Lab #2

Reliable Data Transfer Assignment

Phase #2

Selective repeat, go back N

and congestion control

Names:

Ahmed Shawky (10)

Omar Mohamed Eissa

Youssef youssry Nasr

Problem Statement:

Specifications

Suppose you've a file and you want to send this file from one side to the other(server to client), you will need to split the file into chunks of data of fixed length and add the data of one chunk to a UDP datagram packet in the data field of the packet using selective repeat.

At any given time, the server is allowed to have up to N datagrams that have not yet been acknowledged by the client. The server has to be able to buffer up to N datagrams, since it should be able to retransmit them until they get acknowledged by the client. Moreover, the server has to associate a timer with each datagram transmitted, so that it can retransmit a datagram if it is not acknowledged in time.

Congestion control:

In this approach: the sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs. You are asked to implement a dynamic window size that is changed based on the congestion of the network, where a packet loss happens after N0 packets sent, then you will send another N1 packets then another loss happens, and so on.

1. ***Selective repeat***:

Implementation:

Selective Repeat Implementation:
Using 2 threads, the first one for sending the data packets and the second one for receiving the ACK packet and a blocking queue to block the consumer thread (receiver) when there are no elements and blocking the sender thread when the queue is full. Also to handle the timeout for each packet, there's a timer thread which is responsible to handle any timeout for any packet.

Client side:

- main functions:

```
-public SRClientSide(String serverIp, short serverPort, short port, int wndSize, long seed, double plp)
```

Constructor used to initialize server ip, server port number, Client port number, window size and loss percentage

```
-private void receiverWork(String fileName) throws IOException
```

This function is responsible for receiving the packets that are sent by the server till the file is completely sent by the server. It also sends ACKs to server when it receives a packet successfully.

Server side:

Main functions:

```
public SRServerSide(byte[] receiveData, int size, double plp,
long timeOut, InetAddress clientIP, int wndSz)
```

Constructor used to initialize client ip address, window size, loss percentage and timeout

```
private void senderWork() throws IOException,
InterruptedException
```

Function used to send read the file that the user has requested chunk by chunk (500 bytes each) and send them to the client. It stops when the entire requested file is sent.

```
private class ReceiverWork implements Runnable
```

Class used to accept the acks from the client and move the sending window. It stops when the entire file is sent to the client.

2. Congestion control:

- Whenever timeout happens the Cwnd is updated so that it gets back to 1 and ssthreshold is updated to: Old Cwnd / 2 and we get to slow start state.
- When Ack is received, there are two cases:
    1- If we are in the slow start state then Cwnd will be Old Cwnd + 1
    2- If we are in the congestion avoidance state then Cwnd will be Old Cwnd + 1/ Old Cwnd
- We can get from slow start state to congestion avoidance state by exceeding the ssthreshold.
- We can get back from congestion avoidance state to slow state by an occurrence of timeout.

1. ***Go back N:***

   Go back N is another strategy of sending files form server to client.
   It first sends all the packets within the server sending's window then wait for ACK for the first packet in the sending window.
   If the ACK reaches before timeout, then the window is moved to the next unacknowledged element.
   If the ACK doesn't reach before the timeout, then the whole window is resent to the client.
   On the other side, the client is waiting for a certain packet, when it arrives, the client sends ACK otherwise, it doesn't send ACKs.

Server side:

Main functions:

```
public GBNServerSide(byte[] receiveData, int size, double plp,
long timeOut, InetAddress clientIP, int wndSz)
```

Constructor used to initialize client ip address, window size, loss percentage and timeout.

```
private void reintializeTimerTask()
```

Function used to schedule the actions required when timeout happens.

```
private void resend() throws IOException
```

Function used to resend all the packets in the window when timeout takes place.

```
private void senderWork() throws IOException,
InterruptedException
```

Function used to send read the file that the user has requested chunk by chunk (500 bytes each) and send them to the client. It stops when the entire requested file is sent.

```
private class ReceiverWork implements Runnable
```

Class used to accept the ACKs from the client and move the sending window. It stops when the entire file is sent to the client.

