CS447 – Computer Networks

Project Test Report

A comparison between UDP and TCP

Project Group 7

Hasan Bilgin

Peker Çelik

Egemen Iscan

Ege Oztas

Demirkan Yilmaz

1. **Introduction**

TCP and UDP (Acronyms for Transmission Control Protocol and User Datagram Protocol respectively.) are both transport layer protocols used for transporting application layer messages through computer networks. They are the most prominent protocols in current use besides the IP protocol. (Which will not be in the scope of this report)

The purpose of this report will be to analyze and investigate these two protocols and observe these two protocols work in practice. We will be conducting a set of tests to see how differing transmission sizes will affect both protocols and how UDP and TCP differ from each other in practice.

1. **Setup**

To conduct these tests, and observe UDP and TCP we have created two pairs of Python code. One pair for UDP and one pair for TCP. We have used latest Python library, *Python 3.9.0* to make these classes of code. To further create network interfaces for these codes we have also used the *socket.py* interface in both pairs. Purpose of this is to create and work with network sockets.

One additional interface we have used for the UDP pair is the *base64.py* interface. The purpose of this interface is to turn the test data into a format that UDP can transmit across.

The class pairs, for both protocols, are consisting of one client and one server class. Upon correct execution of a pair a given text file will be transmitted between the ports created by these classes. The exact relationship between pairs of UDP and TCP classes are not the same due to the implementation of the python code by the project group members, but these small differences does not hinder our experiments. Fuhrer explanation for the workings of these pairs of classes are given in the testing segment of this report.

To observe these classes working and sending each other this data we will be using latest available version of the Wireshark software. Which enables us to track the data packets coming and going from the sockets and the addresses.

1. **Testing**

TCP pair works as follows: In order the tester executes server class and the client class. Server, once executed will create a port and begin listening for the client to be executed, and reach out to Server itself. When this TCP connection is established then server will start to transmit given data to the client side. Once the data is finished the connection between server and the client will be closed.

UDP pair works different from the TCP pair. Order of the execution of the classes is the same, first the server class then the client class. But unlike TCP, when using UDP we had to manually split the given file into small chunks to fit into the buffers. And UDP should ran every time for a datagram.Because of this we expect UDP’s datagram sizes to be inefficient.

To observe how these two procedures, react to a varying data size, we will be testing both of them with tree different data to be transmitted. Ranging from smallest to largest, tests A, B and C are using 2KB, 156KB and 1.790KB .txt files to use in transmission respectively. This range of variance in data size should be enough to highlight any differences of how UDP and TCP scale with transmission size.

Python classes also create an output.txt file upon a successful transmission, filling it with the data that has been delivered. This way we can detect any corruption on the transmissions.

Tests are conducted in a machine with windows 10 OS.

To observe these tests, we will be using Wireshark packet sniffer. Which will be able to show us the packets send to the specific ports. It’s I/O graphs will be of use to show packet sending rates for the procedures. And for ITP we will be able to analyze throughput, window size scaling and other useful data.

**3.1 Expectations**

What we are expecting to see from these tests are the foretold properties of the both protocols. We expect UDP to be less reliable than TCP with a number of packets lost during their transportation to various circumstances of the network they are transmitted through. For TCP we are also expecting both client and the server to establish a dedicated ‘handshake’. Also we expect its transfer window size to be changing with time.

1. **Test Results**

**4.1 Test A**

For test set a we have used a 2Kb .text file as the variable test size. Because of this size we have observed that for both TCP and UDP the number of actual datagrams sent are too few to make patterns and trends whilst using Wireshark to observe the transmission.

**TCP**

Due to the small number of packets send we can see how TCP establishes a connection before and disconnects after datagram transmission. We can also observe that in while the transmission time is too short for a human observation only a fraction of this time is actually spent transmitting data. Establishment and ending of a dedicated connection. 9.32 x 10^-2 seconds took for the whole transmission to begin and end. But the data transmission started with the packet with No. 14 and ended with ack with No. 17 taking only 5 x 10^-5 seconds.

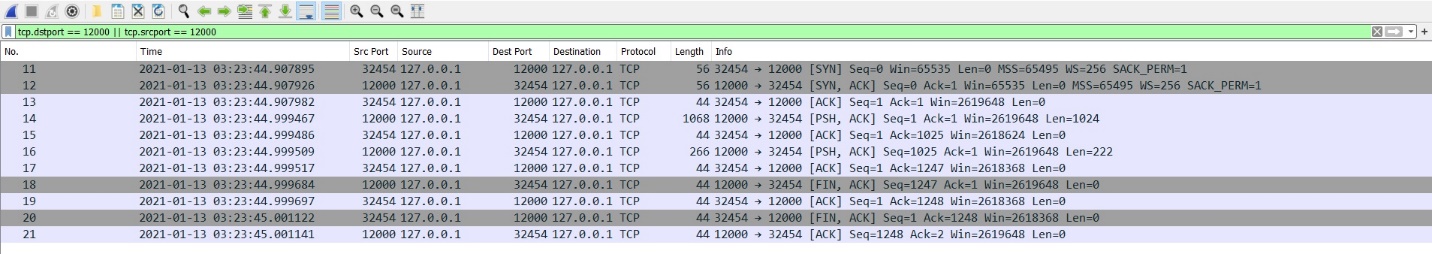
****

Figure 1. Complete packet History of TCP test A

We can see that transmission of data, in the middle of the time table, causes a spike in transmission speed and time spent transmitting last connection ending messages slows the data transmission speed down.

