Politecnico di Torino

Internet Performance and Troubleshooting Lab

Analysis of Chargen

Report 2

Group 9

Giuseppe Bruno s319892
Salvatore Giarracca s314701

November 19, 2023

1 Introduction

1.1 TCP Window

Within the TCP protocol, the "window size" indicates the maximum number of bytes the transmitter can send before receiving an acknowledgment. If the transmitter sends a number of bytes greater than the size of the window, the packets will be discarded. This value can be found inside the TCP header and it is 16-bytes long. The TCP option contain also the Window Scale Factor, a value S such that $0 <= S <= 14$: if both sender and receiver support this extension, the Window Size is multiplied by 2^S, increasing the Max Window Size up to $2^{30} - 1$. Having a Receive Window led to an optimization of exchanging data. Without this optimization, the sender could send only one packet and wait for the relative ACK. This would result in being able to send the next packets only after 1 RTT (a significant amount of time), leading to a very slow transmission. The presence of a Receiving Window let the receiver to wait a certain amount of time in which it can receive more packets and then send only one acknowledge to confirm the Bytes received.

```
112998 6.382394159    10.0.9.12       10.0.9.11       TCP    1514     65280 19 → 57760 [ACK] Seq=135875577 Ack=1 Win=65280 Len=1448 TSval=727642047 TSecr=2727983729
112999 6.382394184    10.0.9.12       10.0.9.11       TCP    1514     65280 19 → 57760 [ACK] Seq=135877025 Ack=1 Win=65280 Len=1448 TSval=727642047 TSecr=2727983729
113000 6.382394211    10.0.9.12       10.0.9.11       TCP    1514     65280 19 → 57760 [ACK] Seq=135878473 Ack=1 Win=65280 Len=1448 TSval=727642047 TSecr=2727983729
113001 6.382394235    10.0.9.12       10.0.9.11       TCP    1514     65280 19 → 57760 [ACK] Seq=135879921 Ack=1 Win=65280 Len=1448 TSval=727642047 TSecr=2727983729
113002 6.382394258    10.0.9.12       10.0.9.11       TCP    1514     65280 19 → 57760 [ACK] Seq=135881369 Ack=1 Win=65280 Len=1448 TSval=727642047 TSecr=2727983729
113003 6.382441601    10.0.9.11       10.0.9.12       TCP      66      6912 57760 → 19 [ACK] Seq=1 Ack=135882817 Win=6912 Len=0 TSval=2727983729 TSecr=727642047
```

Figure 1: Wireshark example

1.2 Character Generator Protocol

When a client establish a TCP connection with a server running **chargen** daemon, the server will start to generate characters and send them to the client. Inside the client, the telnet application consume all the data sent by the server and prints the characters inside the terminal (see RFC 864 for further informations). We will analyze deeply using **telnet** and **wireshark** how client and server exchange data during time, focusing mostly on Client Receiving Window and Server Sequence Number and how they evolves during time. The client **SeqNo** is constant over time because does not send data to the server: this is only incremented by 5 when **CTRL + C** is pressed due to how telnet encode the combination (5 bytes) and also incremented by one after the **FIN** message that consumes one Byte. As a consequence the payload of client packets is always 0 excluding when the combination is sent. The Server **AckNo** is incremented according to the the client SeqNo. The payload size of server packets is almost always equal to the MSS (1460B) including the option field of TCP header (12B). This go down to 0 when **CTRL +]** is pressed because the client cannot receive data anymore. Also the Receiving window of the server is always constant because it not process any data. All the plots can be found in the Appendix.

2 Experiment 1

2.1 Environment setup

We set up two computers in the lab, the **client** and the **server**, connected by a switch, with **10.0.9.11** and **10.0.9.12** addresses respectively. In each computer every offloading capability have been disabled.

2.2 Sequence of events

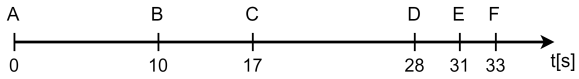
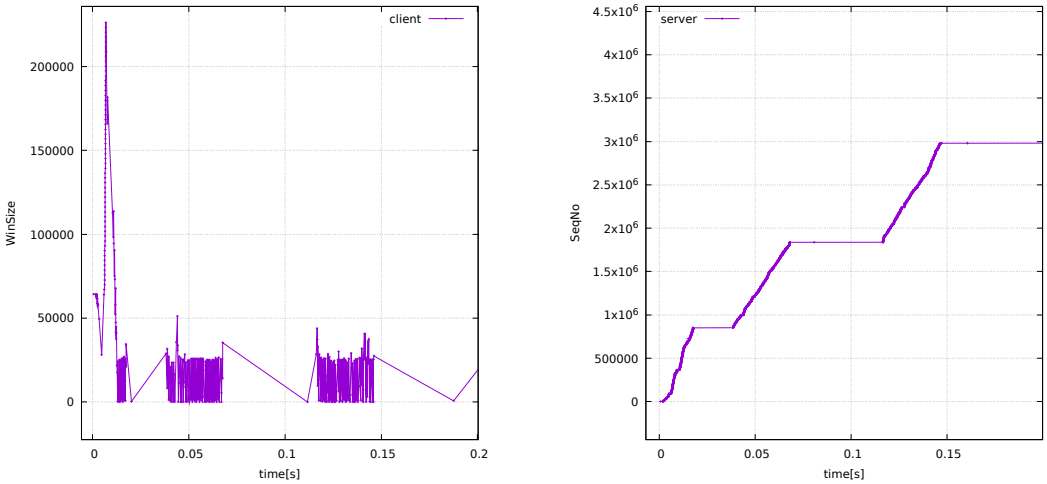


Figure 2: timeline

- **A**: `telnet 10.0.9.2 19`
- **B**: hit **ctrl +]**
- **C**: hit **Enter**
- **D**: hit **ctrl + c**
- **E**: hit **ctrl +]**
- **F**: send **quit**

