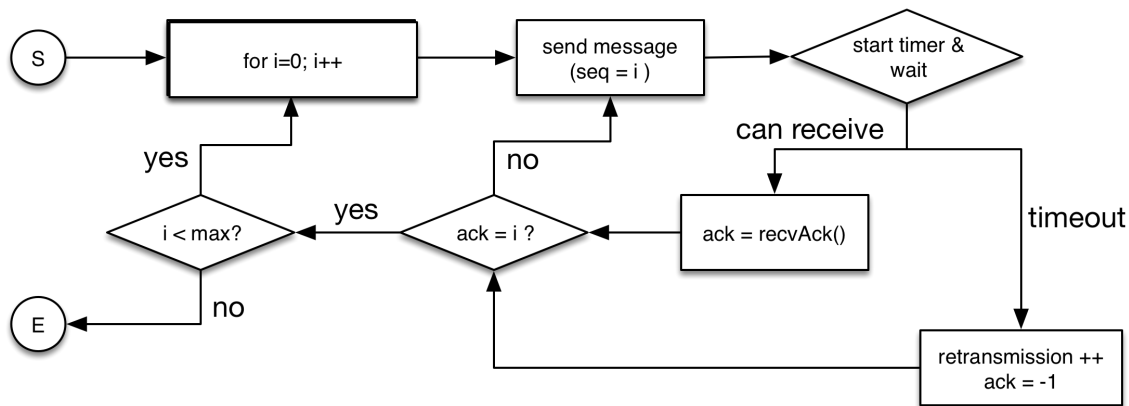


Program 3 - Sliding Window

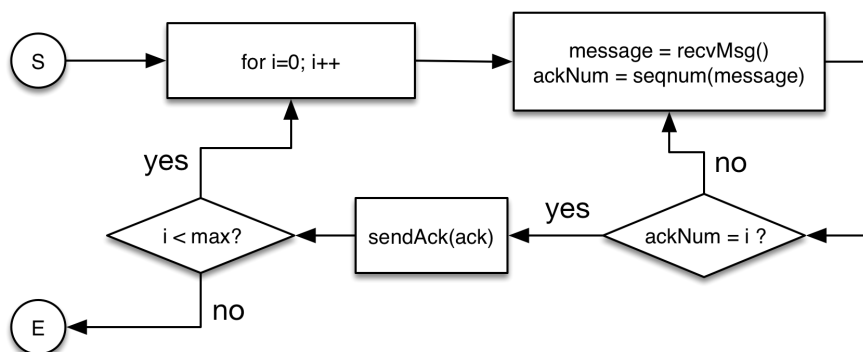
Stop & Wait

Stop & wait is implemented in a fairly straightforward fashion. They use for-loops that wait for acknowledgement of the current packets before incrementing i . The client will constantly send i until it receives an ACK for i . The server will constantly receive, and ignore any numbers which are not i . Once the loop finishes, the transmission has been successful and the method exits, reporting the number of retransmissions performed.