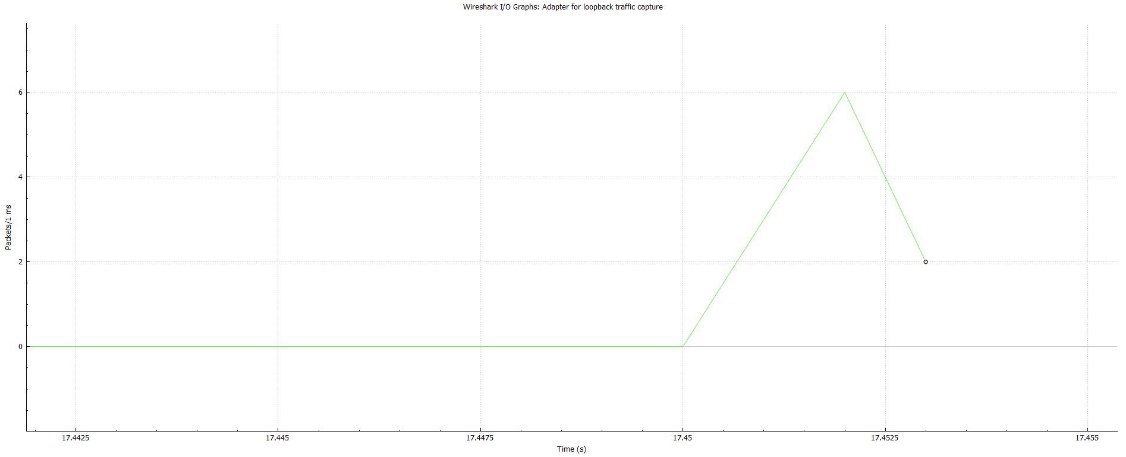
****

Figure 2TCP test A packet transmission rate per ms

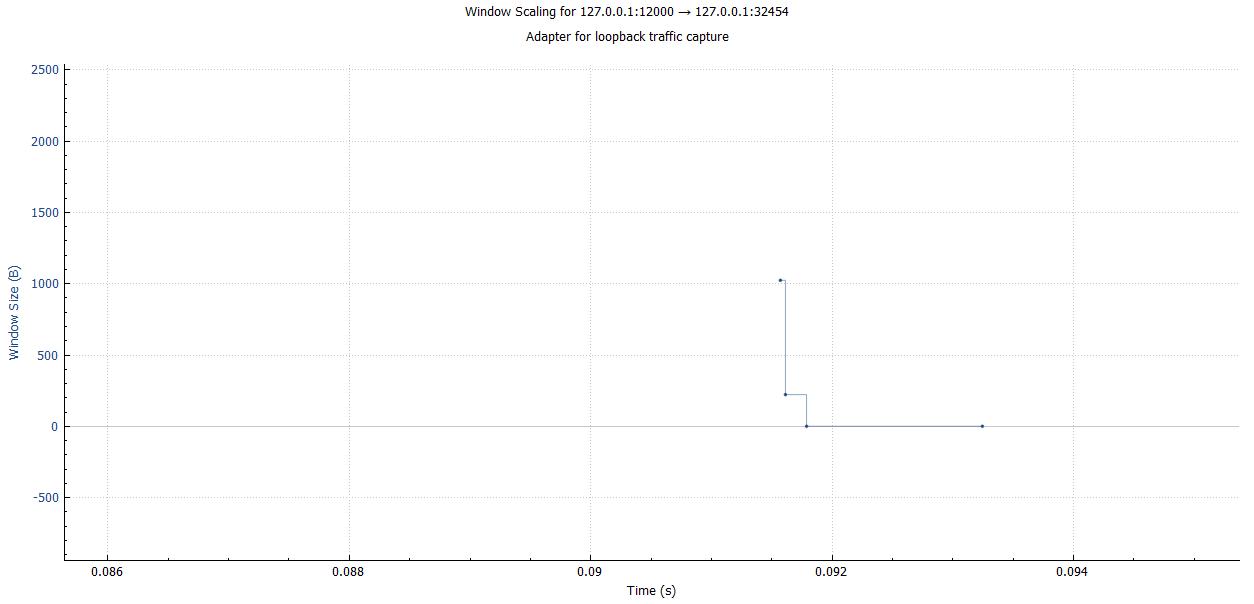


Figure 3 TCP test A Windows scaling

And due to the small number of packets we can’t see the expected windows scaling trend.

**UDP**

Here we can see how much UDP is faster than the TCP. While TCP took 9.32 x 10^-2 at total 2.63 x 10^-3 seconds. Less than one tenth of the time of TCP. We can see that this speed is due to the fact that UDP doesn’t bother with establishing a dedicated connection and just sends the packets out, assuming they can reach the destination. One additional thig is to observe that the amount of data sent in each UDP packet is almost half of how much data is sent in TCP. It doesn’t affect the speed of the transmission speed in a small sized data but we expect it’s effect to increase as we increase file size.