2.3 Event A

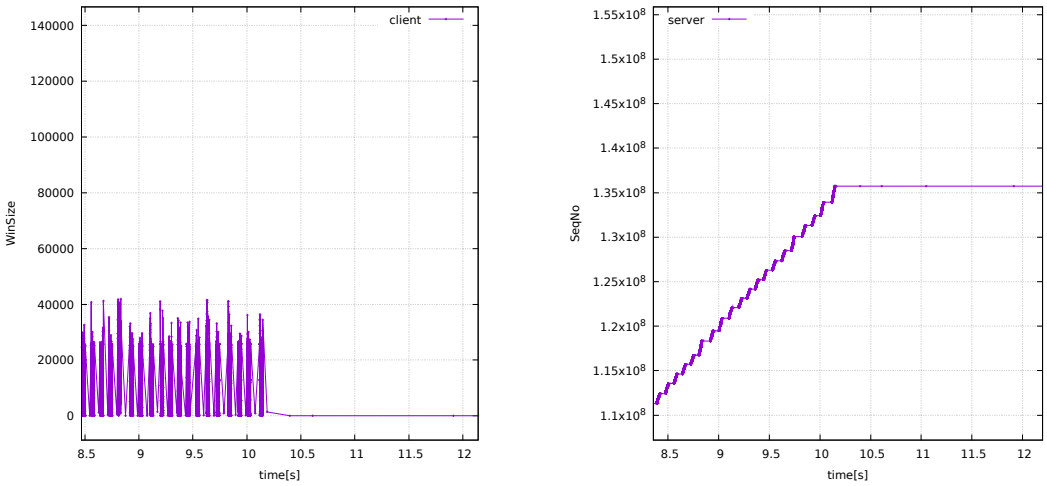


(a) Client Receiving Window (b) Server Sequence Number

Figure 3: Exp 1 start

When TCP connection between client and server is established, the client receiving window have a peak, indicating that the buffer allocated by the client is almost empty. When data arrives, they are put inside the buffer and the CRWIN decreases constantly. After some milliseconds it is possible to observe in Figure 3a that the CRWIN is oscillating up and down. This means that the TCP flow control algorithm is working properly: the server sends data and the client receives it, filling the client buffer almost entirely and when a packet with a **PSH** flag arrives, TCP pushes the data buffered to the application, causing an enlargement of the CRWIN. This is the cause of the oscillations showed in the plot above.
As expected, the Server Sequence Number is always increasing as showed in Figure 3b. It is also possible to notice the "Slow start" algorithm curve at the very beginning of the plot, but since the experiment is done in a clean environment, this ends in few milliseconds.

2.4 Event B

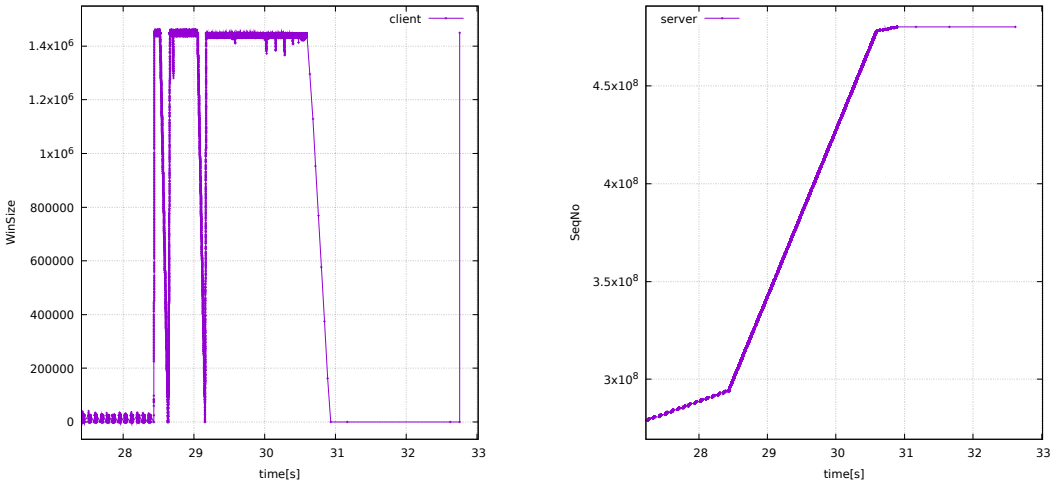


(a) Client Receiving Window (b) Server Sequence Number

Figure 4: Exp 1 middle

When **CTRL +]** is pressed, telnet application enters in "command mode". The client set its receiving window to 0 and when the server receive the packet that contain a window equal to 0, it understand that do not have to send more data. From now on the server wait a certain amount of time and send a "TCP Keep Alive" packet. The client responds to this packet again with a "TCP zero window" that is an "ACK" to the to the server packet. In the Figure 4b those "TCP Keep Alive" packets are the ones in the flat line, and it can be observed that the time elapsed since the previous packet doubles each time. This last until **event C** and after that, the client will send a packet with a Receiving Window different from 0. The server will now starts to resend data as before.

2.5 Event D



(a) Client Receiving Window

(b) Server Sequence Number

Figure 5: Exp 1 end

After approximately 28.5 seconds, the size of the receiving window significantly increases and tends to stabilize, while the trend of SeqNo sees an increase in its slope. This behavior is due to the pressing of the **CTRL + C** keys on the Client, which interrupts the printing of characters on the screen. Thus, the receiver experiences a reduced workload and increased speed in processing data, focusing solely on verifying the received Bytes. As a consequence, the CRWIN increases nearly to the Max Receive Window as showed in Figure 5a. Simultaneously, the transmitter gains efficiency in packet transmission due to higher Window Size perceived and rise the transmission rate of the data. As expected, we can notice that from second 25.5 to second 30.5, the throughput is near to the theoretical limit ($7 * 10^8$ Mbit/s).

