Assignment 3 Report

Haram Kwon

Overview

Through this assignment, we are simulating TCP (Transmission Control Protocol) by UDP (User Diagram protocol). In order to implement the TCP cumulative acknowledgement, we first implement “stop and wait” protocol, and then based on “stop and wait” we implement “sliding windows”

Stop and Wait

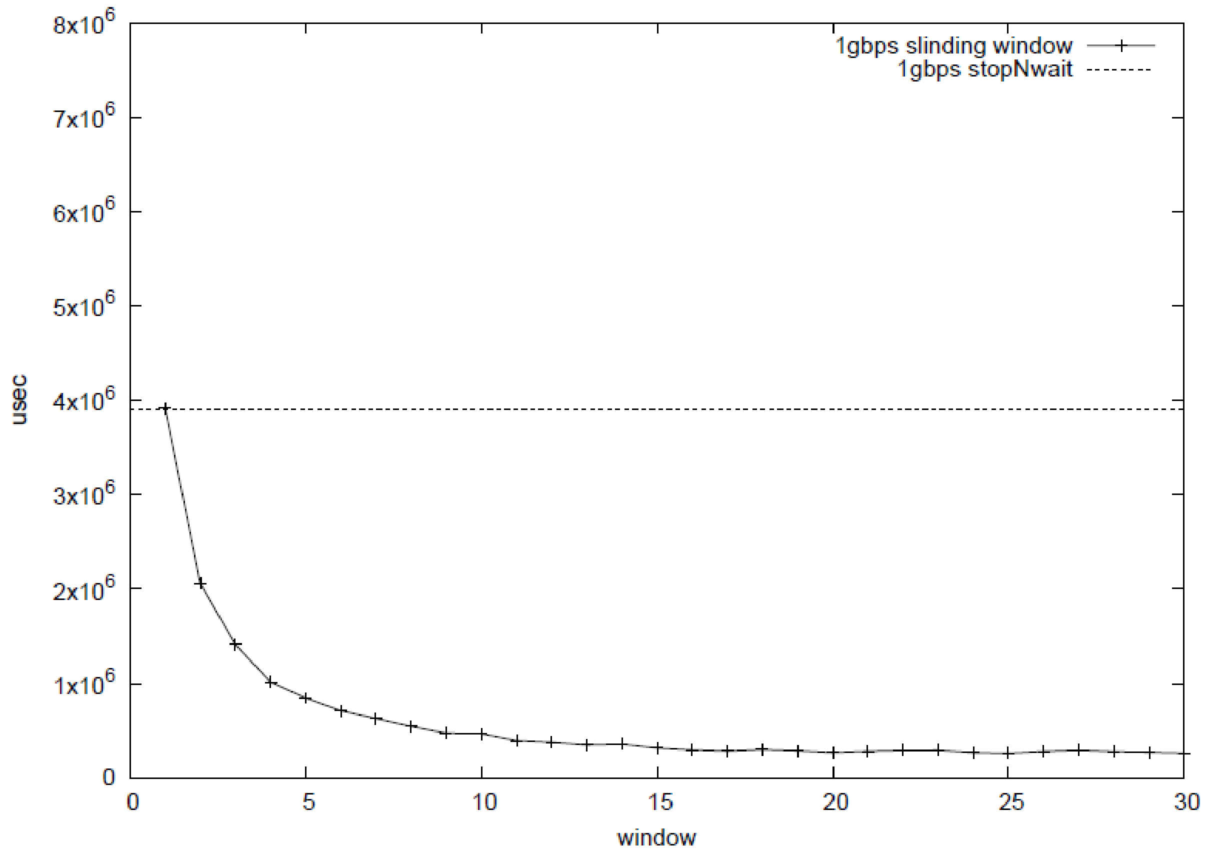
The stop and wait protocol guarantee the packet arrival and the packet acknowledge. There are two roles in this protocol: sender and receiver. At first the sender sends one packet to the receiver with sequence number and wait for the acknowledge from the receiver in a certain amount of time. (1500 us for our implementation) If there is acknowledgement from the receiver, the sender sends the next packet to the client. If there is no acknowledge from the receiver within the countdown, the sender sends the current packet to the receiver again.