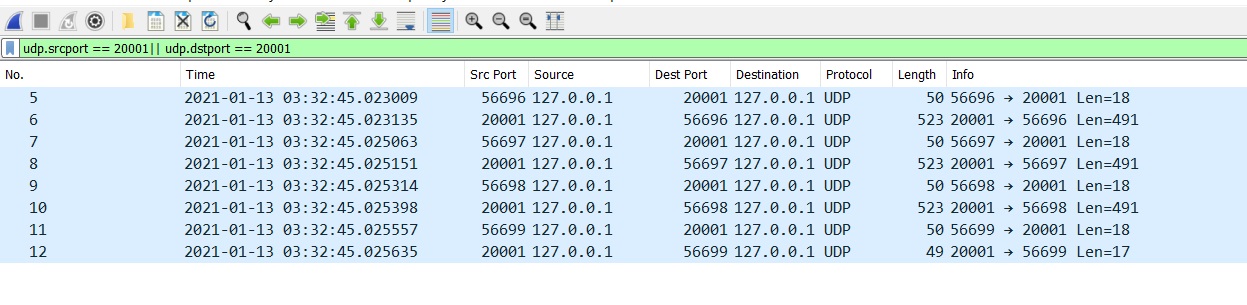
****

Figure 4 UDP test A complete packet history

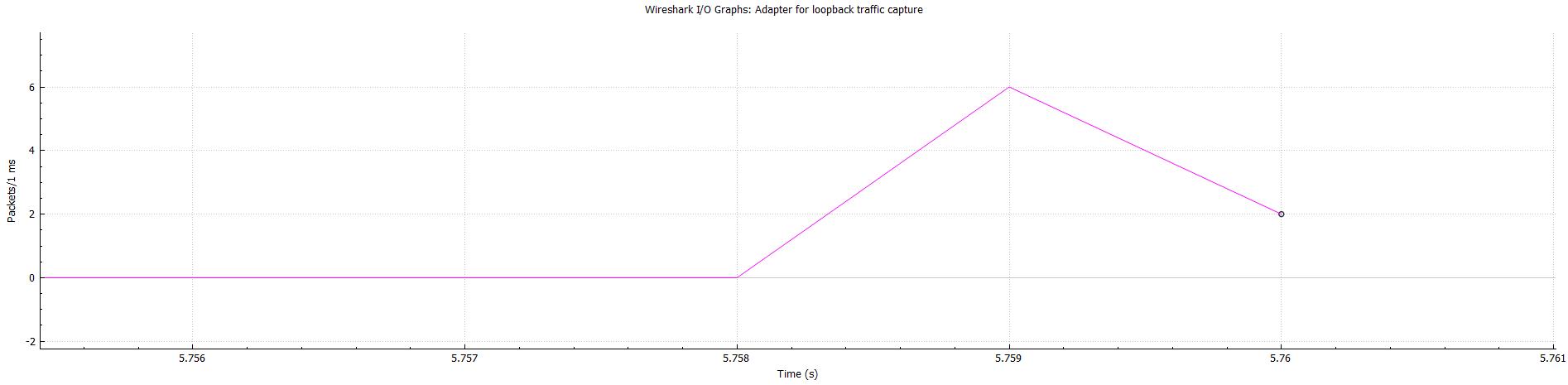
****

Figure 5 UDP test A packet transmission rate per ms

**4.2 Test B**

In test B we have used a .txt file of 156Kb size. Because of this we were able to see trends and patterns in the transmission graphs that Wireshark provided. In addition to this in this test set we had to chance of observing transmission fail, most probably a bug in the code that occurred due to some wrong execution order. Terminating the classes and re-executing them was enough to overcome this problem.

**TCP**

Said failure occurred at TCP test of the data set B. Once re-executed the code ran properly.

Disregarding the bug curiously it took TCP less time to transmit the 156Kb of data than it took to transmit 2Kb of data. Upon closer inspection of the Figure 6 we can see that it took much less time for TCP to start transmitting actual data. 7.02 x 10^-2 seconds to be exact. Upon further testing in data set C we can see that this is not because TCP had a problem with the data set A but the bug in the software caused the TCP code to transmit faster second time it has been executed.