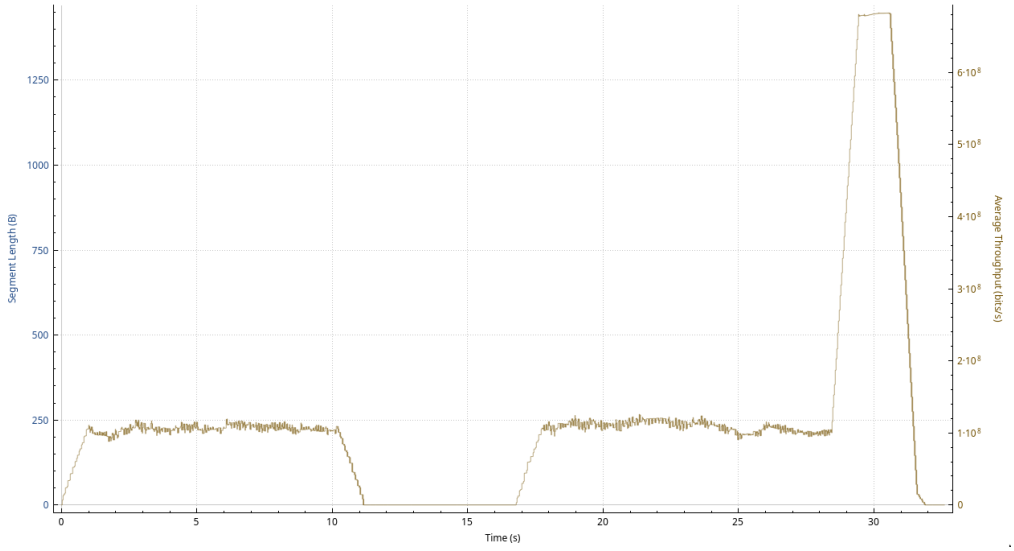


Figure 6: Throughput

3 Experiment 2

3.1 Environment setup

We set up two virtual machines in virtualbox, the **client** and the **server** with **10.0.9.1** and **10.0.9.2** addresses respectively, connected in the same lan using 100 Mbit/s NICs (PCnet-FAST III). The purpose of the experiment is to analyze the data exchange between the two peers when there is an high cpu load. To achieve this, we ran to the host machine a stress test using **stress** command (`stress -c 14`). The host machine have an intel processor with 16 cpu cores. The stress command previously mentioned run a stress test that saturate 14 cores, leaving only two free to the guest machines.

3.2 Sequence of events

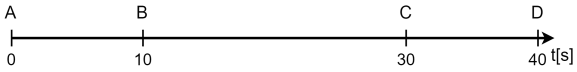
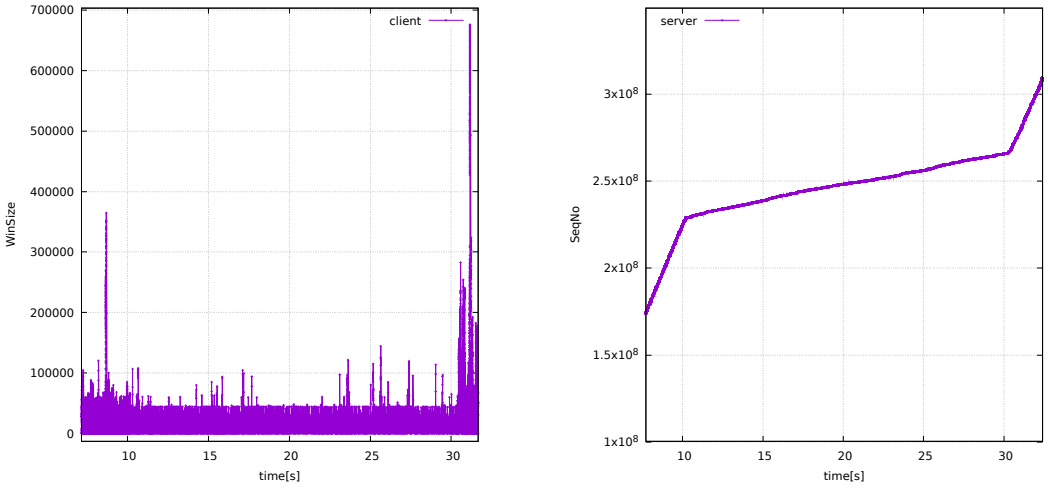


Figure 7: timeline

- **A**: `[telnet 10.0.9.2 19]`
- **B**: host OS stress begin
- **C**: host OS stress end
- **D**: send **quit**

3.3 Event B



(a) Client Receiving Window

(b) Server Sequence Number

Figure 8: Exp 2 stress

From the setup of the TCP connection up to 10s, always remain almost unchanged with respect to the previous experiment. There is a bit of overhead due to the virtual environment as we can see from the Figure 8a. Thus, the oscillations of the CRWIN are more emphasized. Therefore, after stress test begin, we can see a significant decrease of Server Sequence Number slope. When there is an high load on the cpu, The resources shared to other processes are less and this cause a decreasing of the speed in which packet are sent by the server. This last until the stress test ends (**event C**), after that the server start to sent packets at normal speed(Figure 8b). However, during the stress test, we notice that the CRWIN is more stable, in the sense that it oscillates as always but with less "outlier" values of size.

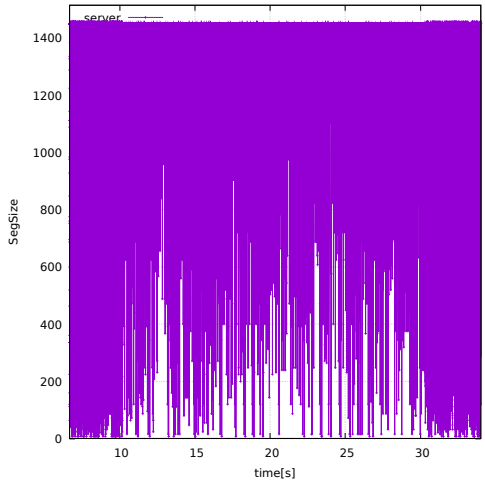


Figure 9: Server Segment Size

Another interesting thing to consider is the Server Segment Size over time during the stress test (Figure 10). Here we can see, between 10s and 30s, the segment size is on average bigger compared to normal conditions. This could be related to the slowness of the environment in that moment, because the stress test is performed on the host machine. Which means that the server is slower sending packet but also the client receiving them.

3.4 Considerations

With the stress test performed on the host machine we saturated the resources of both client and server. In a real world scenario, an interesting possible experiment could be performing a stress test only to one peer, for example the server. In this case we expect Server SeqNo over time plot similar to the one obtained in the second experiment, but a different plot of the CRWIN over time. In particular, we expect a bigger CRWIN on average, compared to 8a. This because the client could process data normally, while the server will have less resources, becaming the bottleneck of the conversation.

4 Acknowledgment and Sequence Number Plots

The server and the client use the same values to indicate sequence numbers and acknowledgments. Plotting the trend of Ack and SeqNo over time results in two slightly different graphs. Considering the client, the number of packets received is much higher than the number of packets sent, so the number of samples the green trend is lower (less precise) compared to the violet one. These two graphs are actually distinct; the client does not send an Ack for every received packet, resulting in a scenario where there is a SeqNo for every acknowledgment, but not conversely. When visualizing both trends on the screen, it becomes evident that the green graph is less accurate than the purple one, primarily due to a lower number of samples.