Client



Server

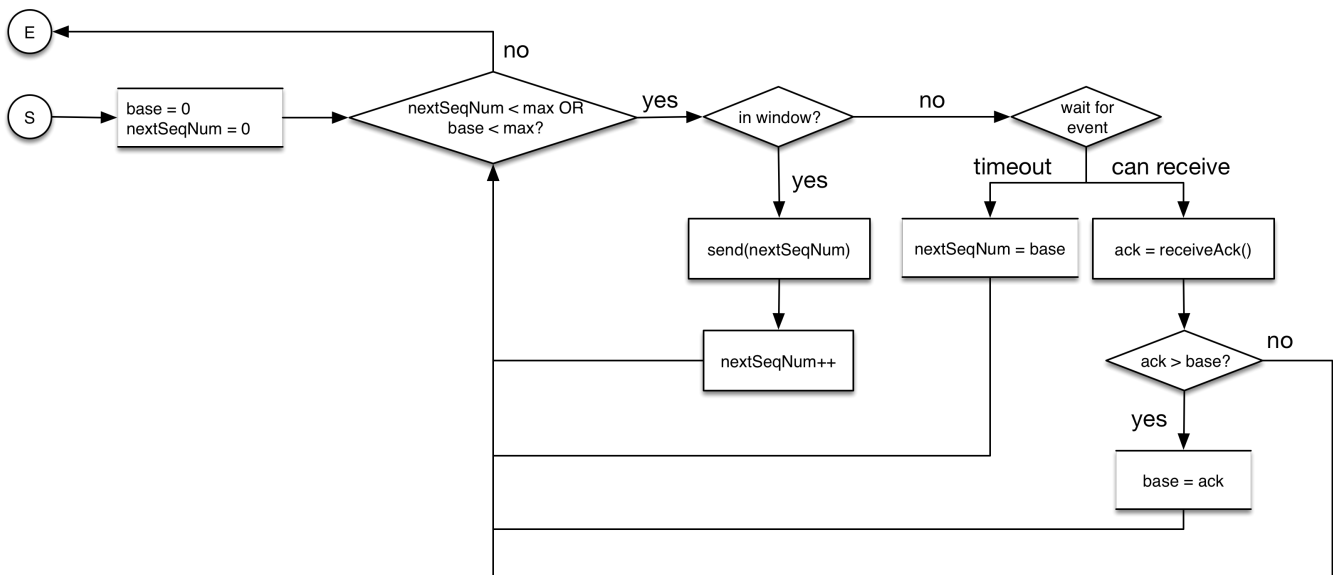


Sliding Window

The Sliding window has a much more complex state than the stop & wait algorithm, as it is a generalization to introduce an arbitrary window size.

Client

Client: Sliding Window



The client maintains state regarding the start of the window, and the next sequence number to send. It then loops while both the window start and sequence number are less than the max. While looping there are two states the algorithm can be in: inside or outside the window.

If inside the window, the client continually send packets, increasing it's sequence number until it hits the end of the window.

Outside Window

If we are outside the window, the programs main intent is to now either move the window so that transmission of packets can continue, or move the sequence number back to retransmit packets that were not received. A timer is started and a while loop traps code execution until an event

occurs (canReceive OR timeout). When an event occurs, the loop exits, and we hit the event logic below.

If the client can receive an acknowledgement, it does. If the acknowledgement can move the window forward (the ACK is greater than the current window base), the window base is moved to the sequence number acknowledge and movement is recorded.

The timeout event is then checked. if nothing was received or the window was not moved, this is considered a timeout. Upon timeout, the next sequence number is set back to the window base, so that retransmission of the window can occur.

Retransmission counting

The client keeps a boolean array (called “sent”) with an index for every sequence number to be transmitted. When it is time for the sequence number to be transmitted, the client first checks if sent[sequence number] is true. If it is, the retransmission counter is incremented. The client then proceeds to set sent[sequence number] to true and transmit the message.

Server: Early Retransmit

The server maintains state regarding which sequence numbers have been received, and what the next expected sequence number is. This is accomplished using two while loops and a boolean array.

Recording Received Sequence Numbers

A boolean array is maintained which has an index corresponding to each sequence number in the set of 20000. When a message is received, the index for the sequence number of the message is set to true. This happens for all messages, received, regardless of they are the expected sequence number.

Acknowledging the Next Expected Sequence Number

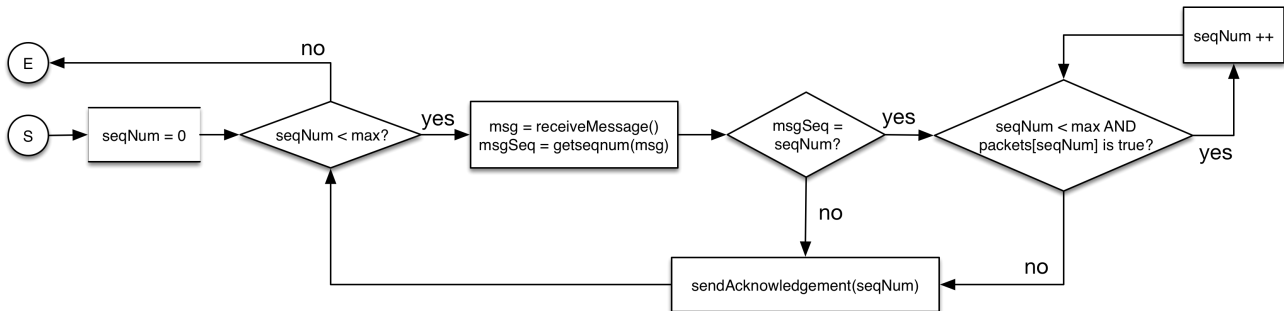
If the received sequence number was the next expected sequence number, we can respond to the client with a higher sequence number, in order to move the window. Since we may have recorded other sequence numbers after the current one, the acknowledgment may be cumulative of several packets.