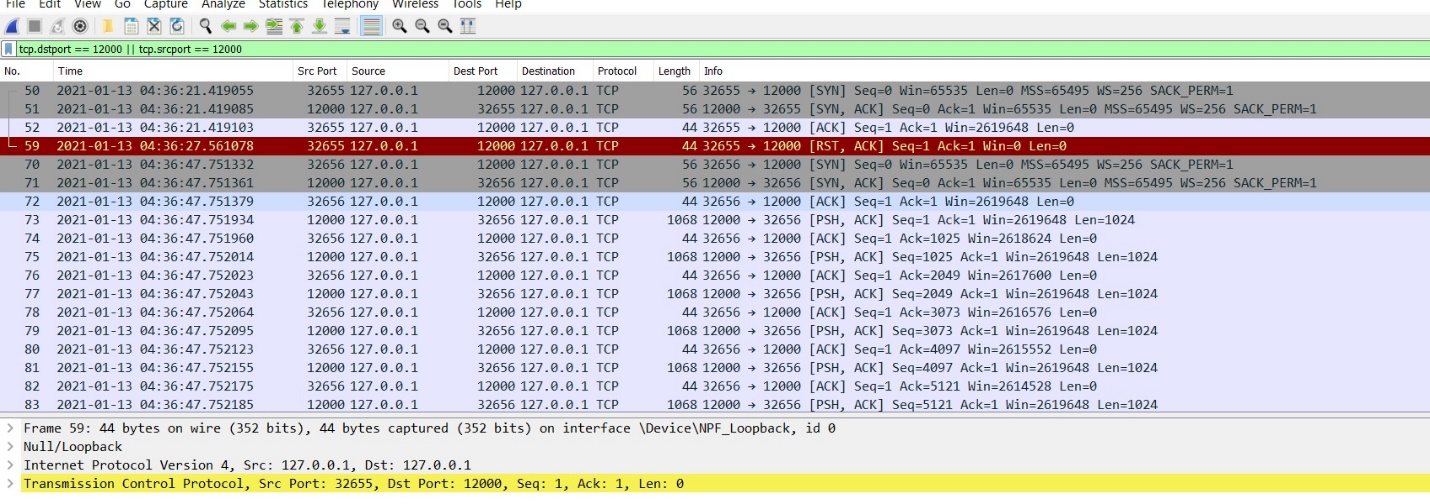
****

Figure 6 beginning of the packet History of TCP test B

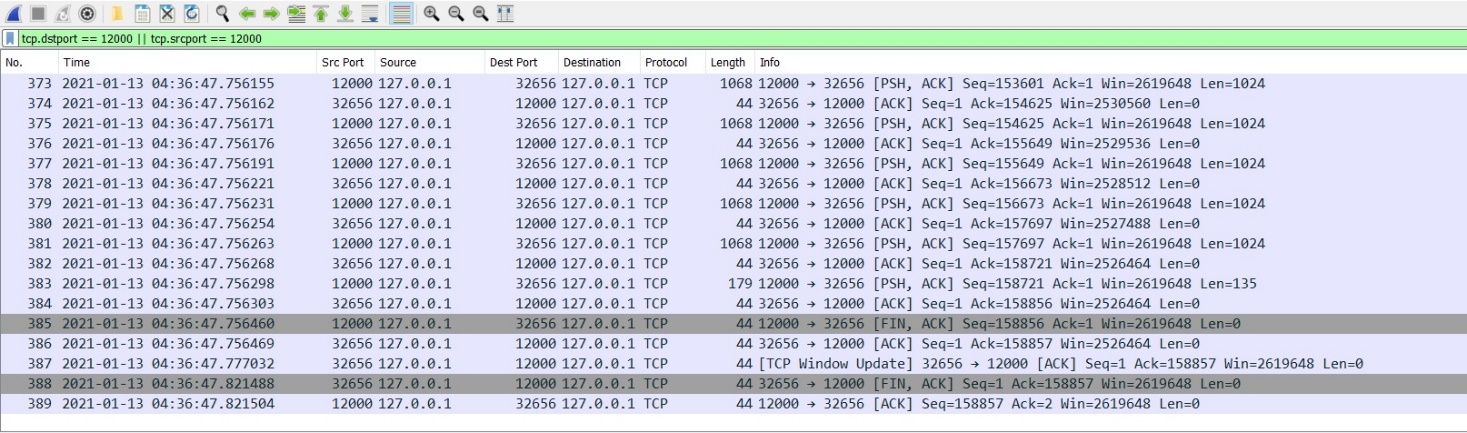
****

Figure 7 ending of the packet History of TCP test B

In window scaling graph we observe that there is no pause of transmission at the start unlike the other two data sets but a constant plateau. Another thing is that this graph subverts our expectations of window size changing according to the traffic destination receives whilst getting the transmission. Our hypothesis is that this is happening because TCP instead of connecting to another device it is connecting to the same device.

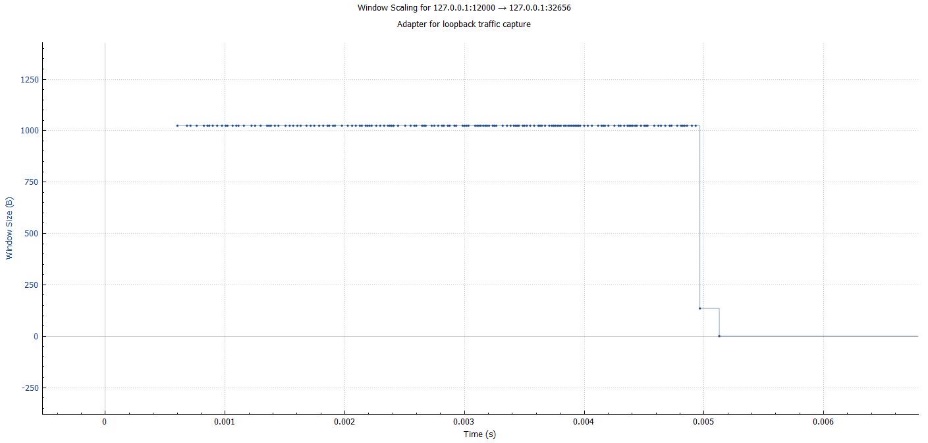


Figure 8 TCP test B window scaling

In the Packet speed graph, we observe the same ‘slowness’ the TCP handshake causes to the graph at the end. In addition to that unlike window size we can see that over time speed of TCP transmission increases similarly to how we expect it increase from TCP’s congestion control.

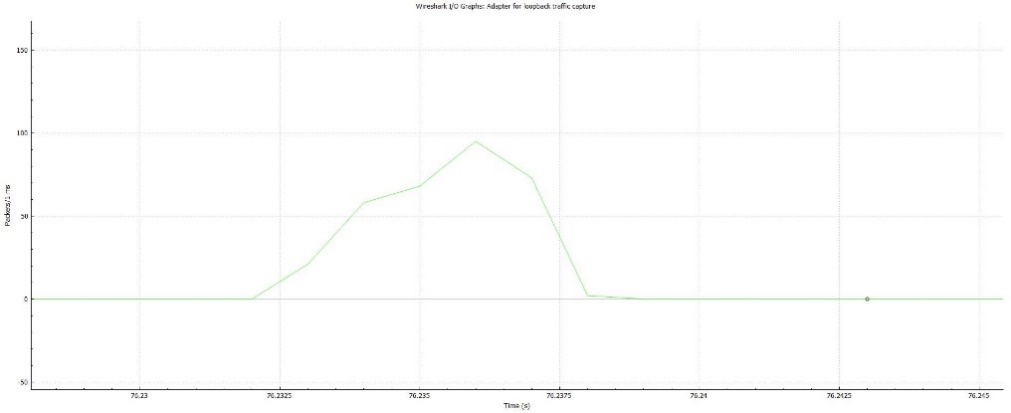


Figure 9 TCP test B packets send per ms.

**UDP**

In this Test we can see how much slower UDP has become due to the amount of data it caries on one datagram. It took UDP 1.93x10^-1 seconds to transfer the same amount of data.