-

4. Performance analysis:

-File size: 5.28 Mbs

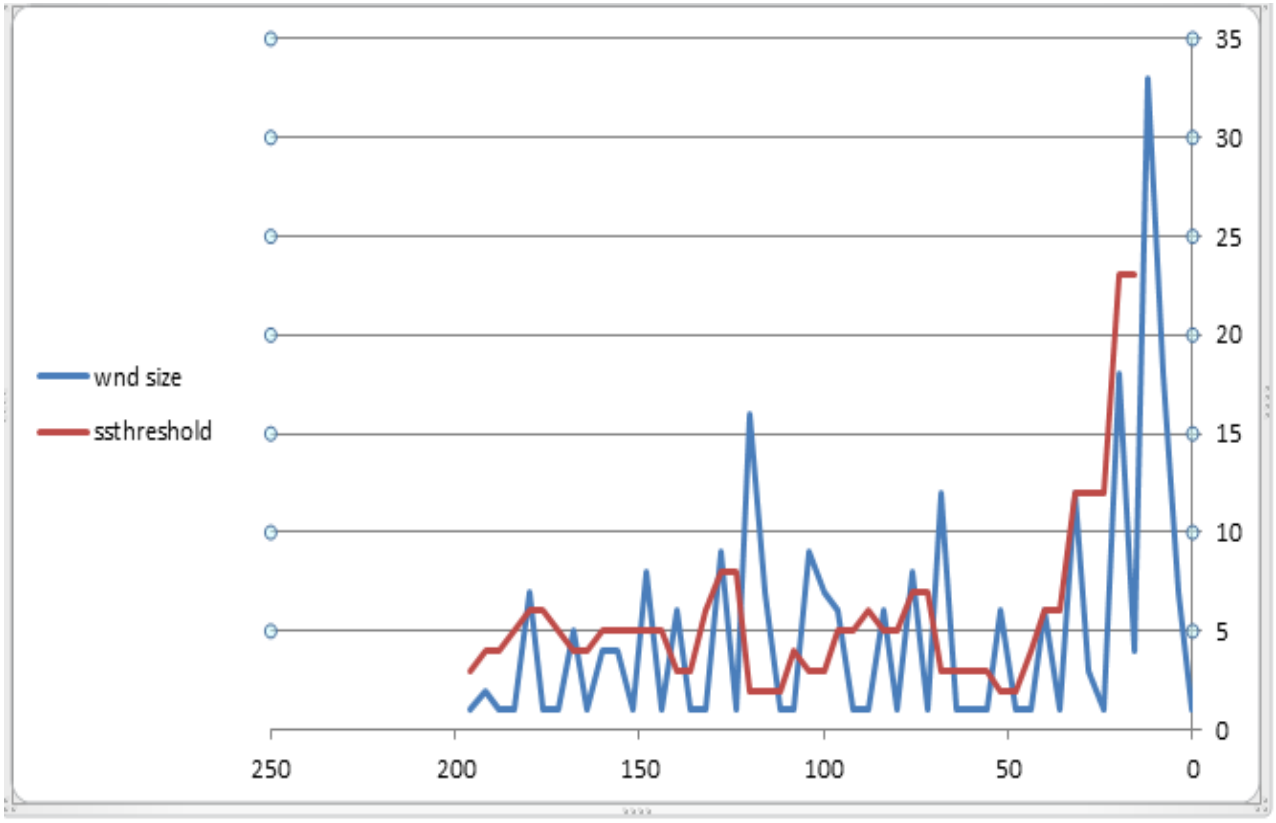
At timeout 20 msec

At window size: 1000

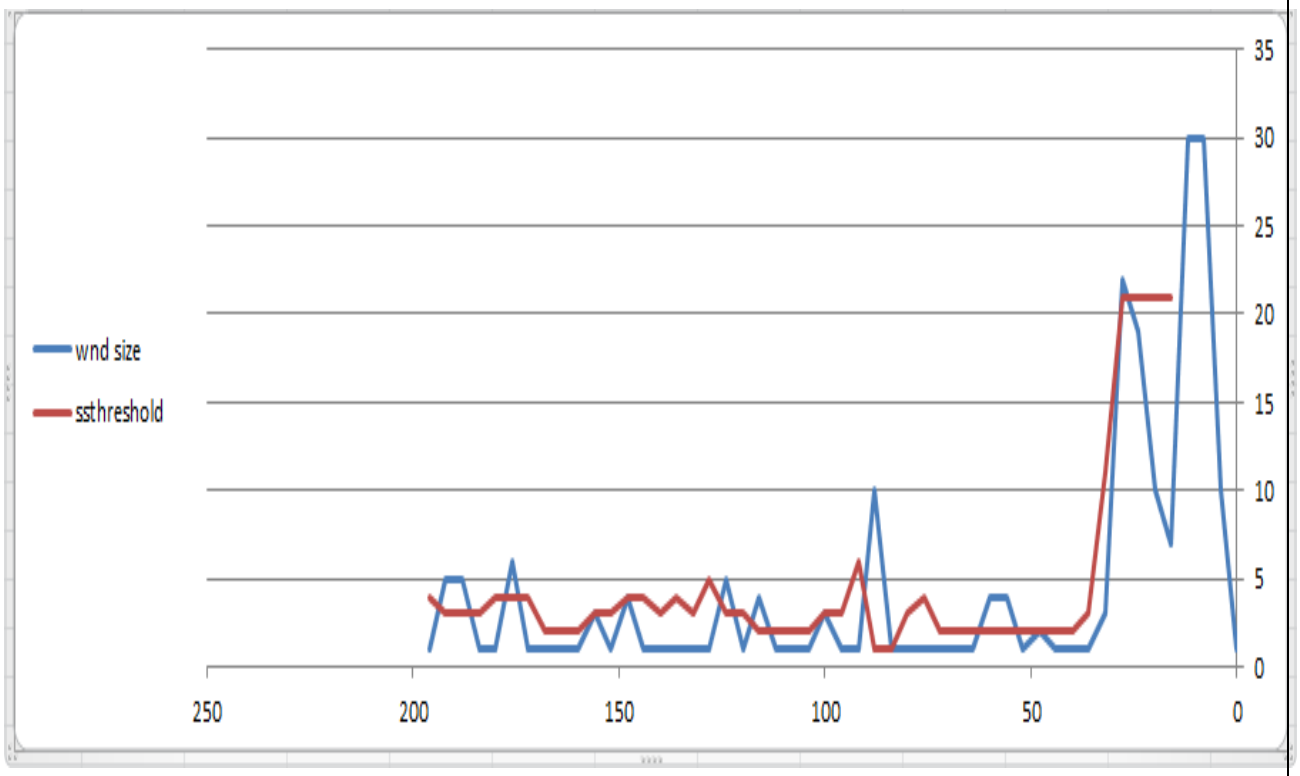
Lose %	1st trial Time(ms)-resend	2nd trial Time(ms)-resend	3rd trial Time(ms)-resend	4th trial Time(ms)-resend	5th trial Time(ms)-resend	AVG.
0	1085 ms-210	840 ms-190	800 ms-182	784 ms-177	804 ms-185	862 ms-188
1	2451ms-112	2382ms-153	2736ms-150	2461ms-134	2078 ms-121	2421 ms-134
5	8978ms-547	7894ms-528	8275ms-524	8381ms-535	7866ms-508	8278 ms-528
10	15797ms-1087	14801ms-1051	14776ms-1023	16003ms-1125	16210ms-1167	15517 ms-1090
30	51242ms-3169	51159ms-3096	51122ms-3027	51534ms-3188	51560ms-3198	51323 ms-3135