In order to achieve this, a while loop is used. While the boolean at the index of the expected sequence number is true, the expected sequence number is incremented. This process will stop once it is false. We then know that every sequence number behind this number has been received. We use this sequence number to perform the cumulative acknowledgement to the client.

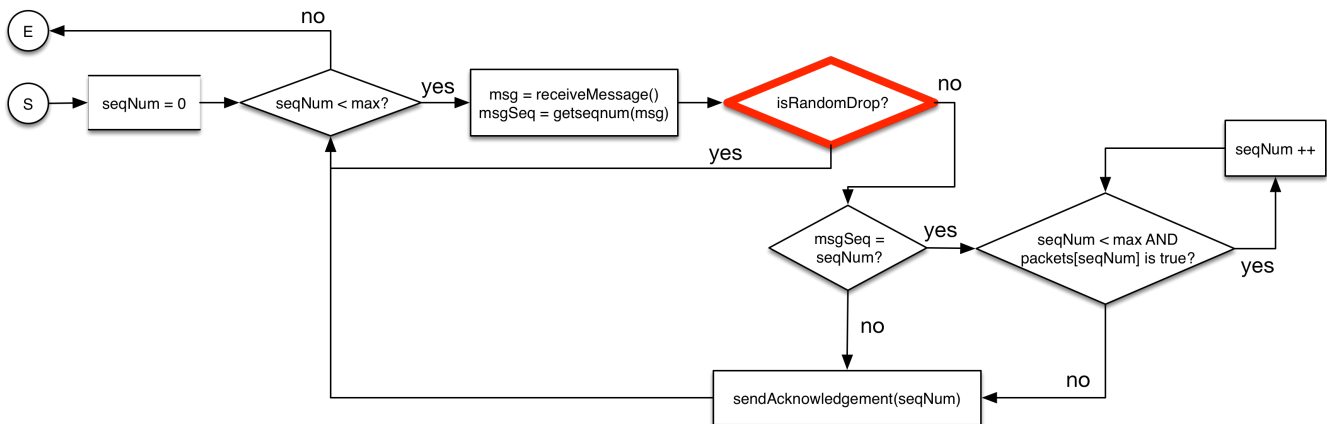
Differences in Server for UDPA

The main difference in the server is that after the receive block, the rest of the server is wrapper inside an if-statement. The if-statement determines if a random drop has happened. In this event the server will not do anything with the message it received; effectively a message drop.

Server: Early Retransmit



Server: Early Retransmit (with drop)



Discussion

Overall, better results were observed in all circumstances by increasing the window size. **These benefits became minimal in my tests after achieving a window size of 10.** After 10, then the results became fairly similar, most likely due to network saturation and the high speed of the local area network.

For UDPA, I saw the expected results, in that **as the drop percentage increased, the amount of time to fully transmit 20,000 messages also increased.**

1 GBPS vs 100 MBPS

The difference between 1gbps and 100mbps seemed to be mostly in terms of performance, where 100 MBPS performed much worse. The chart for the 100 mbps also seemed to have a much higher variance in the results. This led to less smooth of a graph. for these results, whereas the 1gbps sliding window had a much smaller variance between similar results.

