University of Science and Technology in Zewail City

CIE 447

Reliable Transport Protocol

May, 2023

Computer Networks Project

Ahmed Ibrahem - 201902227
Amr Elmasry - 201901202
Eman Allam - 201900903
Youssef Elshabrawy - 201900667

Contents

1 Introduction

The reliable transfer of data is a fundamental aspect of networking. Reliable data transfer protocols aim to provide upper layer entities with a reliable channel for transmitting data, overcoming challenges such as packet loss, corruption, and out-of-order packets. This project focuses on implementing the Go-Back-N (GBN) protocol, one of the reliable data transfer protocols studied. The goal is to augment the unreliable User Datagram Protocol (UDP) with the GBN protocol to provide reliability services.

The purpose of the code is to transfer a file over a network using the Go-Back-N protocol. The code operates on a PNG image file that is read in binary mode and turned into hexadecimals for transmission.

2 Implementation Details

2.1 Timeout Interval Choice

In order to determine an appropriate timeout value for reliable communication, the calculation of the timeout duration plays a crucial role. A common approach involves considering the round-trip times (RTTs) of transmitted packets. By collecting a series of measured RTTs, we can calculate the average RTT as the sum of all RTTs divided by the total count. Furthermore, to capture the variation in RTTs, the standard deviation (dev rtt) is computed. This involves squaring the difference between each RTT and the average RTT, summing up these squared differences, dividing by the total count, and finally taking the square root. With the average RTT and dev rtt in hand, the timeout duration can be derived using the provided function "calculate timeout rtt".

```
def calculate_timeout_rtt(rtt, dev_rtt):
    timeout = rtt + 4 * dev_rtt
    return timeout
```

Figure 1: Calculate Timeout Function

This function incorporates the average RTT and four times the standard deviation (4 * dev rtt) to determine an appropriate timeout value. The resulting timeout duration can be employed in the system for setting timeout thresholds for retransmissions, ensuring reliable communication in the face of network delays and variations.

```
# Calculate average RTT
average_rtt = sum(rtt_list) / len(rtt_list)

# Calculate deviation of RTT
dev_rtt = (sum((rtt - average_rtt) ** 2 for rtt in rtt_list) / len(rtt_list)) ** 0.5
```

Figure 2: Calculate Dev RTT Function

Then we put a minimum value for it which can be controlled, **0.2 seconds in our case**.

2.2 Window Size Choice

In the context of reliable data transmission, the choice of an optimal window size is critical for achieving efficient and fair network utilization. The Additive Increase Multiplicative Decrease (AIMD) algorithm offers a popular approach for dynamically adjusting the window size. AIMD operates on the principle of increasing the window size additively when the network conditions are favorable and decreasing it multiplicatively when congestion is detected. This algorithm aims to strike a balance between maximizing network throughput and avoiding network congestion. By starting with a small window size and gradually increasing it, AIMD allows the sender to explore the available bandwidth. Upon detecting packet loss or experiencing congestion, the window size is reduced multiplicatively to alleviate network congestion and prevent further packet loss. This dynamic adjustment of the window size based on network feedback allows AIMD to adapt to changing network conditions and optimize the overall transmission performance, ensuring efficient and fair utilization of network resources. In our case we chose the following:

- Initial window size = 1
- Minimum window size = 1
- Maximum window size = 20

2.3 MSS Choice

The choice of maximum segment size (MSS) ultimately depends on the specifics of the network (network condition, available bandwidth and packet loss rate) and the application requirements. In some cases we might need to use a suitable MSS without deeply knowing those specifics and requirements such as our case, so based on some research we figured out that the MSS suitable range for our case is between 1000 bytes and 1500 bytes as it needs to be small enough to avoid fragmentation while still being large enough to avoid excessive overhead from too many small packets.

3 GBN Vs SR

- In GBN, if a packet is lost during transmission, the receiver notifies the sender by sending an acknowledgment that this packet is lost, but the sender re-sends all packets in the current window even if they were transmitted before, not just the lost one. In other words, rather than just sending the lost packet again, the sender now sends the current window of packets.
  While in SR when a packet is lost during transmission, the receiver asks the sender to resend just the lost packet by sending a selective acknowledgement (ACK).
- In GBN, the receiver only stores packets that are received in order and discards any out-of-order packets.
  While in SR the receiver buffers all received packets, whether they are in-order or out-of-order which allows the receiver to send acknowledgement for each packet individually and wait to receive any lost packet while buffering the packets after it in order.
- In GBN, the sender uses cumulative acknowledgement which means that if packet number 3 is acknowledged, then any packet before this packet is received successfully at the receiver even if the sender does not receive its acknowledgement.