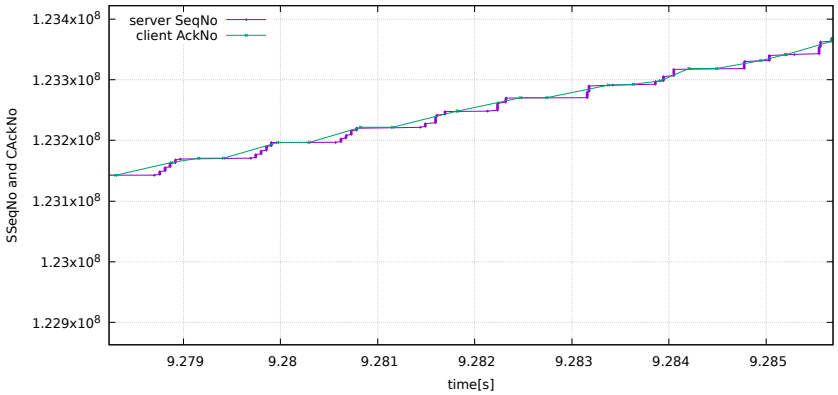


Figure 10: Client ACKs and Server SeqNums

A Script

- a file **data.txt** (with wireshark extracted data) should be present into the directory where the script is executed.

```
#!/bin/sh
CLIENT="./client"
SERVER="./server"

# Client
cat data.txt | grep "Ack=" | grep -P "\s→\s19\s" | tr -s ' ' | cut -d' ' -f 3 > $CLIENT/time.txt
cat data.txt | grep "Ack=" | grep -P "\s→\s19\s" | tr -s ' ' | cut -d'=' -f 2 | cut -d' ' -f 1 > $CLIENT/seqno.txt
cat data.txt | grep "Ack=" | grep -P "\s→\s19\s" | tr -s ' ' | cut -d'=' -f 3 | cut -d' ' -f 1 > $CLIENT/ackno.txt
cat data.txt | grep "Ack=" | grep -P "\s→\s19\s" | tr -s ' ' | cut -d'=' -f 4 | cut -d' ' -f 1 > $CLIENT/rwin.txt
cat data.txt | grep "Ack=" | grep -P "\s→\s19\s" | tr -s ' ' | cut -d'=' -f 5 | cut -d' ' -f 1 > $CLIENT/seglen.txt

paste $CLIENT/time.txt $CLIENT/seqno.txt $CLIENT/ackno.txt $CLIENT/rwin.txt $CLIENT/seglen.txt > $CLIENT/data.txt

# Server
cat data.txt | grep -P "\s19\s→\s" | tr -s ' ' | cut -d' ' -f 3 > $SERVER/time.txt
cat data.txt | grep -P "\s19\s→\s" | tr -s ' ' | cut -d'=' -f 2 | cut -d' ' -f 1 > $SERVER/seqno.txt
cat data.txt | grep -P "\s19\s→\s" | tr -s ' ' | cut -d'=' -f 3 | cut -d' ' -f 1 > $SERVER/ackno.txt
cat data.txt | grep -P "\s19\s→\s" | tr -s ' ' | cut -d'=' -f 4 | cut -d' ' -f 1 > $SERVER/rwin.txt
cat data.txt | grep -P "\s19\s→\s" | tr -s ' ' | cut -d'=' -f 5 | cut -d' ' -f 1 > $SERVER/seglen.txt

paste $SERVER/time.txt $SERVER/seqno.txt $SERVER/ackno.txt $SERVER/rwin.txt $SERVER/seglen.txt > $SERVER/data.txt

gnuplot <<EOF &
        set terminal wxt enhanced font ",22"
        set xlabel "time[s]"
        set ylabel "SeqNo"
        set key left
        set grid
        plot "$CLIENT/data.txt" using 1:2 title "client" with linespoint
        pause 1000
EOF
gnuplot <<EOF &
        set terminal wxt enhanced font ",22"
        set xlabel "time[s]"
        set ylabel "AckNo"
        set key left
        set grid
        plot "$CLIENT/data.txt" using 1:3 title "client" with linespoint
        pause 1000
EOF
gnuplot <<EOF &
        set terminal wxt enhanced font ",22"
        set xlabel "time[s]"
        set ylabel "WinSize"
        set key right
        set grid
        plot "$CLIENT/data.txt" using 1:4 title "client" with linespoint
        pause 1000
EOF
gnuplot <<EOF &
        set terminal wxt enhanced font ",22"
        set xlabel "time[s]"
        set ylabel "SegSize"
        set key left
        set grid
        plot "$CLIENT/data.txt" using 1:5 title "client" with linespoint
        pause 1000
EOF
gnuplot <<EOF &
        set terminal wxt enhanced font ",22"
        set xlabel "time[s]"
        set ylabel "SeqNo"
        set key left
        set grid
        plot "$SERVER/data.txt" using 1:2 title "server" with linespoint
        pause 1000
EOF
gnuplot <<EOF &
        set terminal wxt enhanced font ",22"
        set xlabel "time[s]"
        set ylabel "AckNo"
        set key left
        set grid
        plot "$SERVER/data.txt" using 1:3 title "server" with linespoint
        pause 1000
EOF
gnuplot <<EOF &
        set terminal wxt enhanced font ",22"
        set xlabel "time[s]"
        set ylabel "WinSize"
        set key left
        set grid
```

```
        plot "$SERVER/data.txt" using 1:4 title "server" with linespoint
        pause 1000
EOF
gnuplot <<EOF &
        set terminal wxt enhanced font ",22"
        set xlabel "time[s]"
        set ylabel "SegSize"
        set key left
        set grid
        plot "$SERVER/data.txt" using 1:5 title "server" with linespoint
        pause 1000
EOF
gnuplot <<EOF &
        set terminal wxt enhanced font ",22"
        set xlabel "time[s]"
        set ylabel "SSeqNo and CAckNo"
        set key left
        set grid
        plot "$SERVER/data.txt" using 1:2 title "server SeqNo" with linespoint, "$CLIENT/data.txt" using 1:3 title "client AckNo" with linespoint
        pause 1000
EOF
```