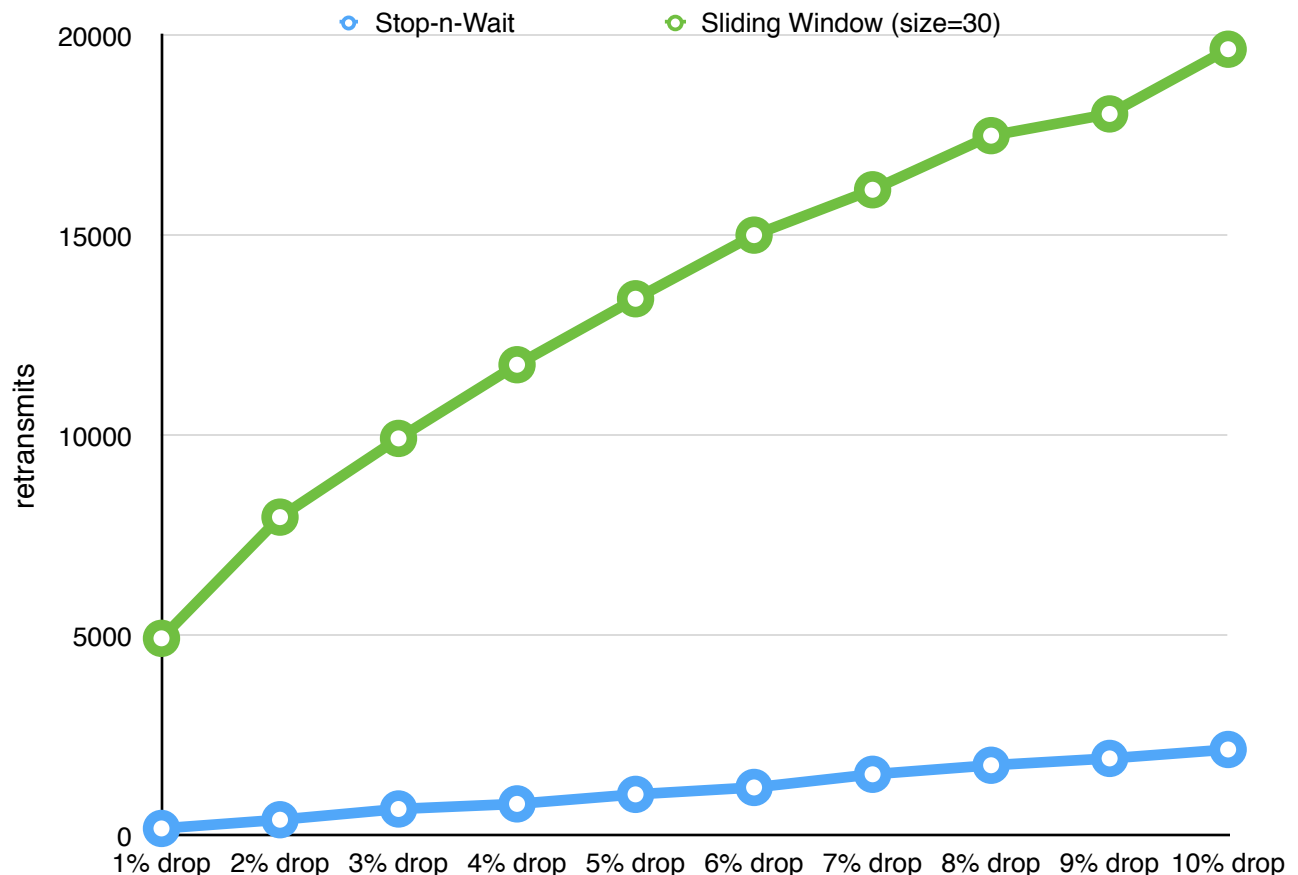
For stop & wait, the difference appeared in that the 100mbps network took significantly longer to transmit the messages than the 1gbps stop & wait.

Drop Rate's Effect on Performance

While it is hard to say for certain from the graph, the drop rate seemed to cause the stop & wait algorithm to perform slightly worse over time than the effect it had on the window size of 30. This could be related to the fact that the stop & wait algorithm will always timeout from a dropped packet, whereas the sliding window will continue to transmit, and can notice from the ACK's received back that it missed a packet.