While in SR, the sender deals with individual packets acknowledgement not the cumulative acknowledgement.

- **SR is more efficient** than GBN because it only re-transmits the lost packets, which means less bandwidth is used and less re-transmitted number of packets. However GBN is used in application that doesn't have the facility of buffering at the receiver at it may be complex sometimes.

4 Results

4.1 Wireshark

We have successfully completed this step of the implementation, and the accompanying screenshots provide visual evidence of the process. The captured packet listings displayed in Wireshark showcase the exchange of packets between the sender and the receiver. The screenshots verify the accurate reception of the test files and depict the transfer time. Specifically, the highlighted first and last packets of each file, along with their corresponding time stamps, offer clear indications of the successful transmission. Furthermore, the screenshots prominently display the presence of the 0xFF trailer train, confirming that it serves as the final packet in the sequence. These visual representations provide comprehensive documentation of the implemented solution, reinforcing the reliability and efficiency of the data transfer process.

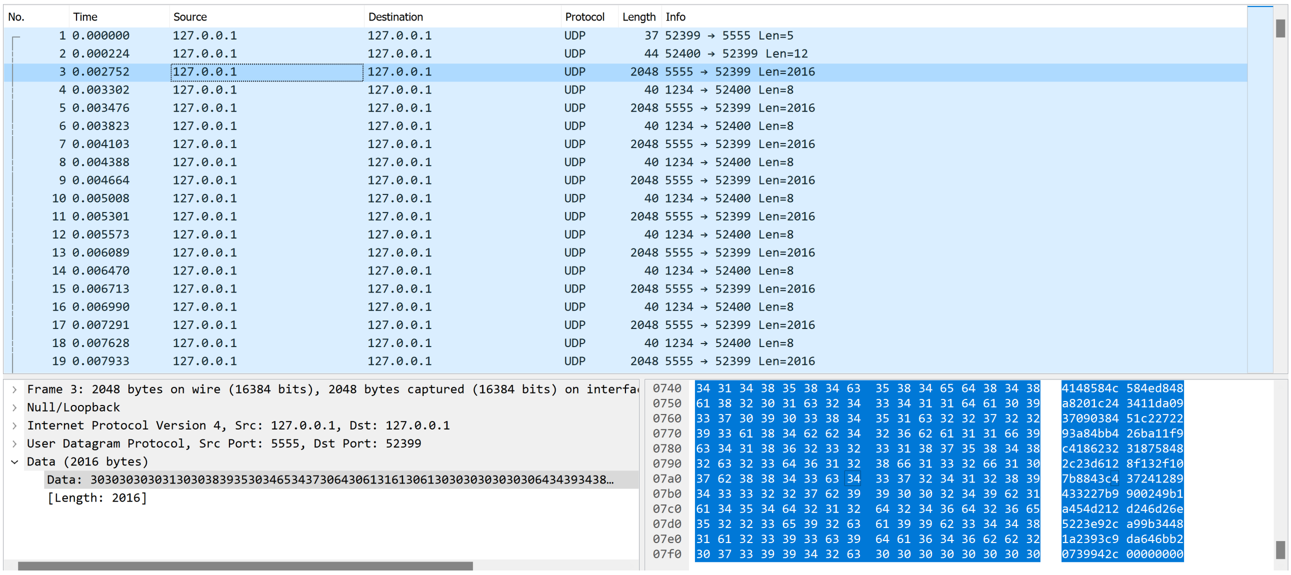


Figure 3: First UDP Packet Transmitted

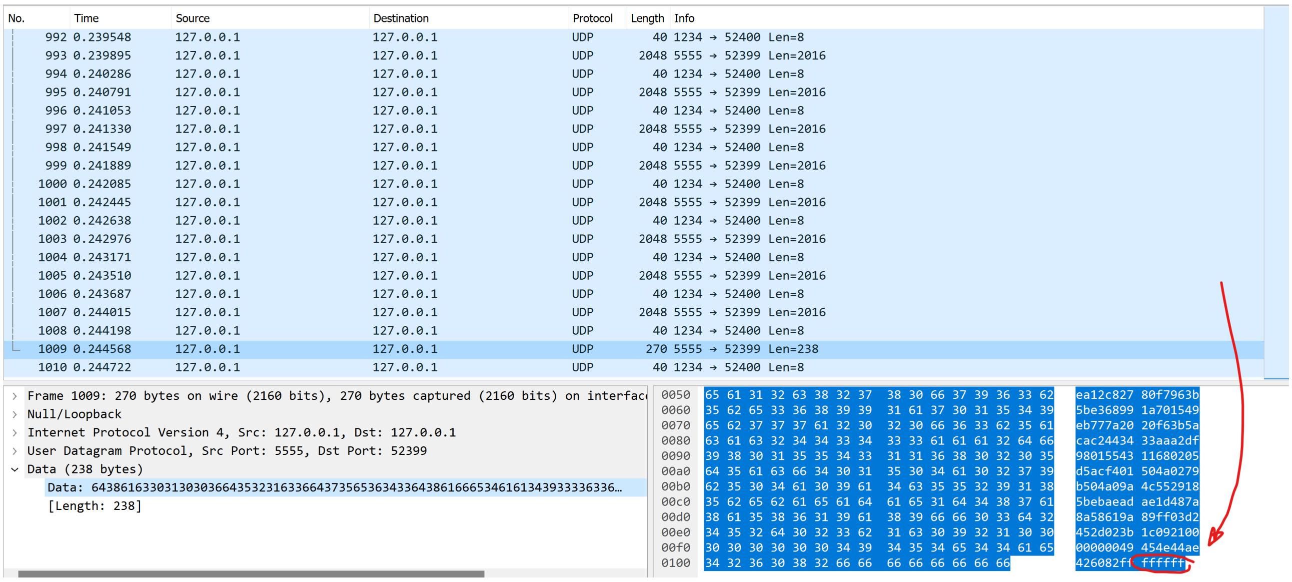


Figure 4: Last UDP Packet Transmitted

Elapsed Time = 0.244568 - 0.002752 = **0.241816 seconds**

4.2 Sweeping Window Size

Here we are sweeping maximum size from 1, 5, and 20 to see the effect in a lossless channel so we can decide.