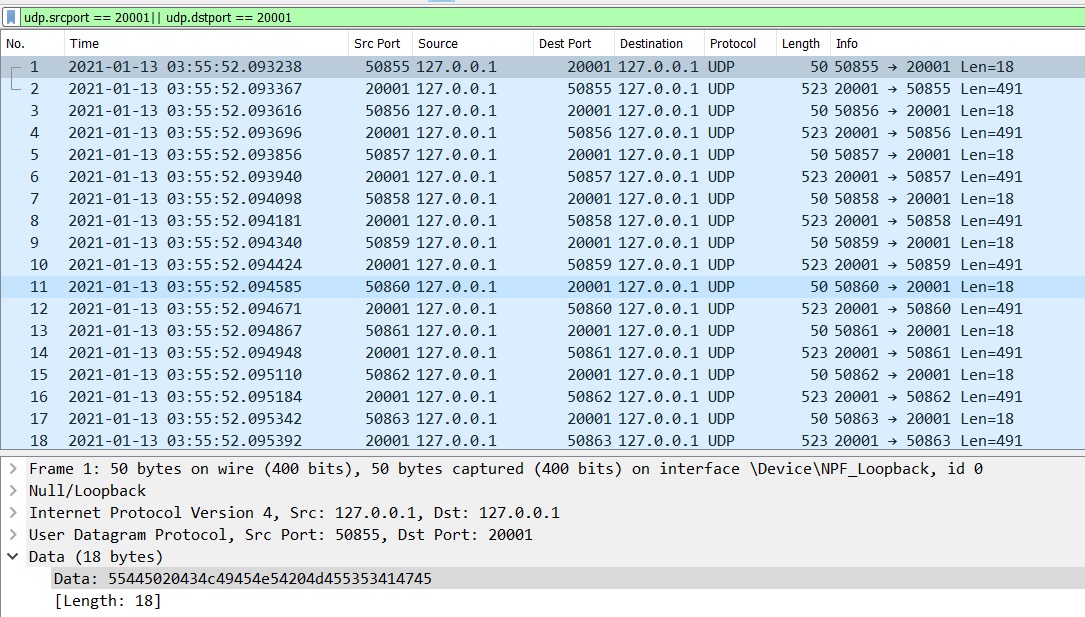
****

Figure 10 UDP test B packet history Beginning

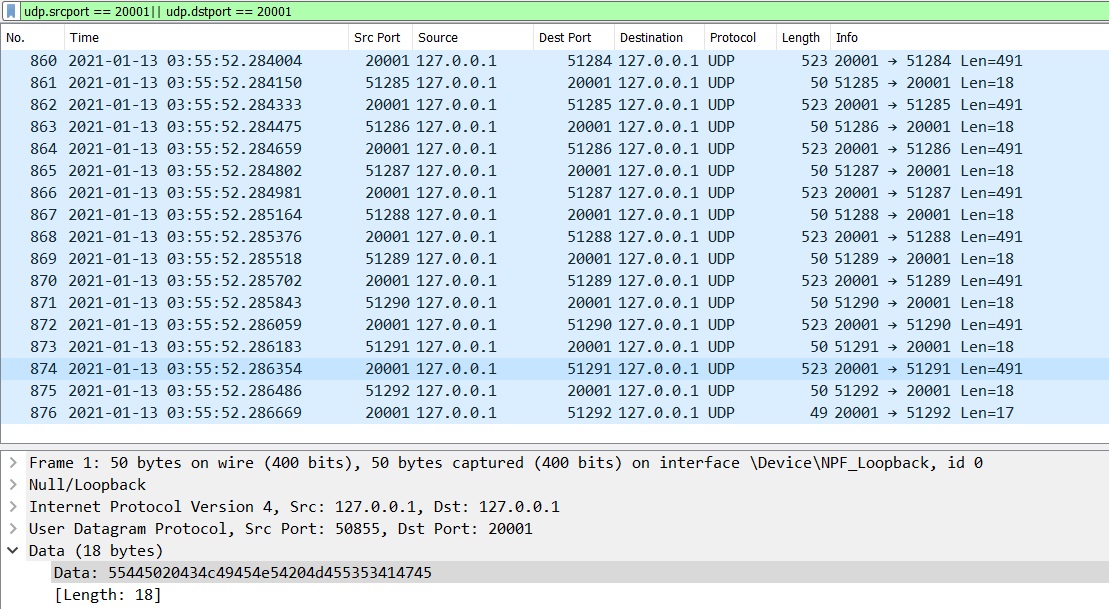
****

Figure 11 UDP test B packet history Ending

We can also see that unlike TCP, UDP doesn’t use transmission control methods. And so that it’s average packet per ms graph is much more linear and straight.

