 3, you need:

- `Hw3.cpp`
- `Timer` both `cpp` and `.h` (Downloaded from assignment page)
- `Udphw3` both header and `cpp`
- `Udpsocket` both `cpp` and `.h` (Downloaded from assignment page)

To run case 4, you need:

- `Hw3case4.cpp`
- `Timer` (Downloaded from assignment page)
- `UdpSocket` (Downloaded from assignment page)
- `Udphw3` both header and `cpp`
- `Udphw3case4` both header and `cpp`

Once you got them in correct position (separate folder), run `g++ -std=c++14 *.cpp`

Run the server first, type `./a.out`

Run the client later, type `./a.out [machine name of the server]`

Remember to choose the option the same for both server and client

Execution output

Below are two pictures of the execution output for case 4 and case 2 and 3

```

Select thuan@Thuan-Laptop: ~
Window size = 5 Elapsed time = 551022
retransmits = 0
Window size = 6 Elapsed time = 496274
retransmits = 0
Window size = 7 Elapsed time = 417230
retransmits = 0
Window size = 8 Elapsed time = 508090
retransmits = 6031
Window size = 9 Elapsed time = 505557
retransmits = 7083
Window size = 10 Elapsed time = 507877
retransmits = 1179
Window size = 11 Elapsed time = 705123
retransmits = 1119
Window size = 12 Elapsed time = 740670
retransmits = 1307
Window size = 13 Elapsed time = 677620
retransmits = 2001
Window size = 14 Elapsed time = 622253
retransmits = 1343
Window size = 15 Elapsed time = 560533
retransmits = 1229
Window size = 16 Elapsed time = 549936
retransmits = 1279
Window size = 17 Elapsed time = 520333
retransmits = 611
Window size = 18 Elapsed time = 517324
retransmits = 215
Window size = 19 Elapsed time = 480150
retransmits = 227
Window size = 20 Elapsed time = 417117
retransmits = 519
Window size = 21 Elapsed time = 418082
retransmits = 629
Window size = 22 Elapsed time = 416311
retransmits = 131
Window size = 23 Elapsed time = 364698
retransmits = 137
Window size = 24 Elapsed time = 373953
retransmits = 143
Window size = 25 Elapsed time = 389365
retransmits = 40
Window size = 26 Elapsed time = 375157
retransmits = 1195
Window size = 27 Elapsed time = 358121
retransmits = 971
Window size = 28 Elapsed time = 317679
retransmits = 55
Window size = 29 Elapsed time = 310265
retransmits = 403
Window size = 30 Elapsed time = 275302
retransmits = 659
Finished
oda1234@uwl-320-09:~/CS5432/HW3/Genes ./a.out uwl-320-04
Choose a testcase
1: unreliable test
2: stop-and-wait test
3: sliding windows
--> 2
Elapsed time = 2879414
retransmits = 2
Finished
oda1234@uwl-320-09:~/CS5432/HW3/Genes

```

Figure 5 Execution output for stop and wait and sliding window

```

3: sliding windows
4: Sliding window with drop
--> 4
drop rate = 0 Window size = 1 Elapsed time = 1294043
retransmits = 3
drop rate = 1 Window size = 1 Elapsed time = 2704340
retransmits = 1
drop rate = 2 Window size = 1 Elapsed time = 1935803
retransmits = 1
drop rate = 3 Window size = 1 Elapsed time = 2516364
retransmits = 1
drop rate = 4 Window size = 1 Elapsed time = 2069781
retransmits = 1
drop rate = 5 Window size = 1 Elapsed time = 1376489
retransmits = 7
drop rate = 6 Window size = 1 Elapsed time = 2593128
retransmits = 5
drop rate = 7 Window size = 1 Elapsed time = 1893259
retransmits = 1
drop rate = 8 Window size = 1 Elapsed time = 1557079
retransmits = 3
drop rate = 9 Window size = 1 Elapsed time = 2021572
retransmits = 1
drop rate = 10 Window size = 1 Elapsed time = 2875918
retransmits = 0
drop rate = 0 Window size = 30 Elapsed time = 317368
retransmits = 0
drop rate = 1 Window size = 30 Elapsed time = 327646
retransmits = 609
drop rate = 2 Window size = 30 Elapsed time = 287123
retransmits = 419
drop rate = 3 Window size = 30 Elapsed time = 322676
retransmits = 59
drop rate = 4 Window size = 30 Elapsed time = 294907
retransmits = 59
drop rate = 5 Window size = 30 Elapsed time = 313394
retransmits = 299
drop rate = 6 Window size = 30 Elapsed time = 325105
retransmits = 179
drop rate = 7 Window size = 30 Elapsed time = 306022
retransmits = 179
drop rate = 8 Window size = 30 Elapsed time = 327664
retransmits = 0
drop rate = 9 Window size = 30 Elapsed time = 326558
retransmits = 179
drop rate = 10 Window size = 30 Elapsed time = 322340
retransmits = 179
Finished
oda1234@uwl-320-09:~/CS5432/HW3/Case4