```
==========Stats===========
start time: 1683924536.8600764
end time: 1683924537.4031143
elapsed time: 0.5430378913879395
# of packets: 504
# of bytes: 503111
average transfer rate (byte/s): 926474.9439751044
average transfer rate (packet/s): 928.1120304733003
==========================
```

Figure 5: CWND = 1

```
==========Stats===========
start time: 1683924757.2930765
end time: 1683924757.7199597
elapsed time: 0.42688322067260742
# of packets: 504
# of bytes: 503111
average transfer rate (byte/s): 1178568.2257721121
average transfer rate (packet/s): 1180.6507625338038
==========================
```

Figure 6: CWND = 5

```
==========Stats===========
start time: 1683924872.5845428
end time: 1683924872.992232
elapsed time: 0.407689332962036l3
# of packets: 504
# of bytes: 503111
average transfer rate (byte/s): 1234054.853347977
average transfer rate (packet/s): 1236.2354352963469
==========================
```

Figure 7: CWND = 20

The results obtained clearly demonstrate that a window size of 1 yields the fastest response compared to window sizes of 5 and 20. The data analysis reveals that smaller window sizes result in quicker acknowledgment and transmission of packets. This can be attributed to the reduced congestion and improved efficiency of the transmission process. But keep in mind that this is because the channel is lossless, If we put loss, we get the same result as well, that is because the needed ACK is served quickly and does not require extra time sending packets that will be discarded.