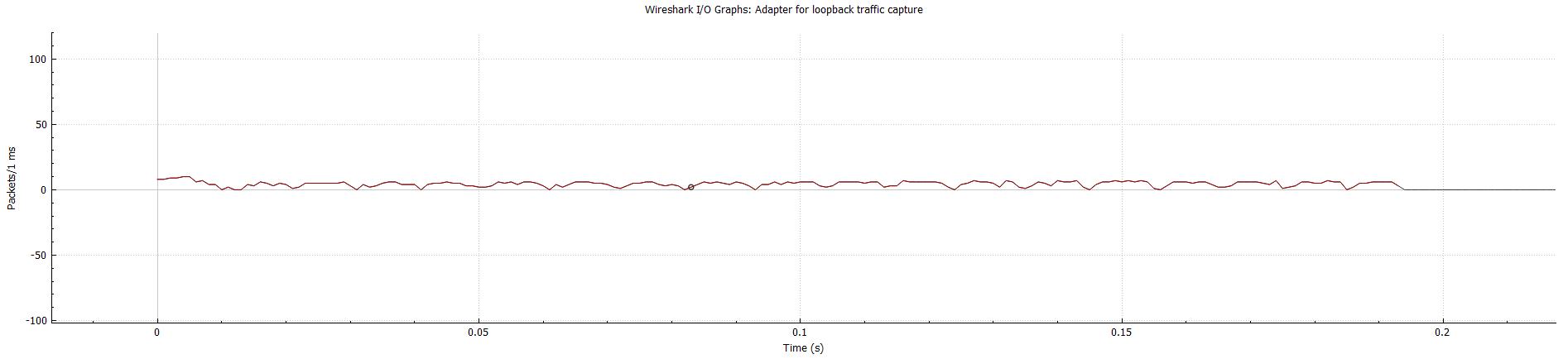


Figure 12 UDP test B packets send per ms

**4.3 Test C**

**TCP**

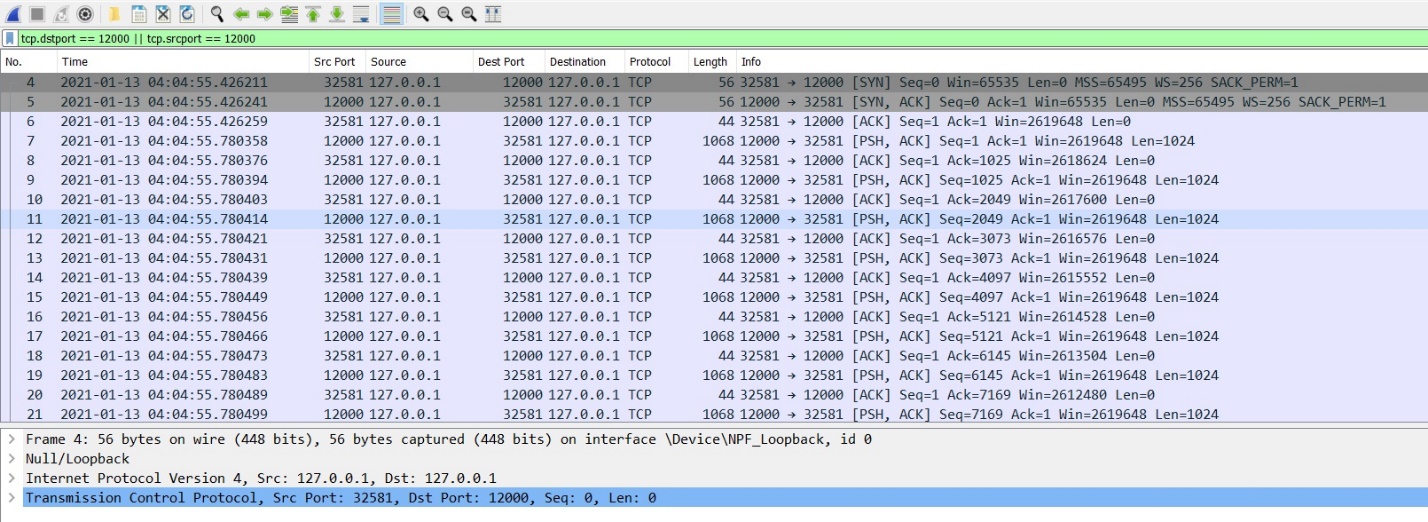
In test C, there is a significant difference than the other tests, in transmission speed, due to window size. Since TCP is transmitting the segments according to its window size, when that window is filled, it has to wait the very first segment to be ACK’ed in the window to proceed with the remaining segments. Since the size of the last file is very big, the waiting process is much more visible than the other tests in the graph.****