B First Experiment

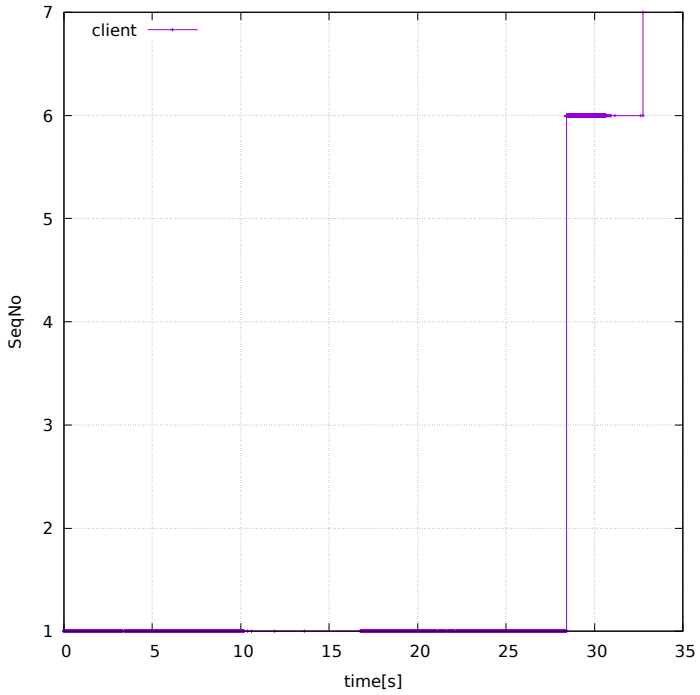


Figure 11: Client Sequence Number

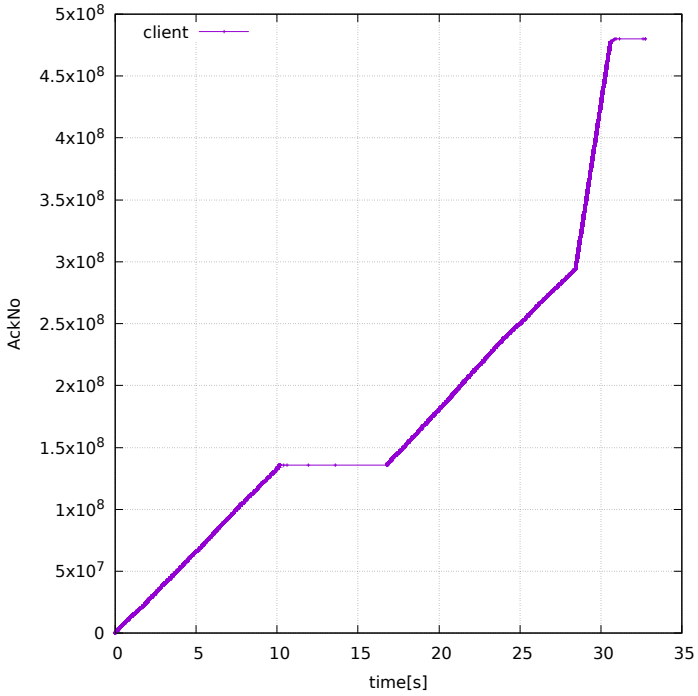


Figure 12: Client Acknowledgment Number

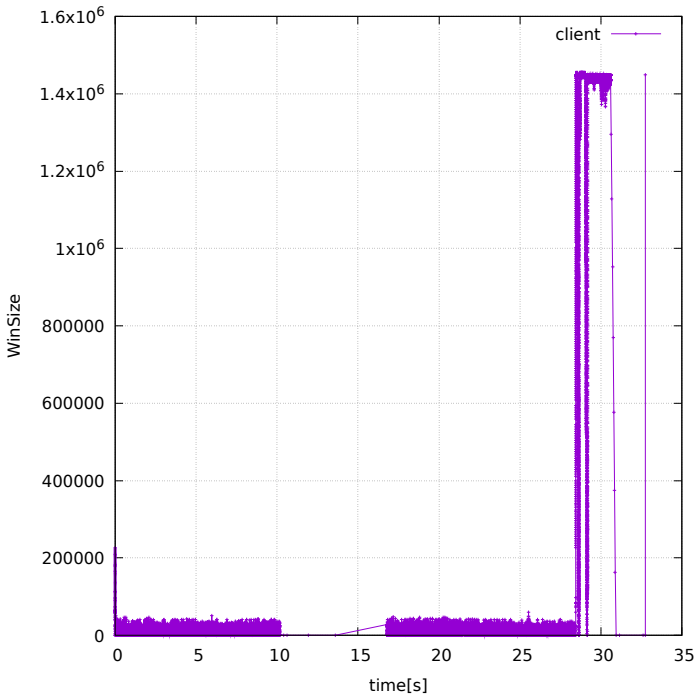


Figure 13: Client Window Size

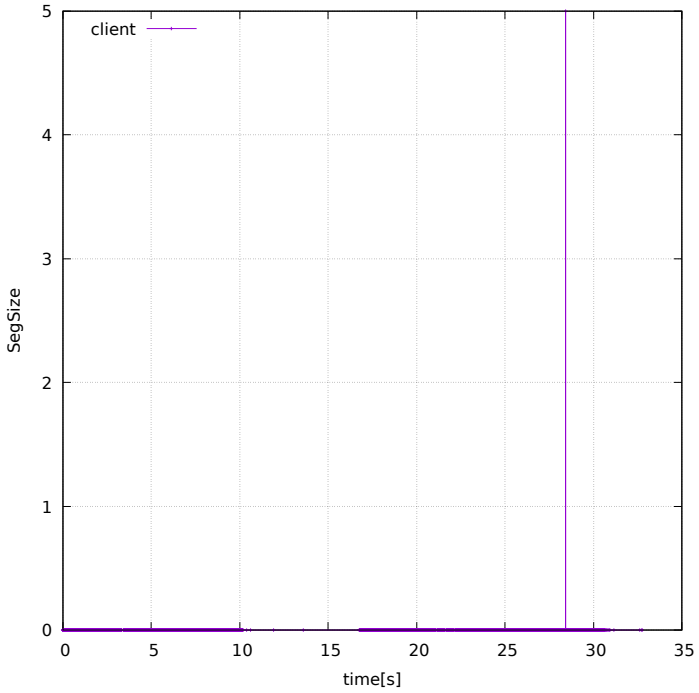


Figure 14: Client Segment Size