4.3 Plots and Statistics - Lossless

4.3.1 Small File

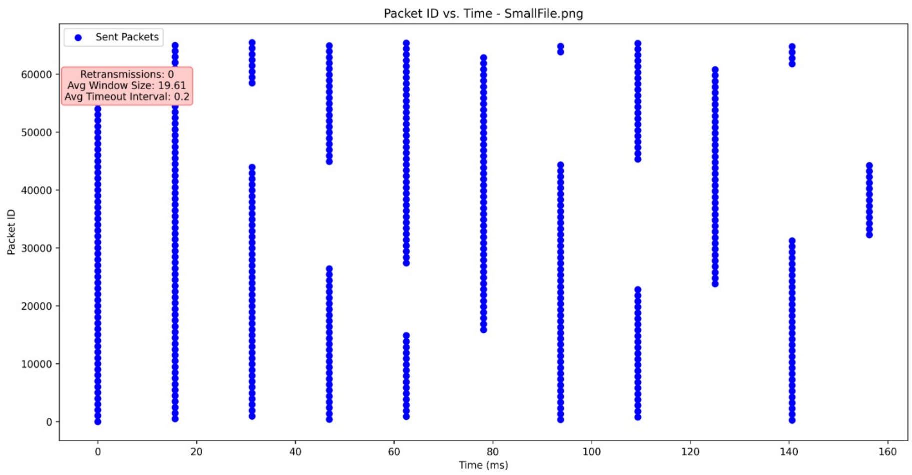


Figure 8

```
=========Terminating...
==========Stats===========
start time: 1683920428.580414
end time: 1683920428.736656
elapsed time: 0.15624189376831055
# of packets: 504
# of bytes: 503111
# of retransmissions: 0
average transfer rate (byte/s): 3220077.4572411287
average transfer rate (packet/s): 3225.7673524322245
==========================
```

```
True
==========Stats===========
start time: 1683920428.5651712
end time: 1683920428.7522984
elapsed time: 0.18712711334228516
# of packets: 504
# of bytes: 503111
average transfer rate (byte/s): 2688605.5741143734
average transfer rate (packet/s): 2693.356355463594
==========================
```

Figure 9

4.3.2 Medium File

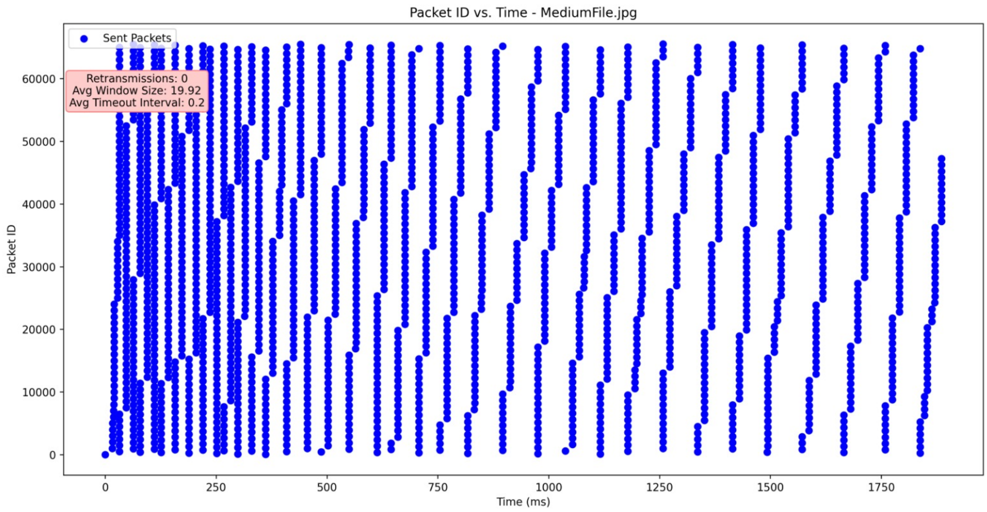


Figure 10

```
lower is 28240, upper is 49240
========Terminating...
=========Stats==========
start time: 1683920495.2643855
end time: 1683920497.1500912
elapsed time: 1.8857057094573975
# of packets: 2342
# of bytes: 2341781
# of retransmissions: 0
average transfer rate (byte/s): 1241859.2086003898
average transfer rate (packet/s): 1241.9753454922184
=========================
```

```
========Terminating...
True
=========Stats==========
start time: 1683920495.2643855
end time: 1683920497.1500912
elapsed time: 1.8857057094573975
# of packets: 2342
# of bytes: 2341781
average transfer rate (byte/s): 1241859.2086003898
average transfer rate (packet/s): 1241.9753454922184
=========================
```