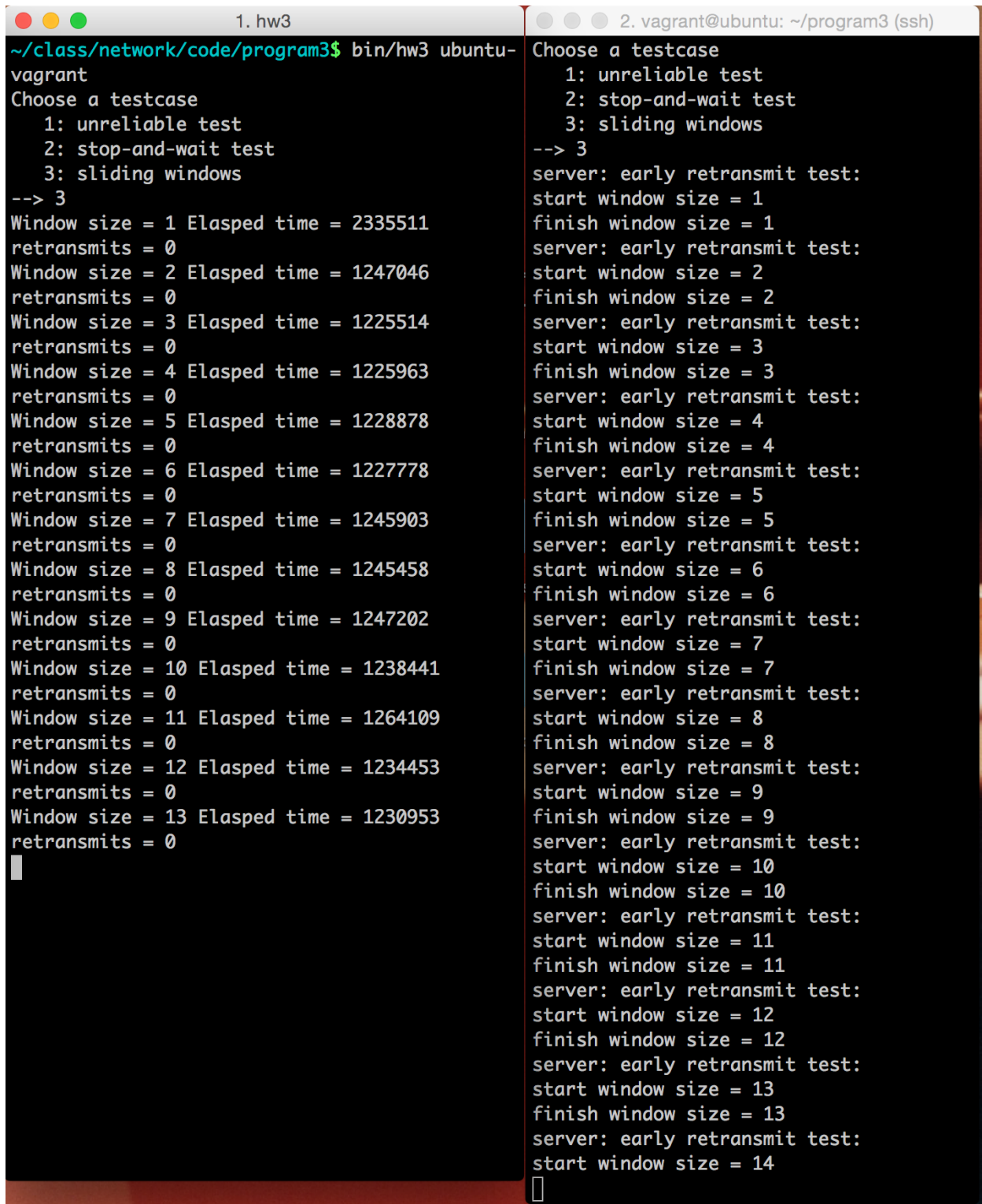
Retransmits

There is a significantly higher amount of retransmits for sliding window size 30, than for the stop & wait algorithm. This is because the sliding window is using the Go-back-N approach, so when a packet is dropped, it jumps back to the beginning of the window, rather than just repeating the dropped packet. **Overall, throughput is still higher for the sliding window.**