Congestion control graphs (time vs window sizes):

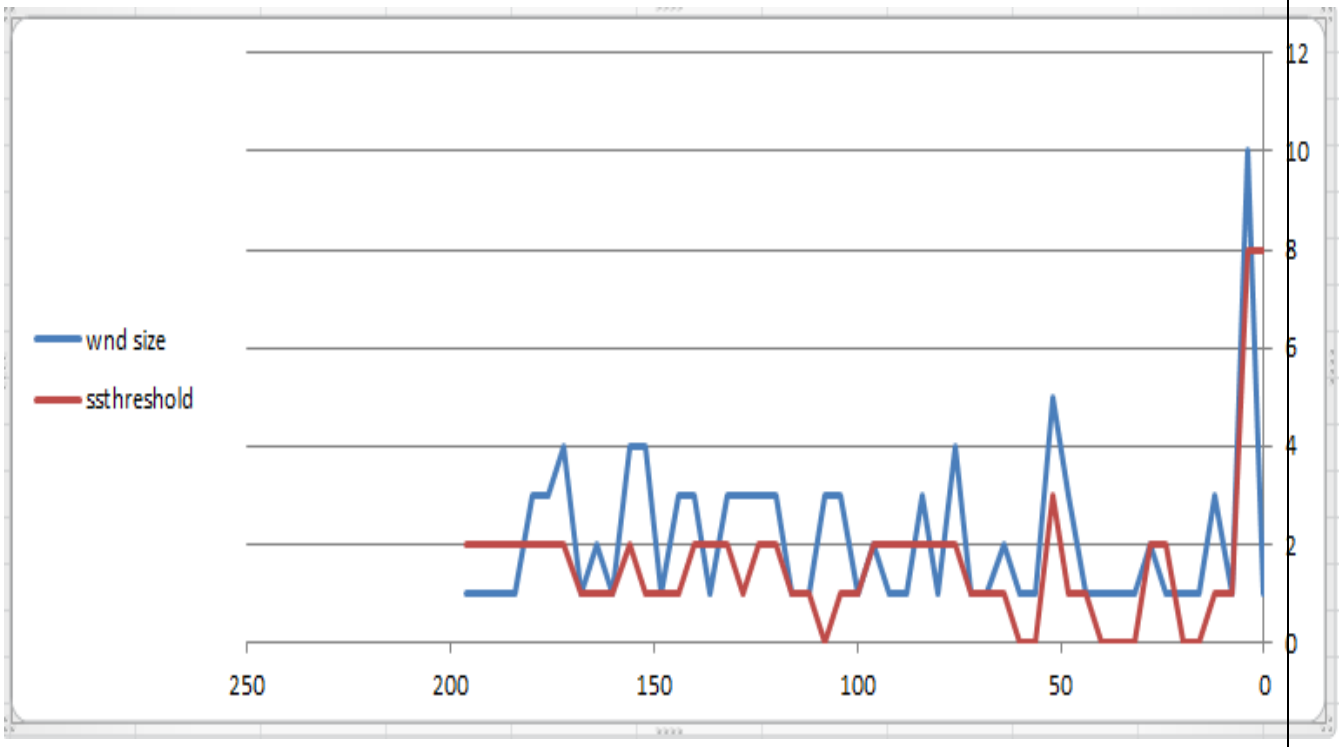
-***For 5% loss***



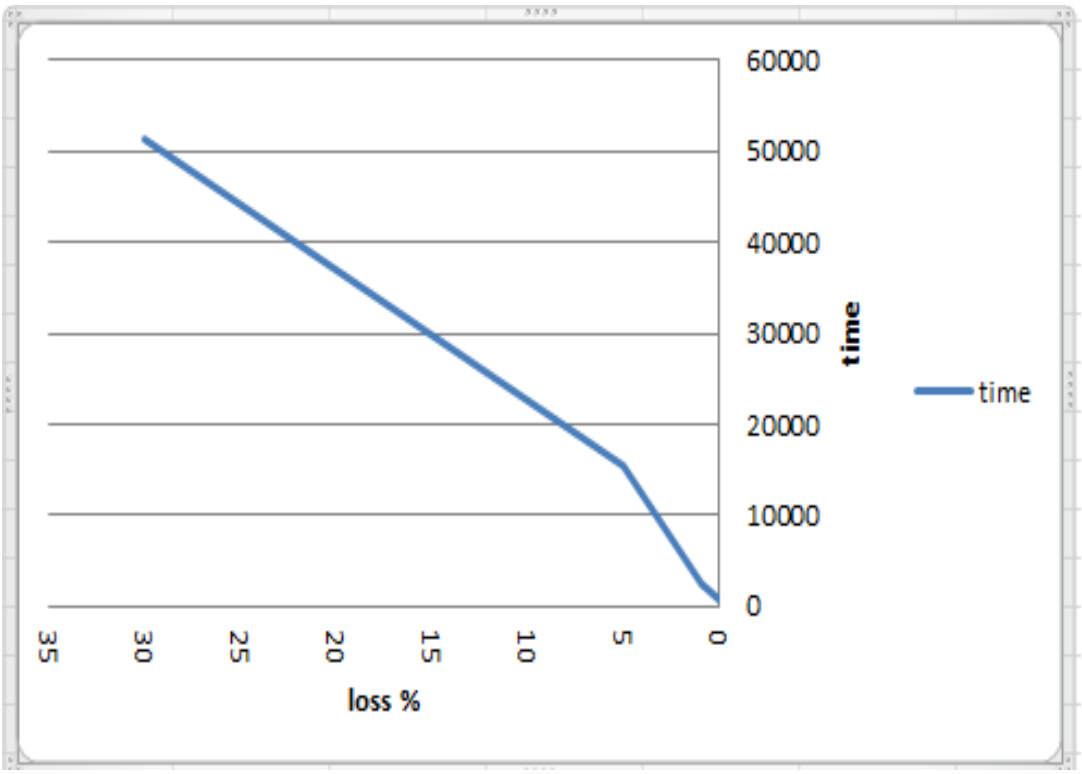
For 10% loss:



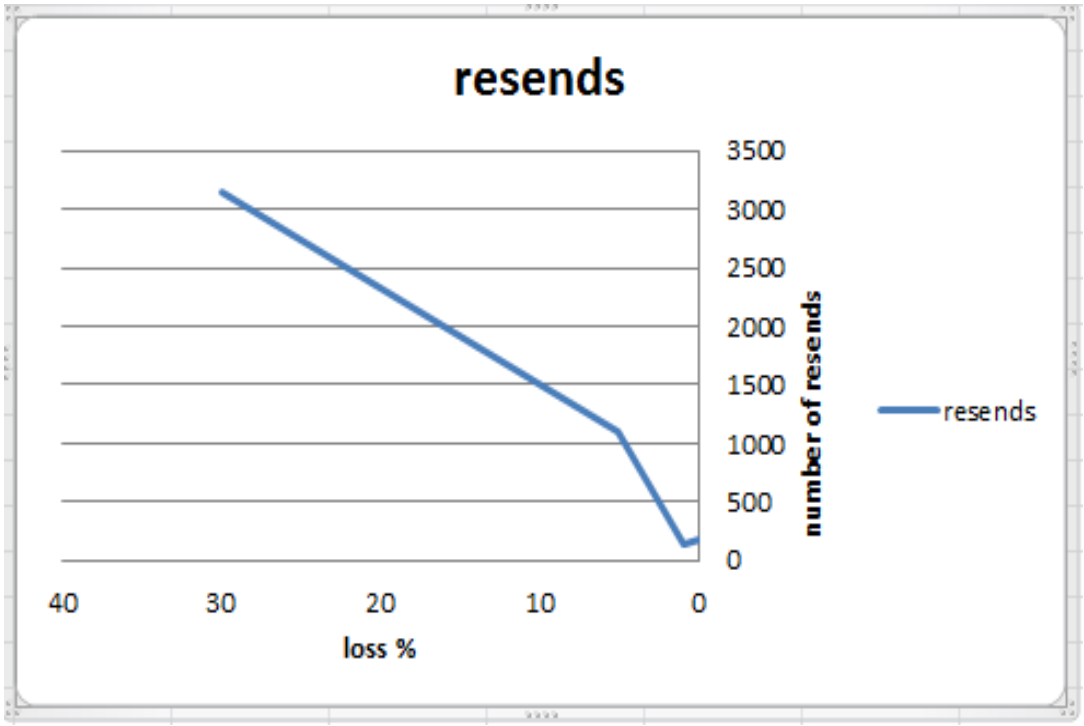
For 30% loss:



- Loss % vs time for sending :