Figure 13 TCP test C packet History Beginning

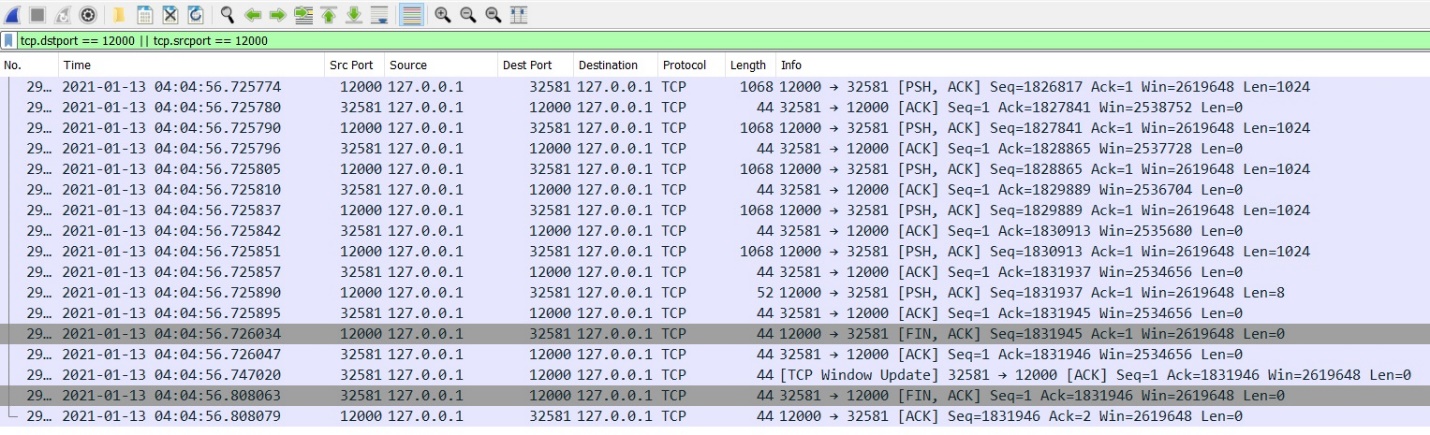


Figure 14 TCP test C packet History Ending

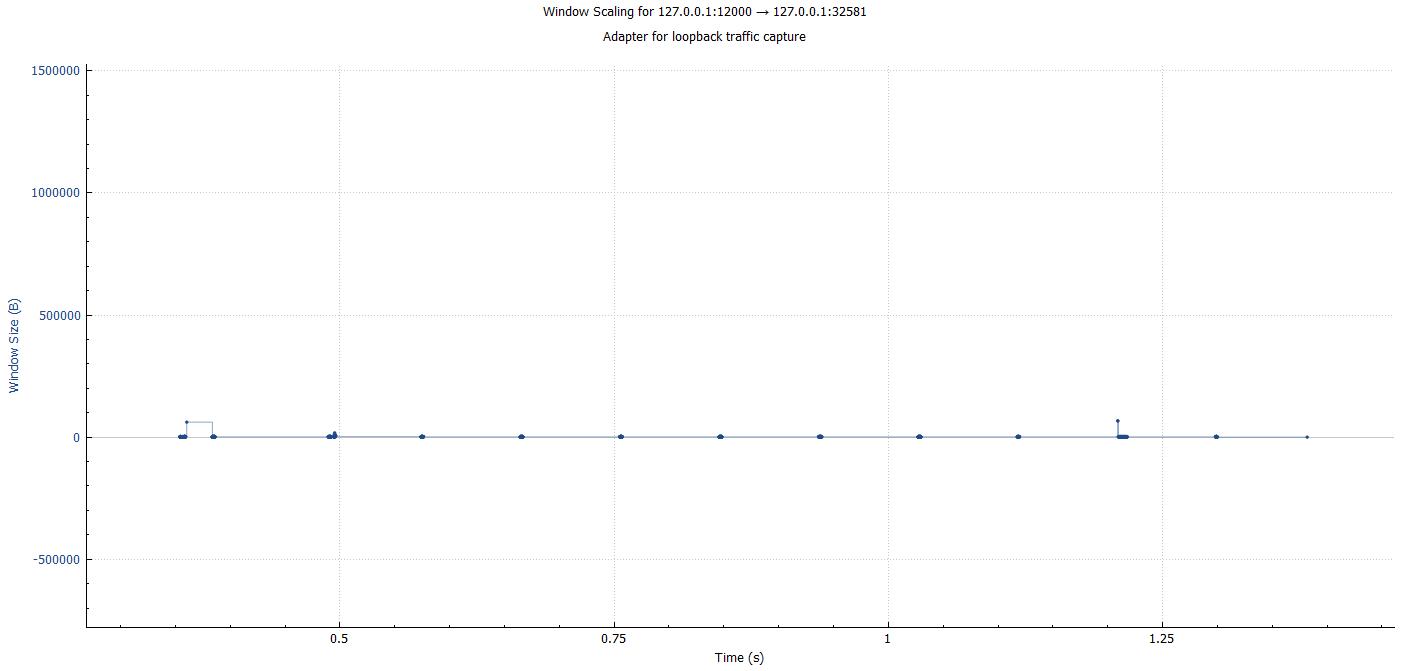


Figure 15 TCP test C windows scaling

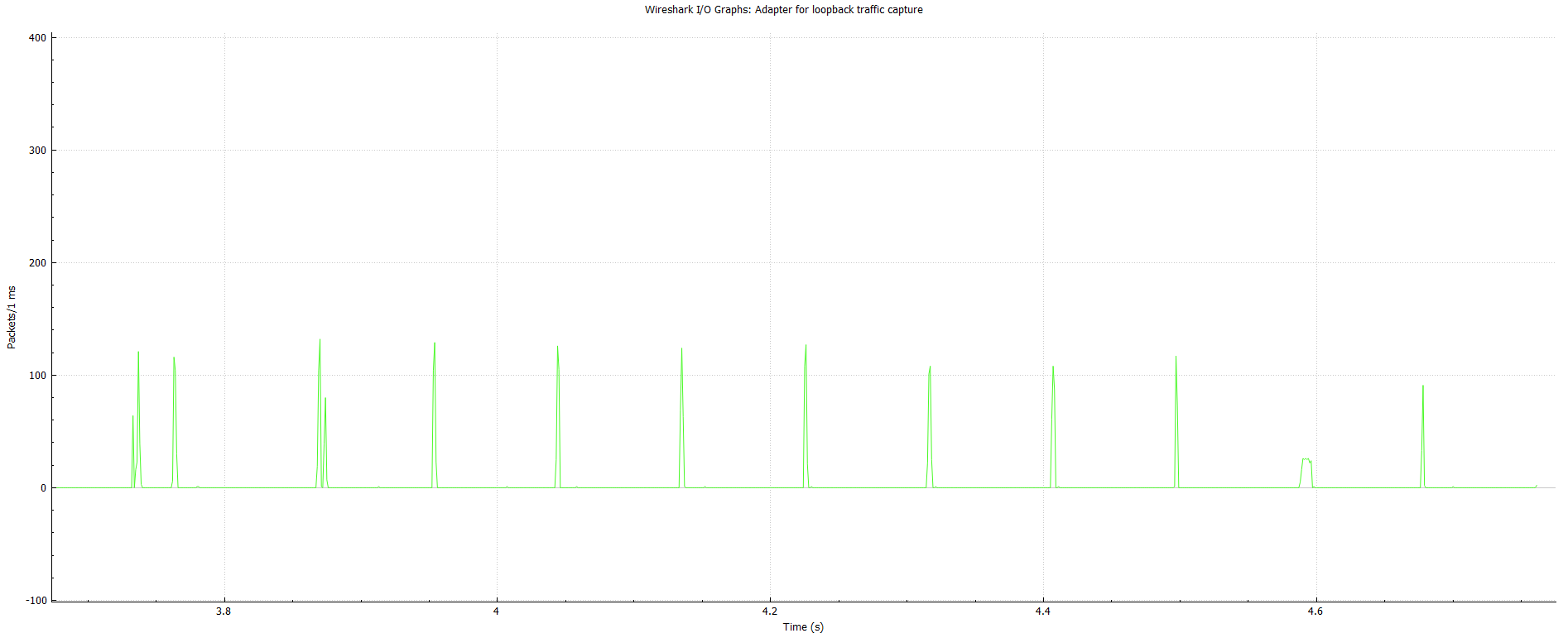


Figure 16 TCP test C packets send per ms

**UDP**

For UDP the size of the file made the effect of it’s datagram size extremely obvious. Making this transmission longest one out of all the other test we have conducted, it lasted more than 2.5 seconds which is considerable in the world of networks.