Figure 11

4.3.3 Large File

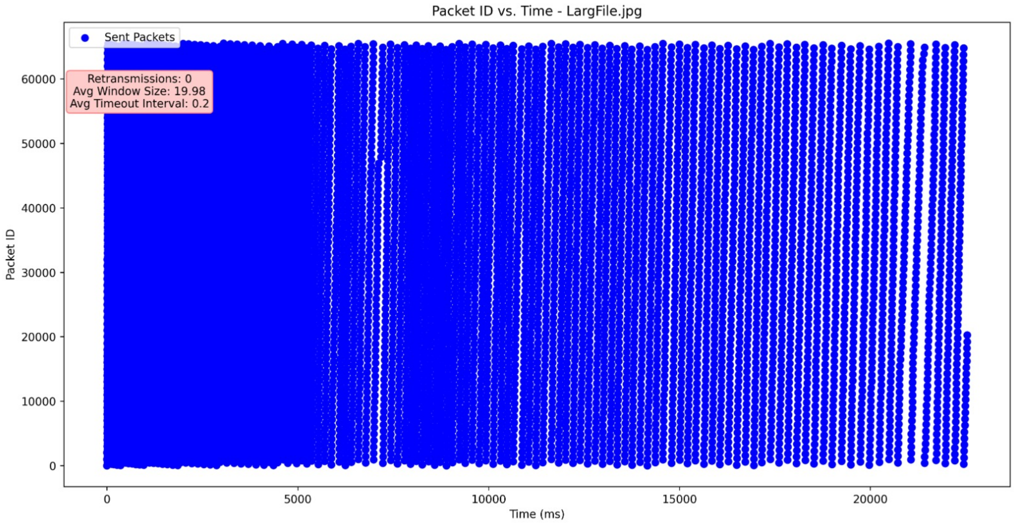
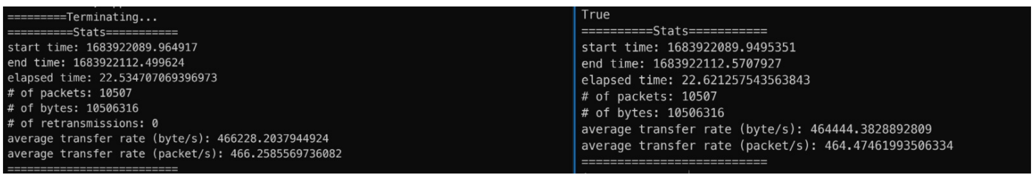


Figure 12



```
========Terminating...
=========Stats==========
start time: 1683922089.964917
end time: 1683922112.499624
elapsed time: 22.534707069396973
# of packets: 10507
# of bytes: 10506316
# of retransmissions: 0
average transfer rate (byte/s): 466228.2037944924
average transfer rate (packet/s): 466.2585569736082
=========================
```

```
True
=========Stats==========
start time: 1683922089.9495351
end time: 1683922112.5707927
elapsed time: 22.621257543563843
# of packets: 10507
# of bytes: 10506316
average transfer rate (byte/s): 464444.3828892809
average transfer rate (packet/s): 464.47461993506334
=========================
```