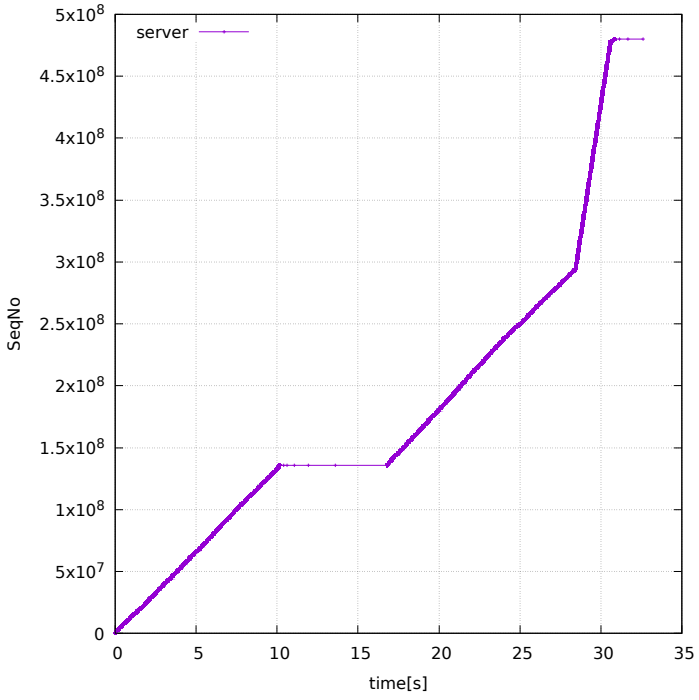


Figure 15: Server Sequence Number

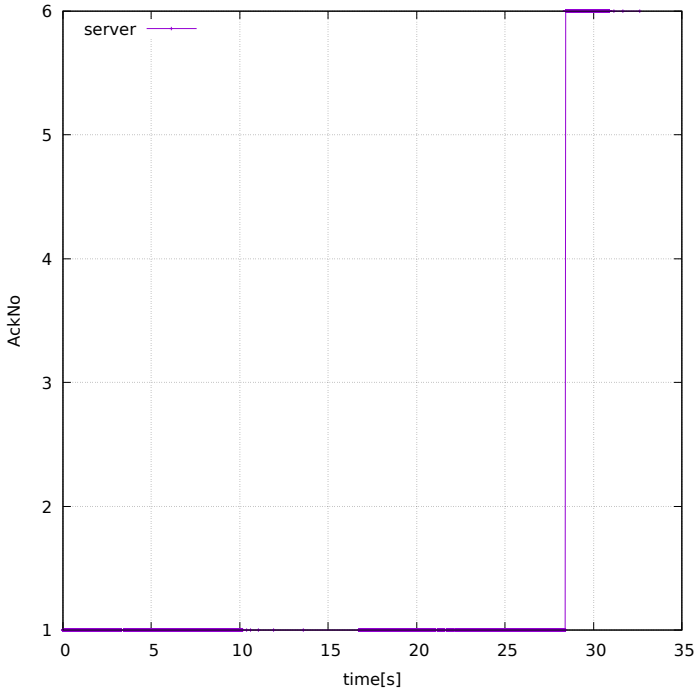


Figure 16: Server Acknowledgment Number

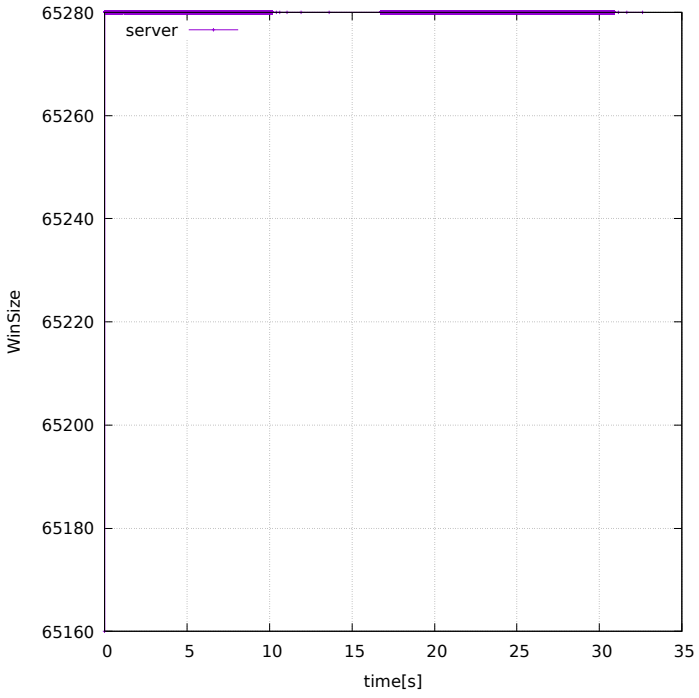


Figure 17: Server Window Size

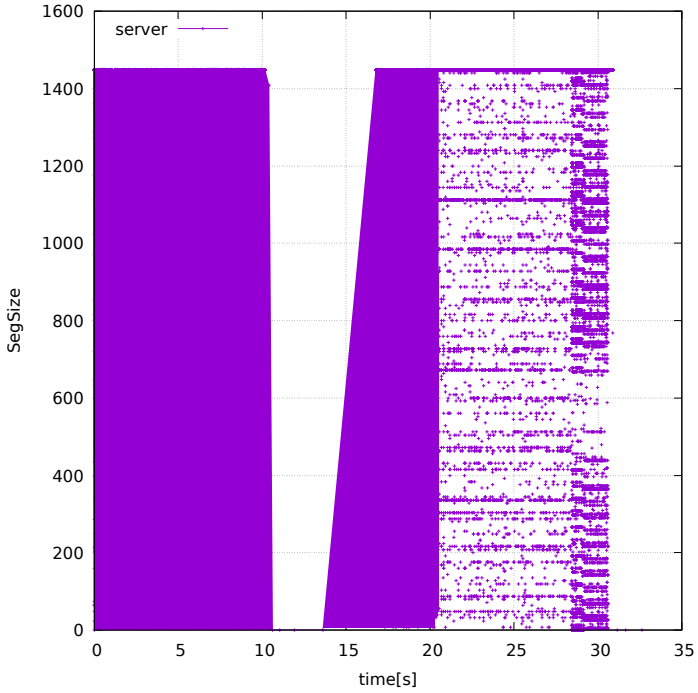


Figure 18: Server Segment Size