For receiver, they wait the packet to be arrival, and acknowledge to the sender with the sender’s sequence number as a verification of getting the packet from the sender.

Sliding Windows

For the sliding windows protocol, both sender and the receiver have the same window size in order to communicate. Moreover, there are two ways to implement the sliding windows: cumulative repeat, and selective repeat. For the selective repeat, the sender selective sends packet where the sender does not get the acknowledge from the sender. For the cumulative acknowledgement, the sender sends the frame from where the receiver does not acknowledged. For this assignment, we are going to implement cumulative sender.

Performance Evaluation



Stop-and-wait performance over 1Gbps

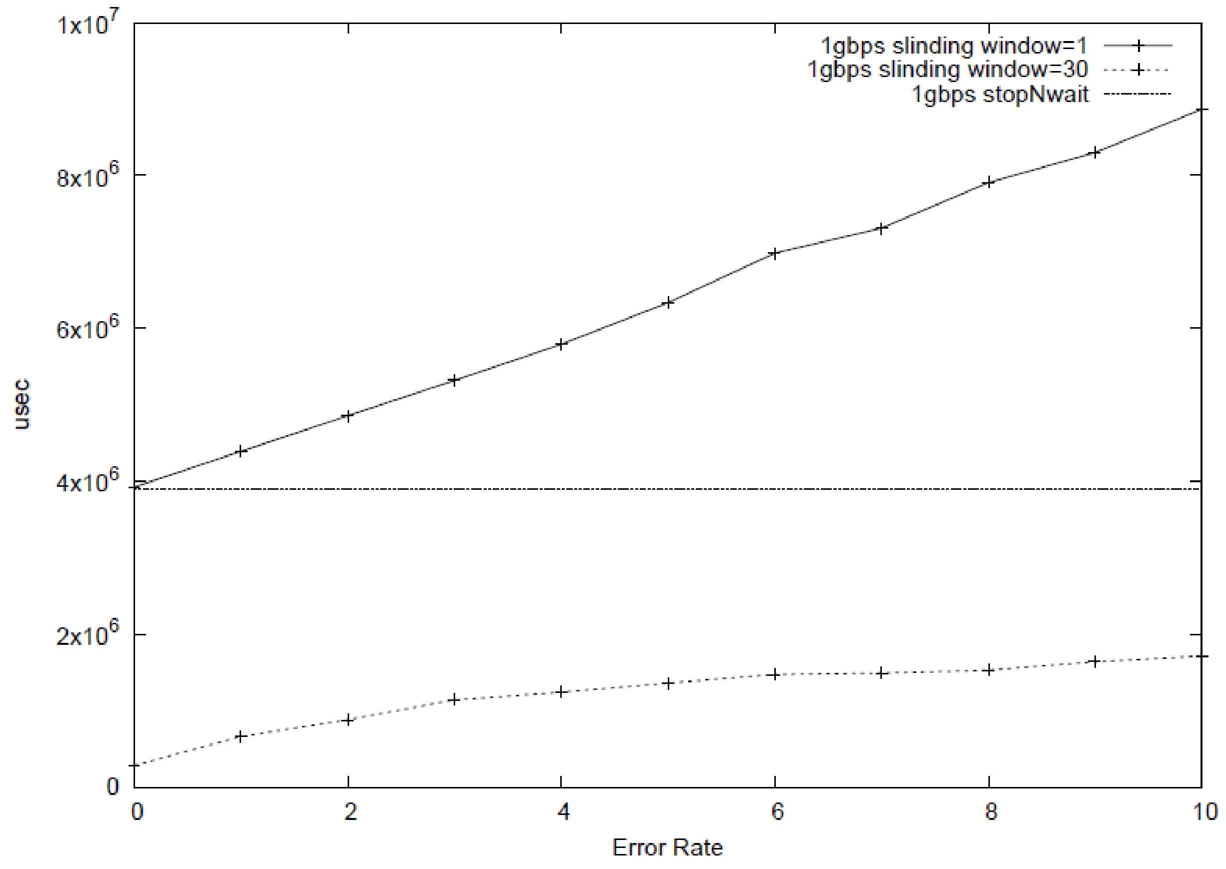
Since the sliding windows send the packet and wait for the acknowledgement from the receiver, the performance is equal to the sliding windows protocol with window size equal 1. It is the upper bound of the sliding windows protocol.