Figure 13

4.4 Plots and Statistics - 10% Loss

4.4.1 Small File

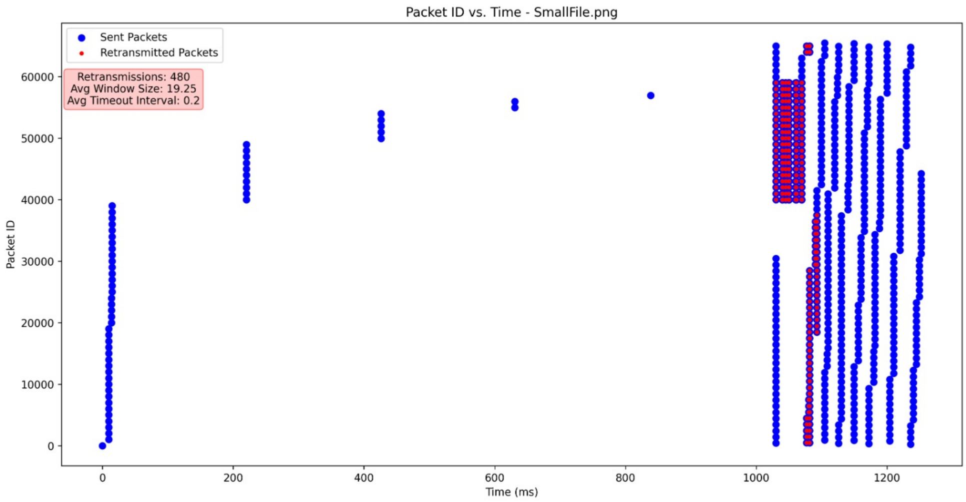


Figure 14

```
==========Stats==========
start time: 1683916816.0032566
end time: 1683916817.2555377
elapsed time: 1.2522811889648438
# of packets: 504
# of bytes: 503111
# of retransmissions: 480
average transfer rate (byte/s): 401755.6156184697
average transfer rate (packet/s): 402.4655200774953
=========================
```

```
==========Stats==========
start time: 1683916816.0032566
end time: 1683916817.2555377
elapsed time: 1.2522811889648438
# of packets: 504
# of bytes: 503111
average transfer rate (byte/s): 401755.6156184697
average transfer rate (packet/s): 402.4655200774953
=========================
input message:
```

Figure 15

4.4.2 Medium File

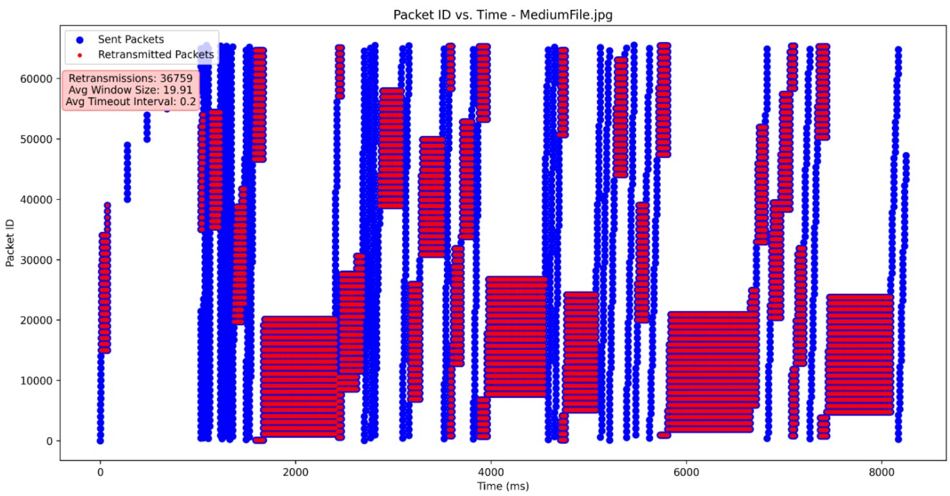


Figure 16

```
=========Terminating...
==========Stats===========
start time: 1683917058.4467268
end time: 1683917066.6947403
elapsed time: 8.248013496398926
# of packets: 2342
# of bytes: 2341781
# of retransmissions: 36759
average transfer rate (byte/s): 283920.6071889212
average transfer rate (packet/s): 283.9471590368414
==========================
```

```
True
==========Stats===========
start time: 1683917058.4394822
end time: 1683917066.6947403
elapsed time: 8.255258083343506
# of packets: 2342
# of bytes: 2341781
average transfer rate (byte/s): 283671.44629008893
average transfer rate (packet/s): 283.6979748368393
==========================
input message:
```