C Second Experiment

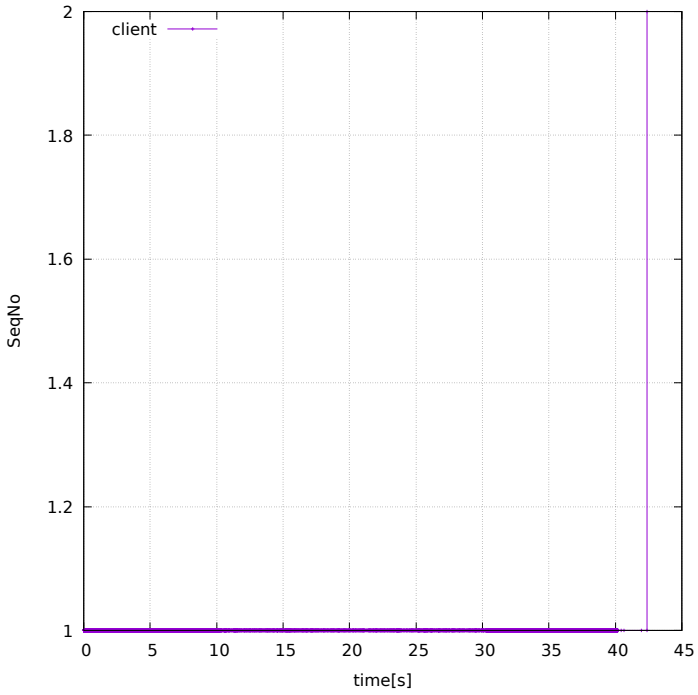


Figure 19: Client Sequence Number

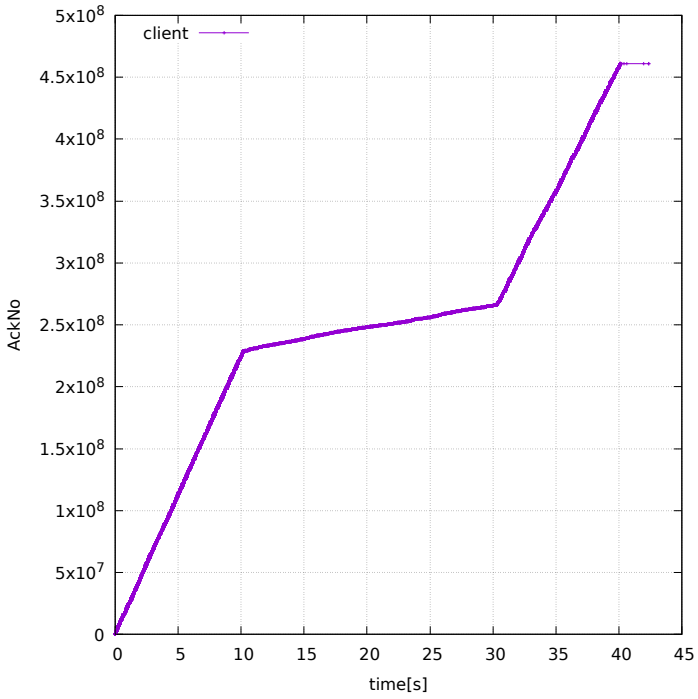


Figure 20: Client Acknowledgment Number

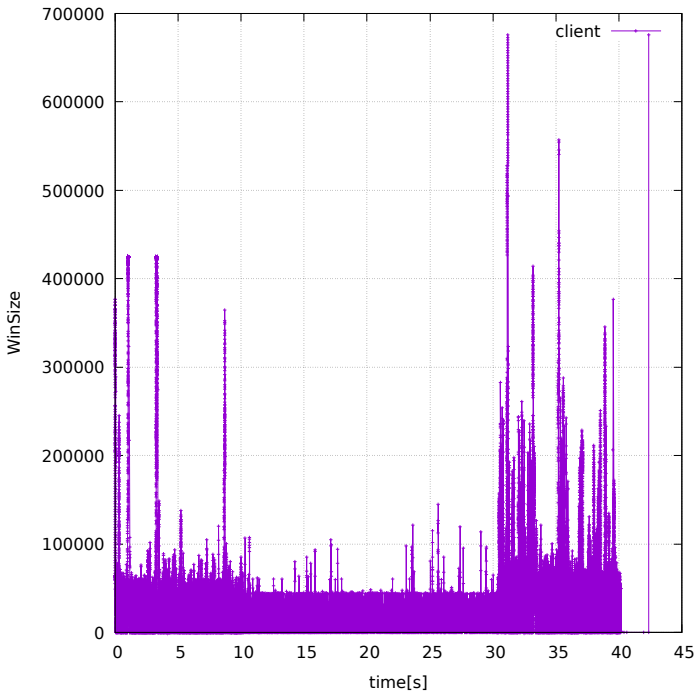


Figure 21: Client Window Size

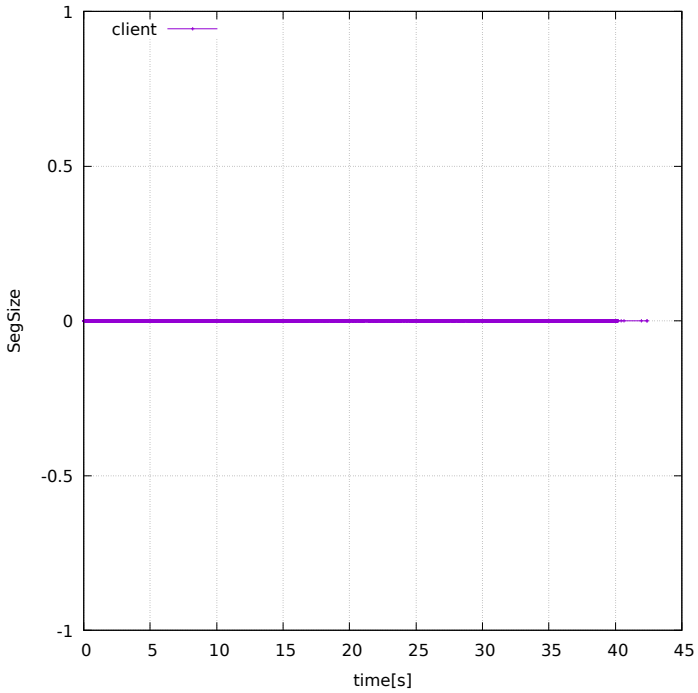


Figure 22: Client Segment Size