For the performance wise, it has the slowest performance, and it would increase linearly if there is package drop.

Sliding windows performance over 1Gbps

For the sliding windows, since it sends multiple packets at the same time, the performance gets better whenever the windows size gets bigger. The graph shows $\frac{1}{windows\_size} = time$ behavior first, and when it gets to some limit, there is no performance improvement. (Because of processing delay, queueing delay, transmission delay, and propagation delay)

Performance with random drops



As the random drop increases, the 1gbps sliding window with windows=1 protocol increase linearly. It implies that for the stop and wait protocol, as the error rate increases, the time that it need to retransmit the packet will increase linearly, For the sliding windows protocol with windows size=30, the graph seems logarithmic increase as the error rate increases. Based on this behavior, we can conclude that the sliding windows can bring the huge performance improvement even with noisy environment (or high drop rate environment.).

Discussion

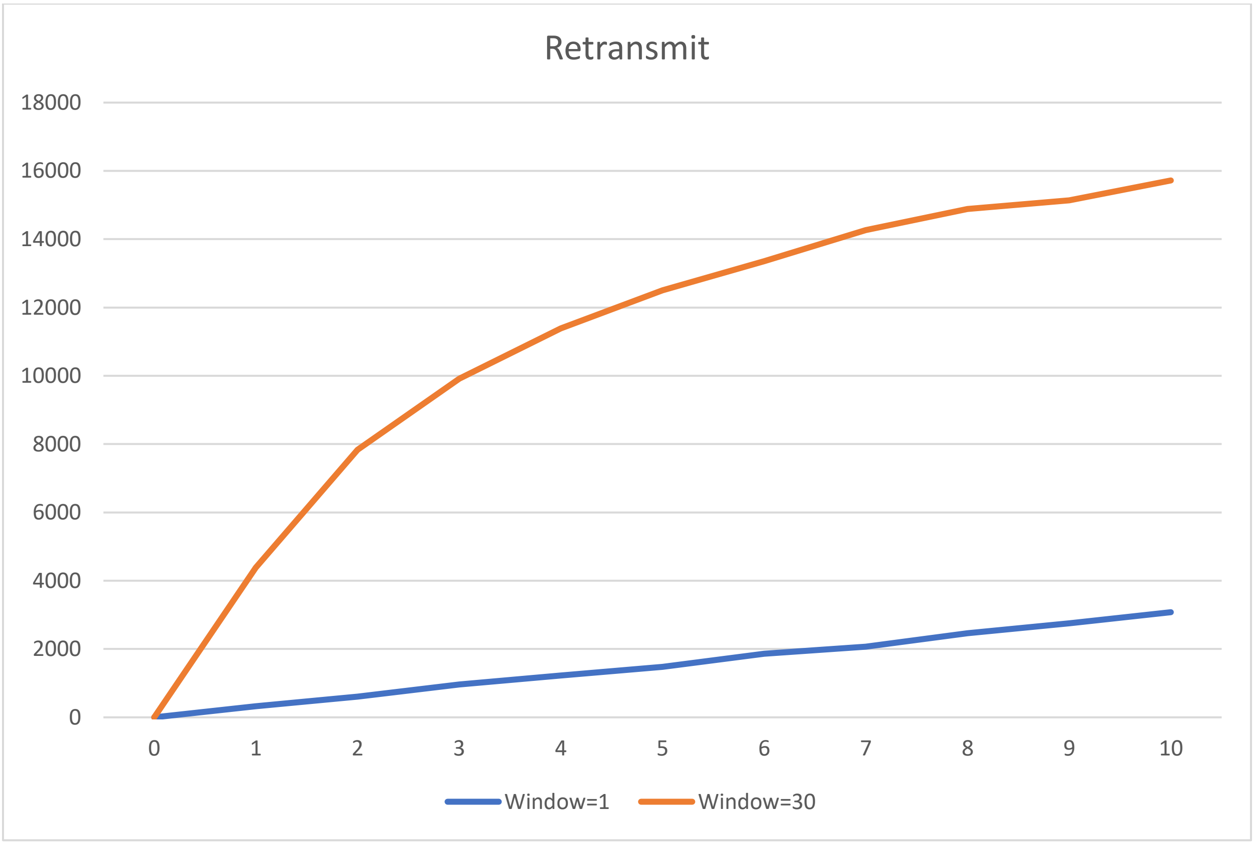
The difference in performance between stop-and-wait and sliding window Influence of window size on sliding window performance