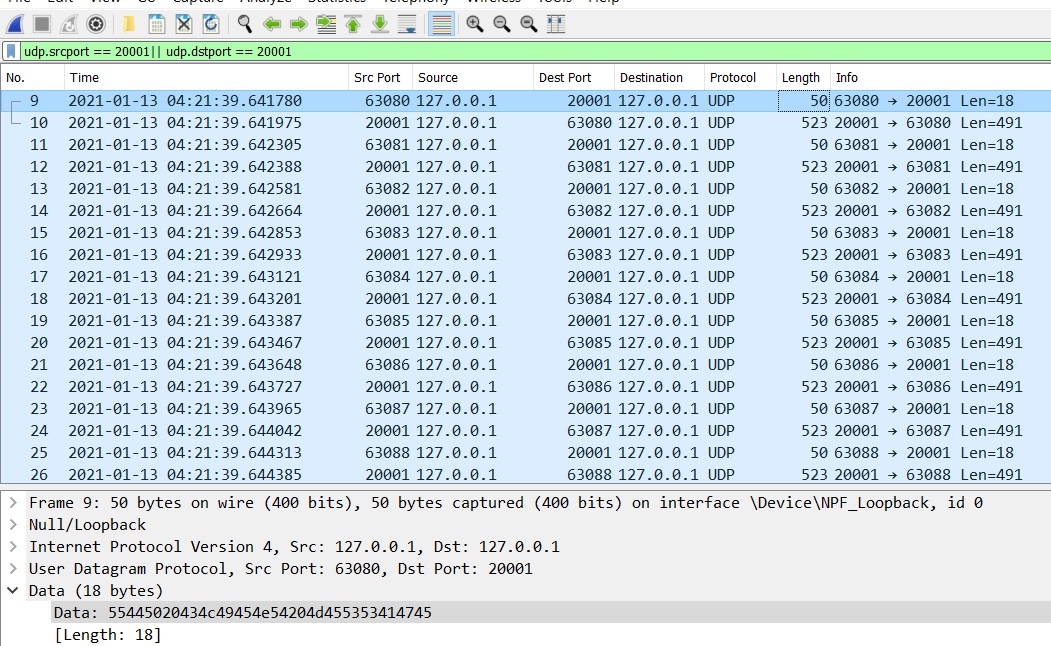


Figure 17 UDP test C Packet History Beginning

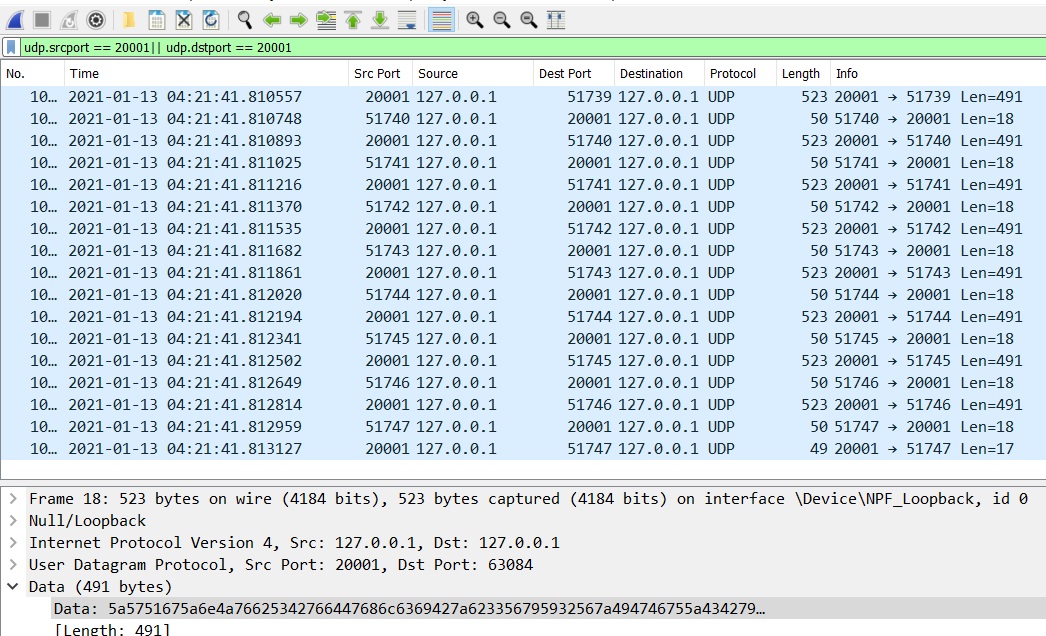


Figure 18 UDP test C Packet History Ending

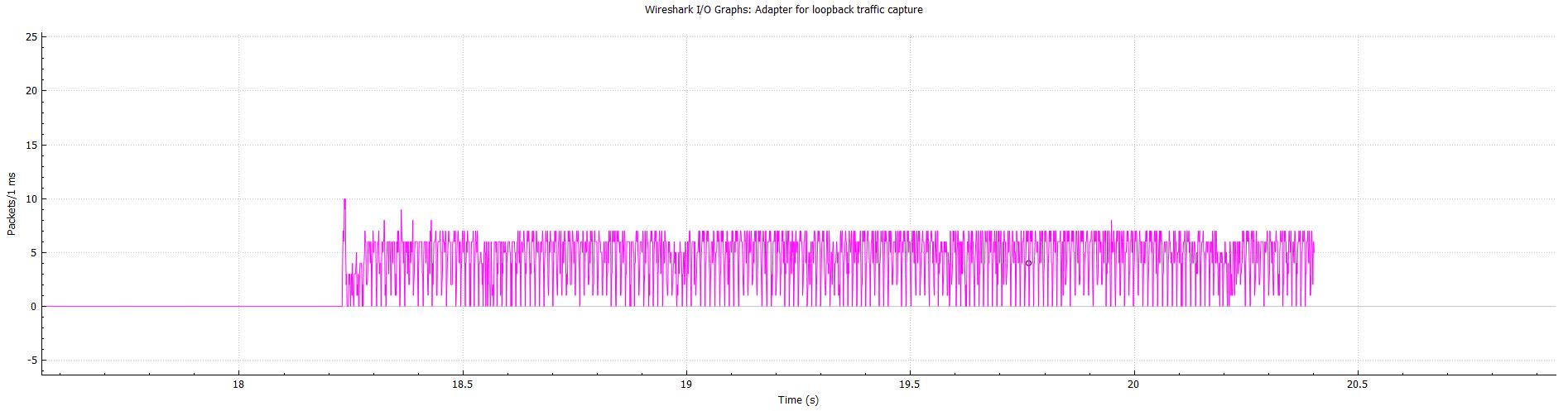


Figure 19 UDP test C Packets Send per ms.

1. **Conclusion**

In all of the tests we have conducted, we saw that there has been no corruption between the data sent from the server and the output file that the client receives. We think this is because of the fact that we do not create these connections through a proper, public network. If that had been the case there would be interference from the outside elements.

We believe that these test and this report can bring insight to how these protocols work in practice