- Loss % vs number of resends



-File size: 5.28 Mbs

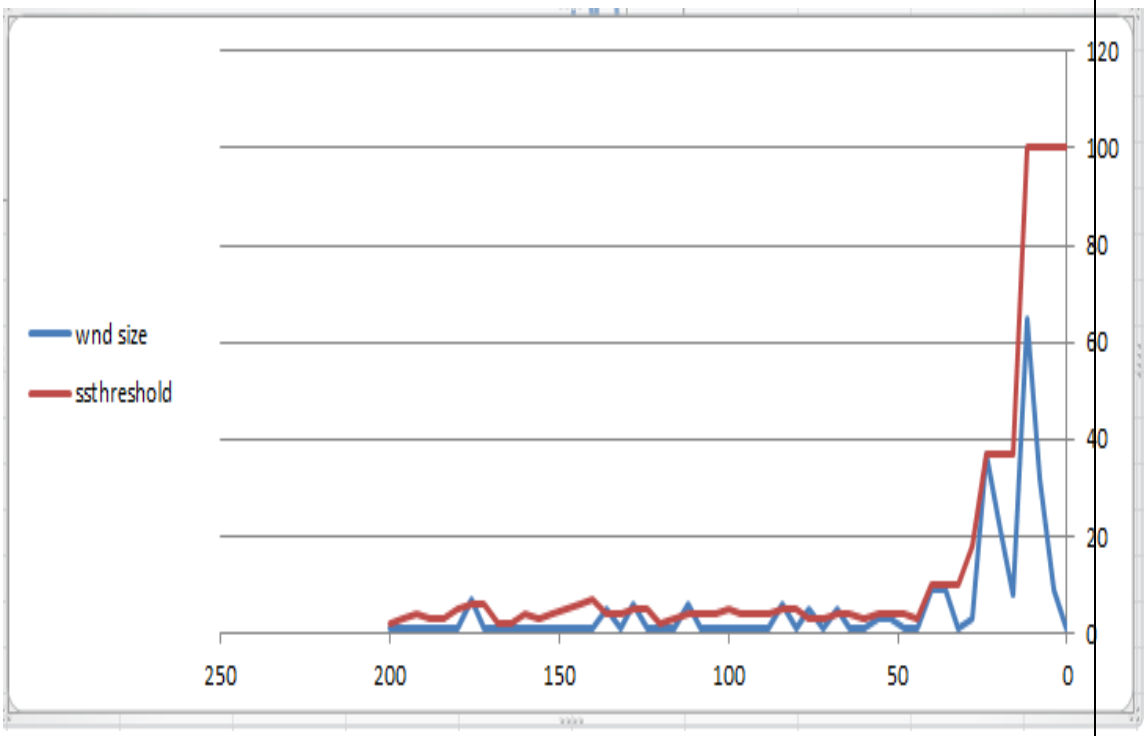
At timeout 20 msec

At window size: 100

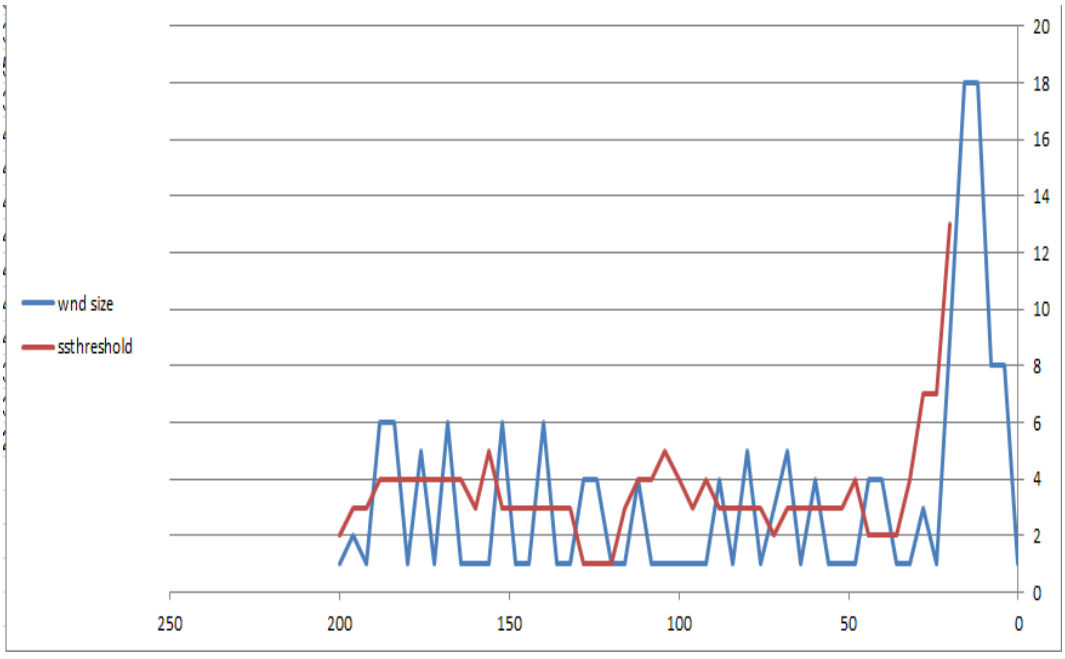
Lose %	1st trial Time(ms)-resend	2nd trial Time(ms)-resend	3rd trial Time(ms)-resend	4th trial Time(ms)-resend	5th trial Time(ms)-resend	AVG.
0	885 ms-81	877 ms-79	800 ms-65	898 ms-85	893 ms-78	872 ms-77
1	2607ms-99	2484ms-108	2380ms-94	2451ms-104	2403ms-95	2465 ms-100
5	9277ms-547	7985ms-528	8105ms-537	8079ms-567	8630ms-508	8415 ms-537
10	16158ms-1087	15058ms-1022	14776ms-1013	16013ms-1113	16110ms-1127	15623 ms-1072
30	51311ms-3169	50697ms-3096	51152ms-3137	51534ms-3188	51450ms-3124	51228 ms-3142

Congestion control:

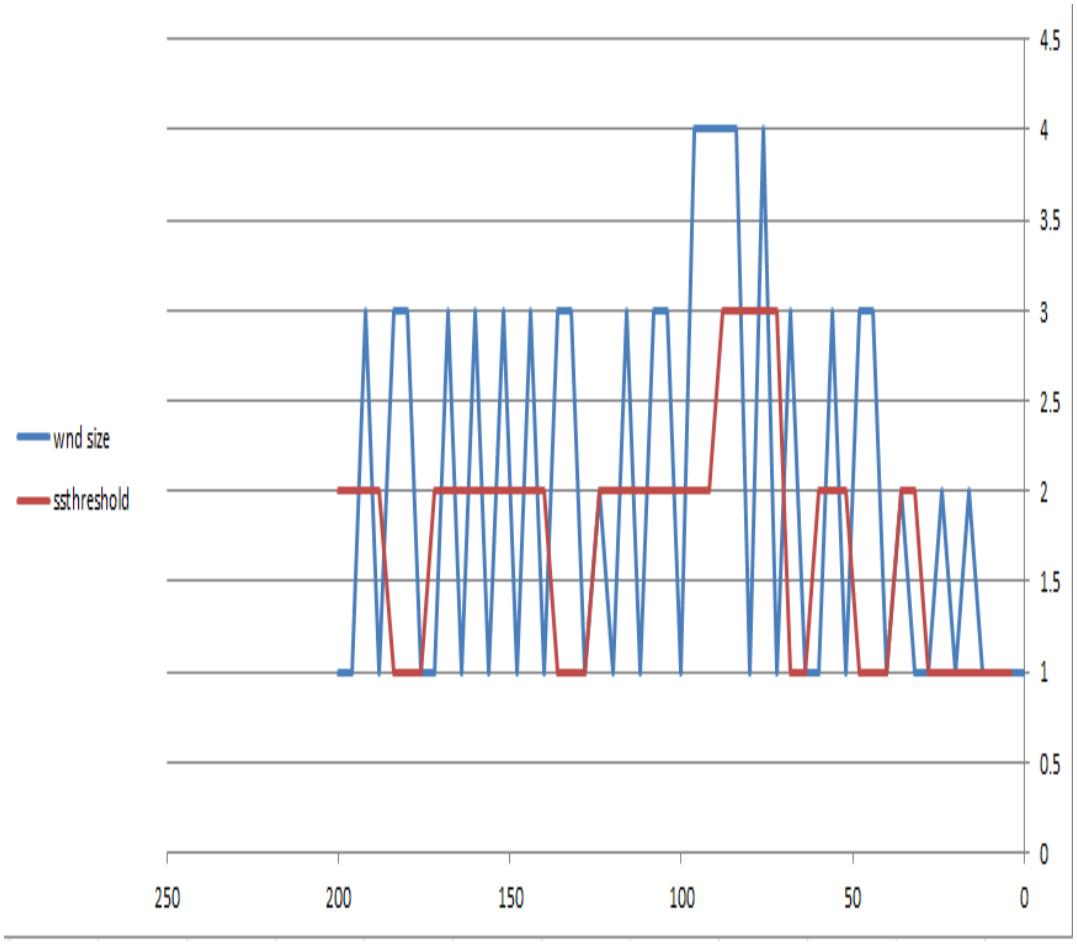
- 5% loss:



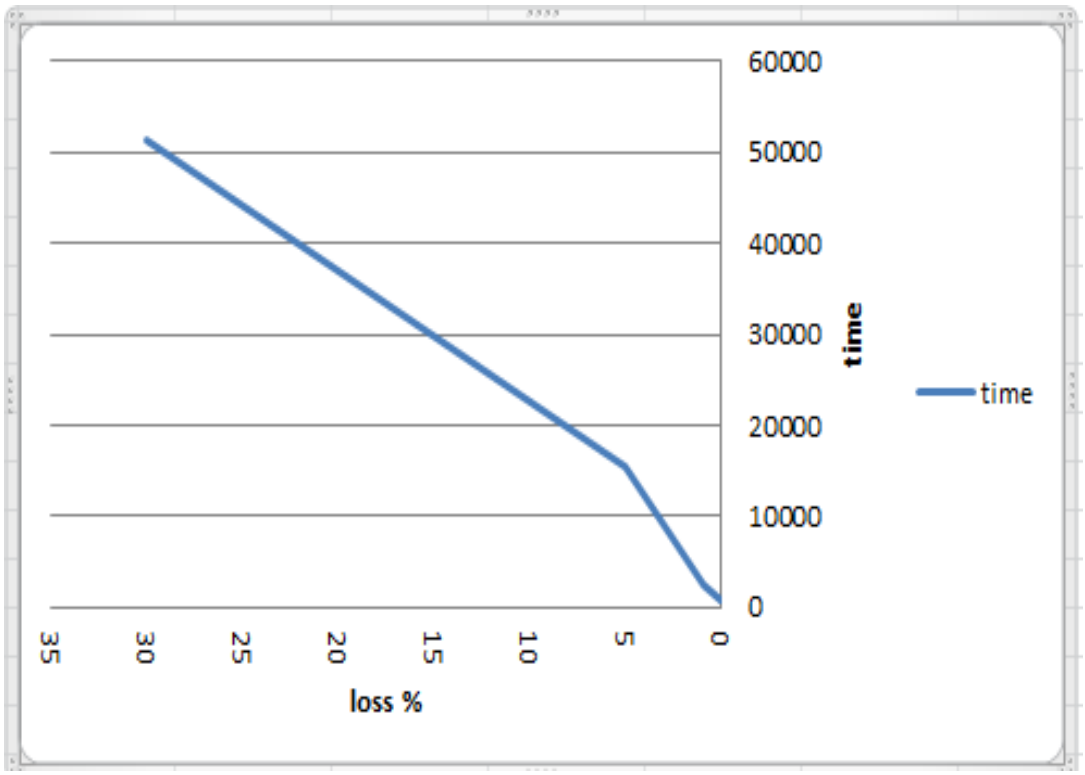
- 10% loss:



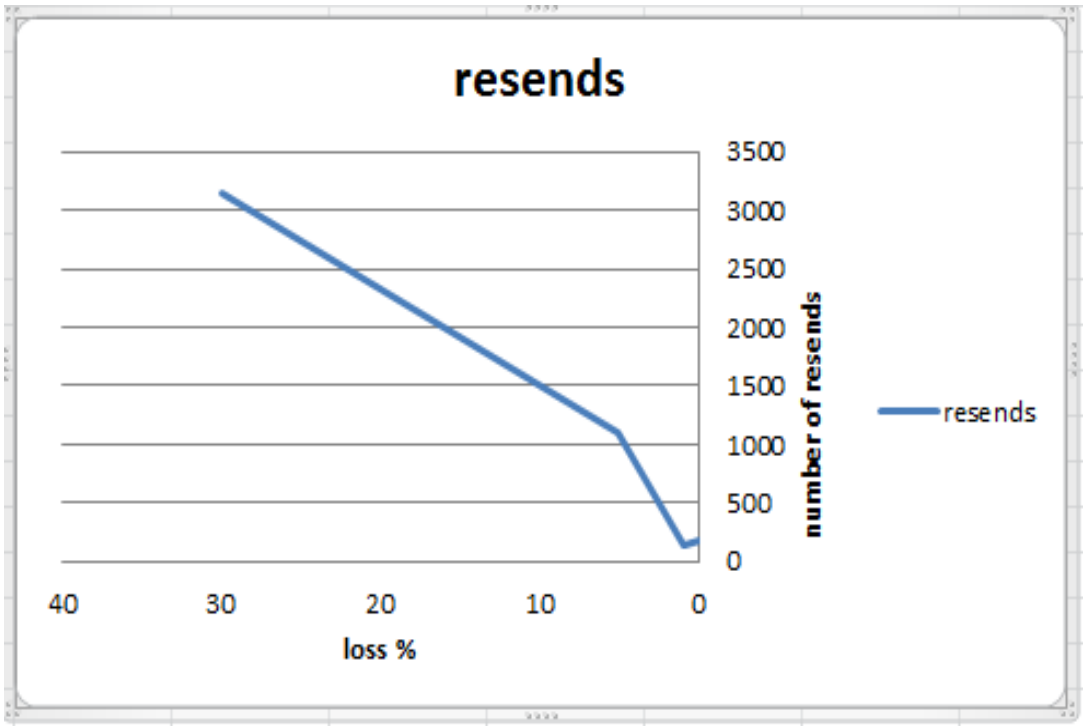
30% loss:


wnd size
ssthreshold
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
250
200
150
100
50
0

- Loss % vs time:

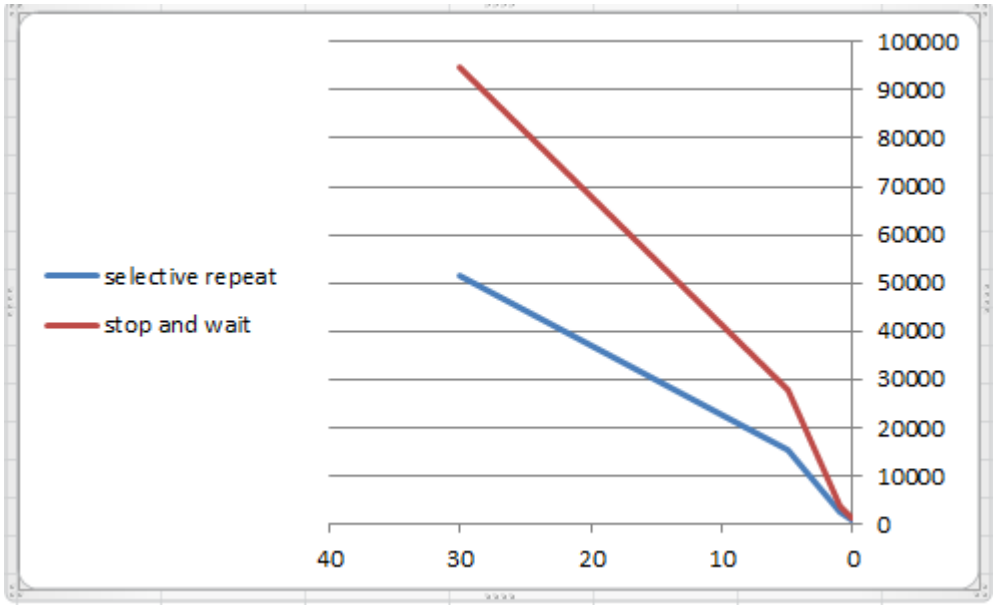

60000
50000
40000
30000
20000
10000
0
time
time
35
30
25
20
15
10
5
0
loss %

- Loss% vs number of resends

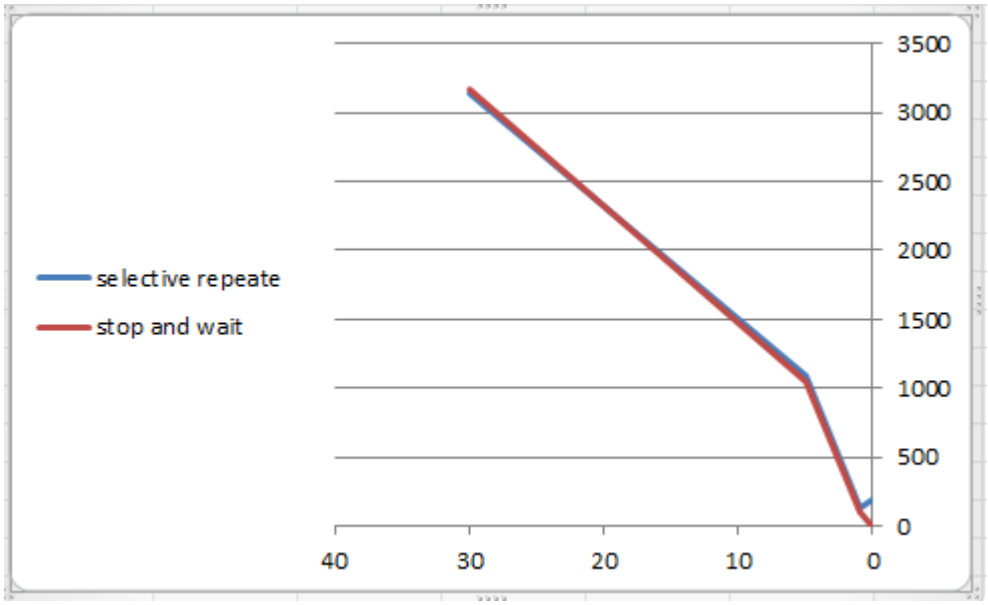


- Comparison between performance of (stop and wait) and selective repeat:

- Loss % vs time required:



- Loss% vs number of resends:



- ***Go back N time analysis***

File size: 5.28 Mbs

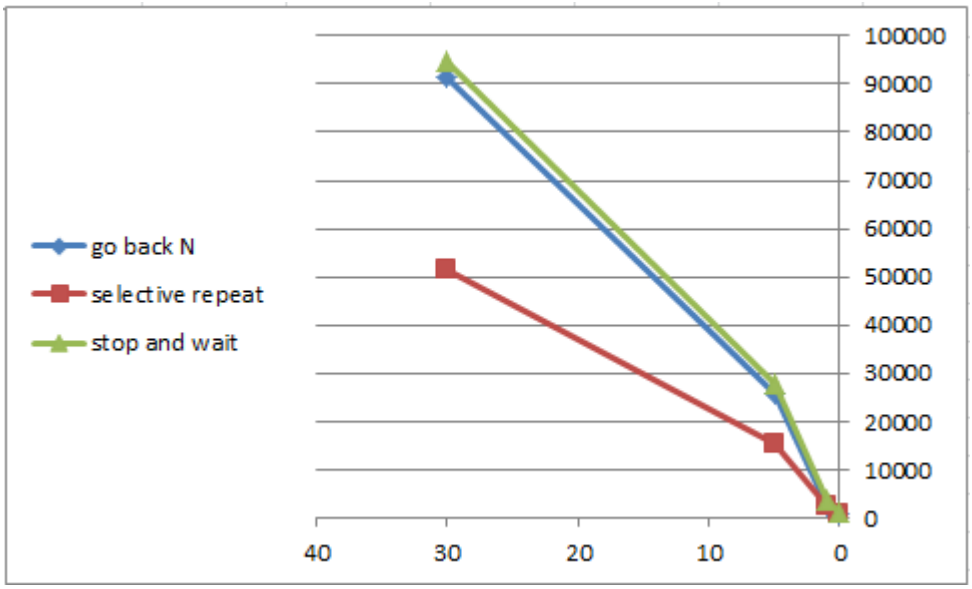
At timeout 20 msec

At window size: 100

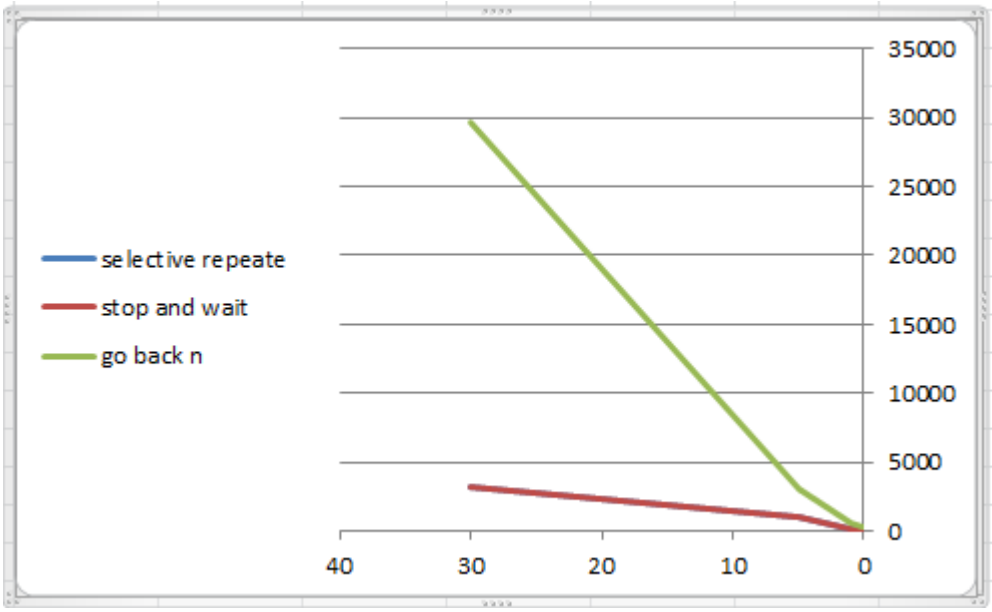
Lose %	1st trial Time(ms)-resend	2nd trial Time(ms)-resend	3rd trial Time(ms)-resend	4th trial Time(ms)-resend	5th trial Time(ms)-resend	AVG.
0	885 ms-81	877 ms-79	800 ms-65	898 ms-85	893 ms-78	872 ms-250
1	2607ms-99	2484ms-108	2380ms-94	2451ms-104	2403ms-95	2465 ms-523
10	25143ms-2993	25197ms-3051	26584ms-3092	26013ms-3017	25930ms-3003	25773 ms-3031
30	92543ms-319096	90575ms-295017	91152ms-297632	91534ms-280423	91725ms-290153	91505 ms-296464

- Comparison between (selective repeat) , (gobackN) and (stop and wait):

Loss % vs Time (ms)



Loss% vs number of resends



- ***Comments:***

- Selective repeat has the best performance in terms of time required to send the data, then GOBACKN and finally comes the stop and wait strategy.
- GOBACKN has a very high resending rates as it resends the all the packets inside the window if a timeout takes place, so it consumes a lot of network resources.
- Changing the window size hasn't led to significant change in performance with respect to time needed to send the data.