The performance of the sliding-window gets better as the window size increases. When the window size is equal to the one, the performance is similar with stop and wait. (theoretically same) I notice that the performance of the sliding windows is:

$$Sliding\_Time = \frac{Stop\_wait}{Windows\_Size}$$

However, although the window size could be infinite, the sliding time does not approach to zero, because there is delay caused by router, link, and process.

Differences in the number of messages retransmitted between stop-and-wait and sliding window algorithms Include discussion of the effect of drop rates on both the window size of 1 and 30.



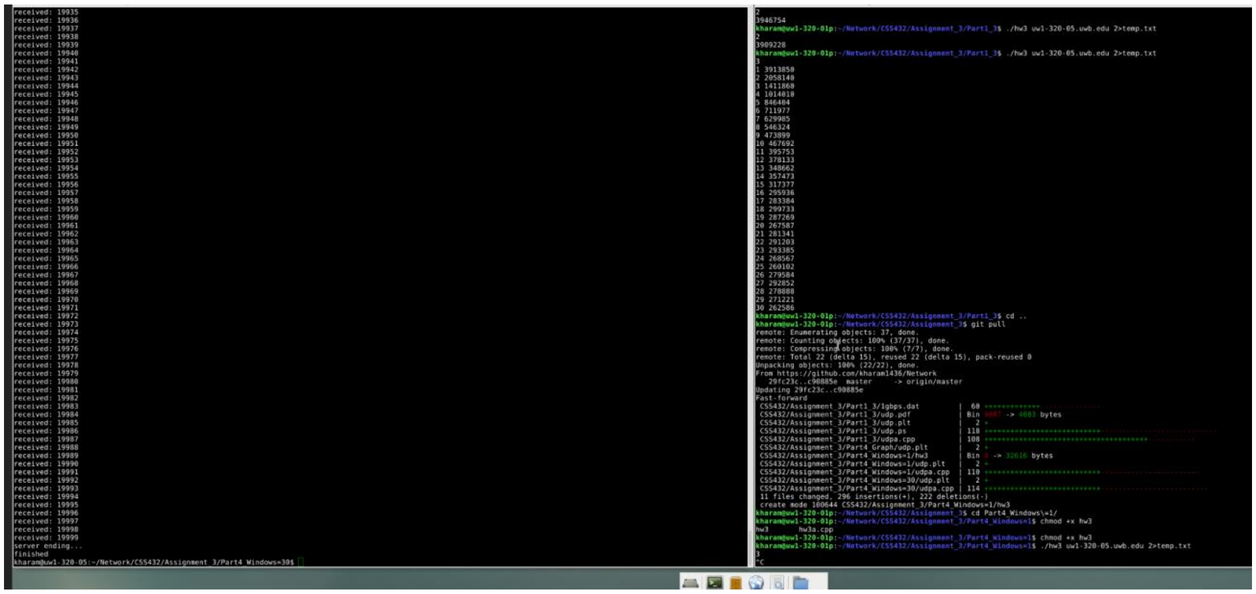
Based on the program run, we can get the following graph. As y-axis represent the number of retransmit, and x-axis represent the error rate. From the chart, we can notice that as the error rate goes up, the retransmit rate drastically increases when window=30, while window=1 protocol show the stop-

and-wait protocol, the number of retransmit is lower for stop and wait, and the number of retransmit with windows=30 is highly larger.

In case there no drop rate, I notice that the retransmit rate become also random (with time=100), and it was not consistent.