```

Figure 6 Execution output for case 4

Performance evaluation

This section only shows the graph. Within the discussion, I will go into details discussing about them

Some graphs below will have stop and wait in combination with window size. The window size has no effect here on the stop and wait. I just followed what is in udp.pdf file and re run the case 2 multiple time

Stop and wait and sliding window

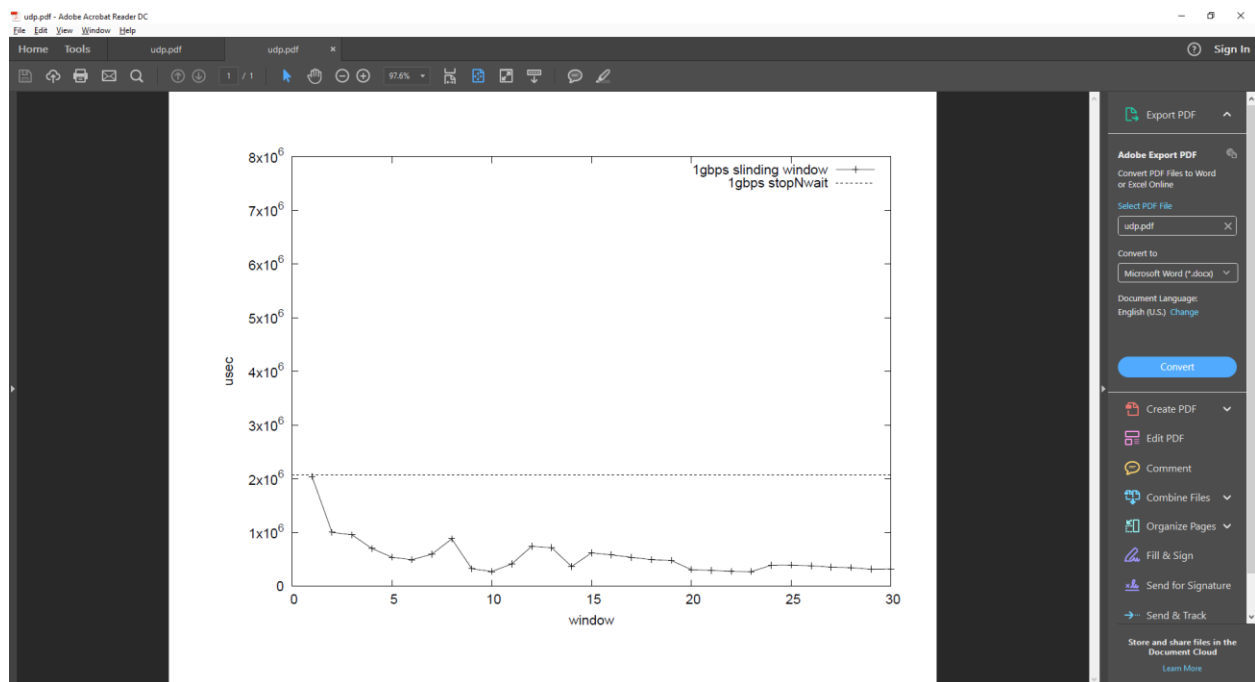


Figure 7 Delay time for case 2 and 3

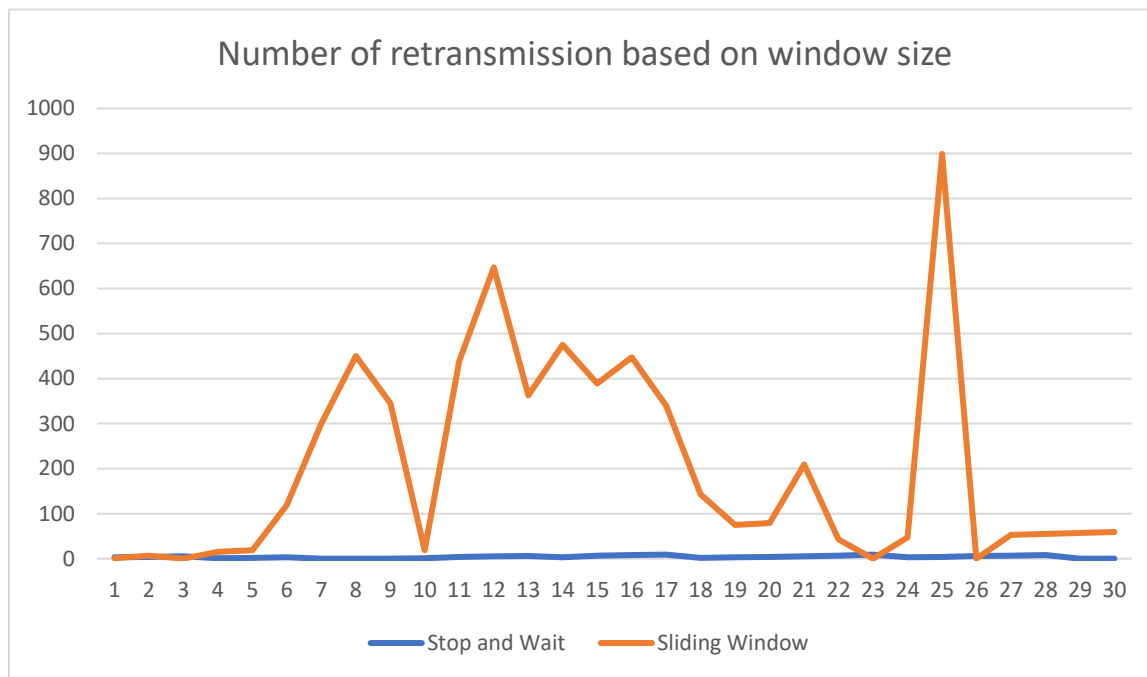


Figure 8 Retransmission for sliding window and stop and wait

Performance with random drops

Some how gnuplot does not like the data that I recorded and refused to create a pdf. So, I have to create a chart within Word

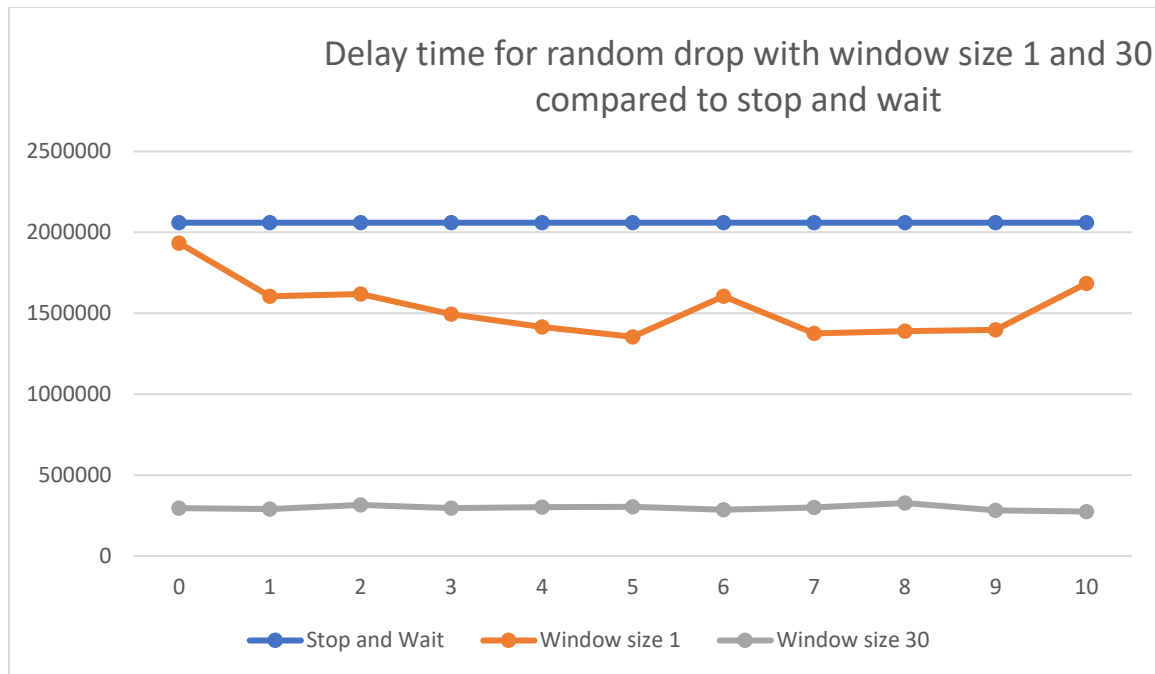
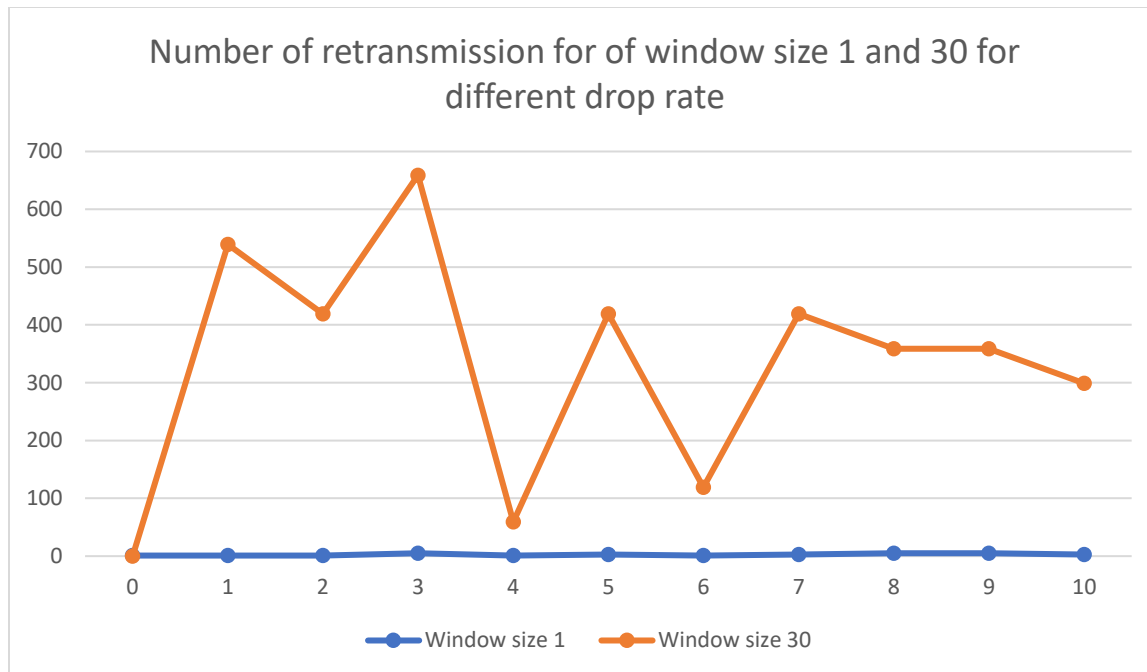