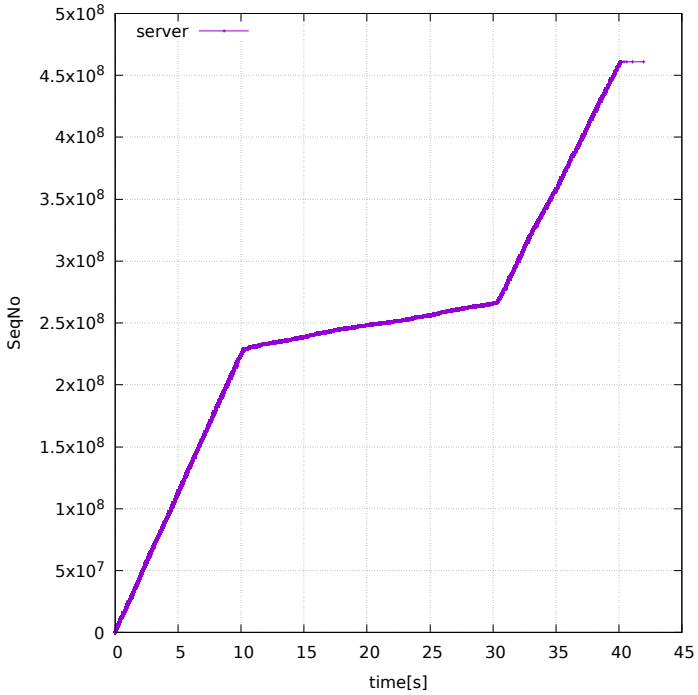


Figure 23: Server Sequence Number

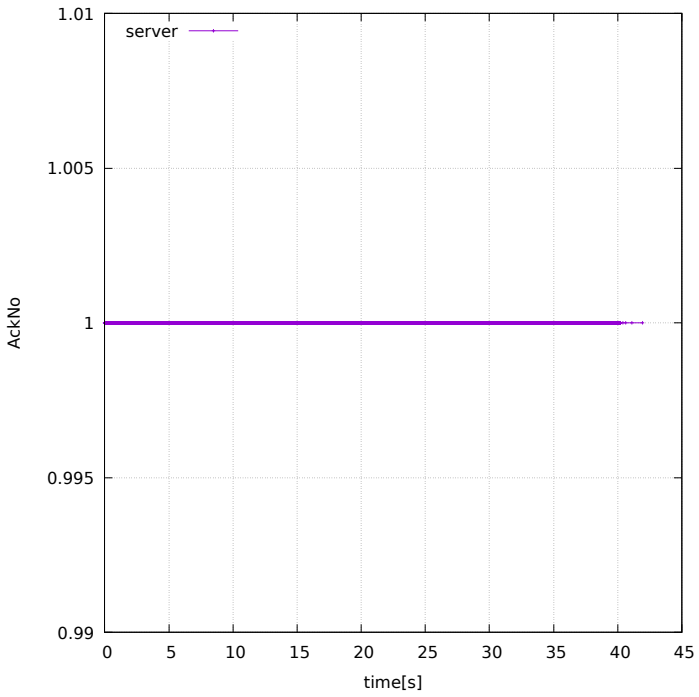


Figure 24: Server Acknowledgment Number

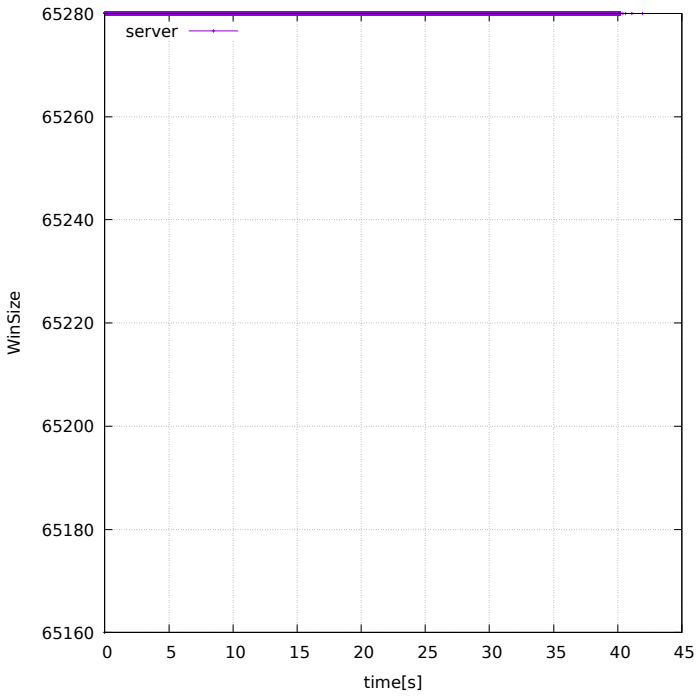


Figure 25: Server Window Size

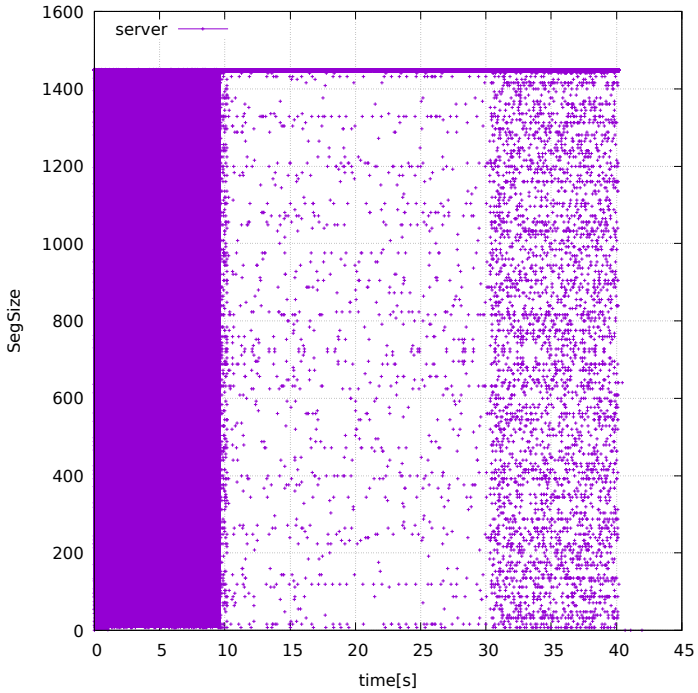


Figure 26: Server Segment Size