Screenshots

UDP Screenshot



```
1. hw3
~/class/network/code/program3$ bin/hw3 ubuntu-
vagrant
Choose a testcase
  1: unreliable test
  2: stop-and-wait test
  3: sliding windows
--> 3
Window size = 1 Elapsed time = 2335511
retransmits = 0
Window size = 2 Elapsed time = 1247046
retransmits = 0
Window size = 3 Elapsed time = 1225514
retransmits = 0
Window size = 4 Elapsed time = 1225963
retransmits = 0
Window size = 5 Elapsed time = 1228878
retransmits = 0
Window size = 6 Elapsed time = 1227778
retransmits = 0
Window size = 7 Elapsed time = 1245903
retransmits = 0
Window size = 8 Elapsed time = 1245458
retransmits = 0
Window size = 9 Elapsed time = 1247202
retransmits = 0
Window size = 10 Elapsed time = 1238441
retransmits = 0
Window size = 11 Elapsed time = 1264109
retransmits = 0
Window size = 12 Elapsed time = 1234453
retransmits = 0
Window size = 13 Elapsed time = 1230953
retransmits = 0
█

2. vagrant@ubuntu: ~/program3 (ssh)
Choose a testcase
  1: unreliable test
  2: stop-and-wait test
  3: sliding windows
--> 3
server: early retransmit test:
start window size = 1
finish window size = 1
server: early retransmit test:
start window size = 2
finish window size = 2
server: early retransmit test:
start window size = 3
finish window size = 3
server: early retransmit test:
start window size = 4
finish window size = 4
server: early retransmit test:
start window size = 5
finish window size = 5
server: early retransmit test:
start window size = 6
finish window size = 6
server: early retransmit test:
start window size = 7
finish window size = 7
server: early retransmit test:
start window size = 8
finish window size = 8
server: early retransmit test:
start window size = 9
finish window size = 9
server: early retransmit test:
start window size = 10
finish window size = 10
server: early retransmit test:
start window size = 11
finish window size = 11
server: early retransmit test:
start window size = 12
finish window size = 12
server: early retransmit test:
start window size = 13
finish window size = 13
server: early retransmit test:
start window size = 14
█
```

UDPA Screenshot

```
1. hw3a
g++ -o bin/hw3 UdpSocket.cpp udp.cpp Timer.cpp hw3.cpp
g++ -o bin/hw3a UdpSocket.cpp udpa.cpp Timer.cpp hw3a.cpp
~/class/network/code/program3$ clear
~/class/network/code/program3$ bin/hw3a ubuntu-vagrant
Choose a testcase
  1: unreliable test
  2: stop-and-wait test
  3: sliding windows
--> 3
Window size = 1 drop percent = 0 Elapsed time = 2314396
retransmits = 0
Window size = 1 drop percent = 1 Elapsed time = 2712650
retransmits = 197
Window size = 1 drop percent = 2 Elapsed time = 3149361
retransmits = 415
Window size = 1 drop percent = 3 Elapsed time = 3658912
retransmits = 684
Window size = 1 drop percent = 4 Elapsed time = 3852093
retransmits = 811
Window size = 1 drop percent = 5 Elapsed time = 4273401
retransmits = 1049
Window size = 1 drop percent = 6 Elapsed time = 4531347
retransmits = 1222
Window size = 1 drop percent = 7 Elapsed time = 5195536
retransmits = 1551
Window size = 1 drop percent = 8 Elapsed time = 5496082
retransmits = 1777
Window size = 1 drop percent = 9 Elapsed time = 5785775
retransmits = 1948
Window size = 1 drop percent = 10 Elapsed time = 6209315
retransmits = 2171
Window size = 30 drop percent = 0 Elapsed time = 1227862
retransmits = 0
Window size = 30 drop percent = 1 Elapsed time = 1594586
retransmits = 4950
█

2. vagrant@ubuntu: ~/program3 (ssh)
Choose a testcase
  1: unreliable test
  2: stop-and-wait test
  3: sliding windows
--> 3
server: early retransmit test:
start window size = 1, drop percent = 0
end window size = 1, drop percent = 0
server: early retransmit test:
start window size = 1, drop percent = 1
end window size = 1, drop percent = 1
server: early retransmit test:
start window size = 1, drop percent = 2
end window size = 1, drop percent = 2
server: early retransmit test:
start window size = 1, drop percent = 3
end window size = 1, drop percent = 3
server: early retransmit test:
start window size = 1, drop percent = 4
end window size = 1, drop percent = 4
server: early retransmit test:
start window size = 1, drop percent = 5
end window size = 1, drop percent = 5
server: early retransmit test:
start window size = 1, drop percent = 6
end window size = 1, drop percent = 6
server: early retransmit test:
start window size = 1, drop percent = 7
end window size = 1, drop percent = 7
server: early retransmit test:
start window size = 1, drop percent = 8
end window size = 1, drop percent = 8
server: early retransmit test:
start window size = 1, drop percent = 9
end window size = 1, drop percent = 9
server: early retransmit test:
start window size = 1, drop percent = 10
end window size = 1, drop percent = 10
server: early retransmit test:
start window size = 30, drop percent = 0
end window size = 30, drop percent = 0
server: early retransmit test:
start window size = 30, drop percent = 1
end window size = 30, drop percent = 1
server: early retransmit test:
start window size = 30, drop percent = 2
█
```