Figure 9 Delay time for case 4



Discussion

Stop and wait and sliding window

By looking at the graphs above for elapsed time, we can see that the elapsed time for case 2 is a straight line (Obviously because the program only run once for case 2). However, the interesting thing here is that for window size 1 of the sliding window, it has almost identical elapsed time like the stop and wait

Which make sense because for window size 1, the client can only send 1 package and have to wait for the ack before sliding the window to send the next one, which is identical to the behavior of stop and wait.

Also, as the window size increase, the elapsed time also decrease as well. Since now the client can send multiple packet at once before having to stop, this increases the throughput of the client side and server side. Making it quicker to send 20k packets.

Looking at the number of retransmitted message, the client stops and wait algorithm does not retransmit a lot, which is exactly what we are looking for because for each message, it will wait and receive the ack from that message before advancing

For the stop and wait, the number of retransmit message tend to go up and down, mostly up as the window size increase. That is because as the window size gets bigger, the number of messages the client can send get bigger. Also, the server has to make sure that it receives message in order, so it will ack

with the most recent number, this may cause some packages to be ignored. Hence the timer may time out on the client while waiting for the correct package ack. Leaving it to retransmit

Random drop

For the random drop algorithm, the one with the window size 30 clearly is quicker than the one with window size 1. For the same reason as discussed above.

There is quite an interesting pattern for the window size 1 as the drop rate increase. It looks like that the elapsed time decreased.

Which is contradicting to what I am expecting, I expect that as the drop rate increase, more packages will be dropped. Hence the client have to retransmit more, making the time longer

This is probably because of the randomness of network. Sometimes it may deliver the packets quick, some time it don't

Looking at the number of retransmission, The window size 1 also exhibit similar behavior like the stop and wait algorithm even with the effect of the drop rate. This makes sense because the client only send 1 package. If it get lost, the client only have to resend 1 package.

However, for the window size 30, it has the tendency to go up and down (mostly up) as the drop rate increase since if one package it lost, it will have to resend from the recent minimum package.

Again, the up and down are probably due to the randomness of network