The output pictures

Server: UW1-320-05.uwb.edu (ubuntu 16.04)
Client: UW1-320-01p.uwb.edu (Debian 9.6)



```
received: 19935
received: 19936
received: 19937
received: 19938
received: 19939
received: 19940
received: 19941
received: 19942
received: 19943
received: 19944
received: 19945
received: 19946
received: 19947
received: 19948
received: 19949
received: 19950
received: 19951
received: 19952
received: 19953
received: 19954
received: 19955
received: 19956
received: 19957
received: 19958
received: 19959
received: 19960
received: 19961
received: 19962
received: 19963
received: 19964
received: 19965
received: 19966
received: 19967
received: 19968
received: 19969
received: 19970
received: 19971
received: 19972
received: 19973
received: 19974
received: 19975
received: 19976
received: 19977
received: 19978
received: 19979
received: 19980
received: 19981
received: 19982
received: 19983
received: 19984
received: 19985
received: 19986
received: 19987
received: 19988
received: 19989
received: 19990
received: 19991
received: 19992
received: 19993
received: 19994
received: 19995
received: 19996
received: 19997
received: 19998
received: 19999
server ending...
finished
kharam@uw1-320-05:~/Network/CSS432/Assignment_3/Part4_Windows=30$
```

```
2
3946754
kharam@uw1-320-01p:~/Network/CSS432/Assignment_3/Part1_3$ ./hw3 uw1-320-05.uwb.edu 2>temp.txt
2
3909228
kharam@uw1-320-01p:~/Network/CSS432/Assignment_3/Part1_3$ ./hw3 uw1-320-05.uwb.edu 2>temp.txt
3
1 3913850
2 2058140
3 1411868
4 1014018
5 846404
6 711977
7 629985
8 546324
9 473899
10 467692
11 395753
12 378133
13 348662
14 357473
15 317377
16 295936
17 283384
18 299733
19 287269
20 267587
21 281341
22 291203
23 293385
24 268567
25 260102
26 279584
27 292852
28 278888
29 271221
30 262586
kharam@uw1-320-01p:~/Network/CSS432/Assignment_3/Part1_3$ cd ..
kharam@uw1-320-01p:~/Network/CSS432/Assignment_3$ git pull
remote: Enumerating objects: 37, done.
remote: Counting objects: 100% (37/37), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 22 (delta 15), reused 22 (delta 15), pack-reused 0
Unpacking objects: 100% (22/22), done.
From https://github.com/kharam1436/Network
   29fc23c..c90885e  master     -> origin/master
Updating 29fc23c..c90885e
Fast-forward
 CSS432/Assignment_3/Part1_3/1gbps.dat          |  60 +++++++++++++-------------
 CSS432/Assignment_3/Part1_3/udp.pdf            | Bin 4087 -> 4083 bytes
 CSS432/Assignment_3/Part1_3/udp.plt            |   2 +-
 CSS432/Assignment_3/Part1_3/udp.ps             | 118 +++++++++++++++++++++++++++---------------------------
 CSS432/Assignment_3/Part1_3/udpa.cpp           | 108 ++++++++++++++++++++++++++++++++++++-----------
 CSS432/Assignment_3/Part4_Graph/udp.plt        |   2 +-
 CSS432/Assignment_3/Part4_Windows=1/hw3        | Bin 0 -> 32616 bytes
 CSS432/Assignment_3/Part4_Windows=1/udp.plt    |   2 +-
 CSS432/Assignment_3/Part4_Windows=1/udpa.cpp   | 110 +++++++++++++++++++++++++++-----------------------
 CSS432/Assignment_3/Part4_Windows=30/udp.plt   |   2 +-
 CSS432/Assignment_3/Part4_Windows=30/udpa.cpp  | 114 +++++++++++++++++++++++++++---------------------------
 11 files changed, 296 insertions(+), 222 deletions(-)
 create mode 100644 CSS432/Assignment_3/Part4_Windows=1/hw3
kharam@uw1-320-01p:~/Network/CSS432/Assignment_3$ cd Part4_Windows\=1/
kharam@uw1-320-01p:~/Network/CSS432/Assignment_3/Part4_Windows=1$ chmod +x hw3
hw3      hw3a.cpp
kharam@uw1-320-01p:~/Network/CSS432/Assignment_3/Part4_Windows=1$ chmod +x hw3
kharam@uw1-320-01p:~/Network/CSS432/Assignment_3/Part4_Windows=1$ ./hw3 uw1-320-05.uwb.edu 2>temp.txt
3
^C
```