Figure 17

4.4.3 Large File

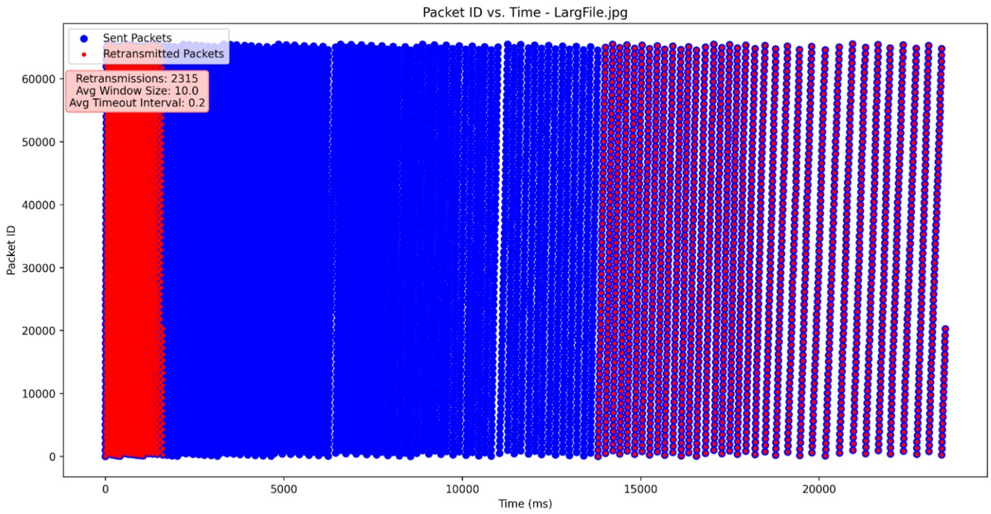


Figure 18

```
=========Terminating...
True
==========Stats===========
start time: 1683918095.4341462
end time: 1683918375.8025184
elapsed time: 280.36837220191956
# of packets: 10507
# of bytes: 10506316
average transfer rate (byte/s): 37473.2567639028
average transfer rate (packet/s): 37.47569641140879
==========================
```

Figure 19

5 Receiver Attack

5.1 Describe the proposed attack

5.1.1 Attack I (implemented)

The proposed attack involves modifying the sender's behavior in a way that disrupts the receiver and causes it to wait indefinitely. By exploiting weaknesses in the protocol specification, the attacker can make the sender keep the session open without sending any data. As a result, the receiver remains idle, waiting for the 0xFF trailer that signifies the end of the transmission. This attack can be particularly harmful as it ties up system resources and prevents the receiver from processing other tasks or accepting new connections.

```
while True:
    pass
```

To mitigate such an attack, several measures can be implemented. One approach is to enforce a timeout mechanism on the receiver side. If the receiver does not receive any data or acknowledgments within a specified time, it can terminate the session and free up resources for other tasks. Additionally, the sender's behavior can be monitored, and if it consistently fails to send data or acknowledgments within a reasonable time frame, it can be identified as malicious and blocked.

5.1.2 Attack II

Another proposed attack is to send malicious data instead of the real data by intercepting the sent data (man in the middle attack), as there is no check on the authenticity or the integrity of the received data.

5.2 Suggested Updates

1. **3-ACK fast re-transmission**: Instead of waiting to send the whole window again, to reduce re-transmissions, we can listen fot 3 similar ACKs to stop transmitting the window and proceed from the needed ACk (as in TCP)

2. **Authentication:** To combat spoofing attacks, the protocol may call for mutual authentication between the sender and receiver.

3. **Strengthened Encryption:** To prevent data access by unauthorised parties and eavesdropping, the protocol may employ better encryption methods.

5.3 Local and International Legal Frameworks

1. A US federal statute known as the Computer Fraud and Abuse Act (CFAA) makes hacking and unauthorised access to computer systems illegal.

2. The General Data Protection Regulation (GDPR) is a European Union law that aims to safeguard people's personal information and right to privacy within the EU.

3. An international convention known as the "Cybercrime Convention" aims to harmonise national cybercrime legislation and serve as a foundation for international collaboration in the fight against cybercrime.

4. Depending on the severity of the offence and the jurisdiction in which it occurred, different penalties may be imposed for breaking these laws. While violating the GDPR can result in hefty fines and other penalties, breaking the CFAA can result in fines and imprisonment in the United States.

5.4 Economic and Societal Impact of Freely Spreading Tools That Can Disrupt Network Communication

The freely spreading tools that can interfere with network connectivity can have a huge negative influence on the economy and society. These tools can be utilised maliciously for espionage, data theft, and cyberattacks. Critical infrastructure may be harmed, businesses and organisations may be disrupted, and personal information may be compromised. In addition to that it may encourage cybercrime, which could have a large negative economic impact. A McAfee analysis found that cybercrime costs the world economy more than 600 billion dollar every year.

6 Conclusion

In conclusion, this project aimed to implement a Go-Back-N protocol for reliable data transmission over a network. The protocol was successfully implemented, allowing the sender to divide the data into segments, transmit them to the receiver, and handle acknowledgments for reliable delivery. The project incorporated features such as packet segmentation, acknowledgment handling, timeout calculation, and window size management using the Additive Increase Multiplicative Decrease (AIMD) algorithm.

The results of the project demonstrated the successful transfer of test files, with the receiver receiving the data correctly and verifying the transfer time. The Wireshark packet capture screenshots showcased the exchange of packets between the sender and receiver, highlighting the first and last packets of each file. Additionally, the screenshots verified the presence of the 0xFF trailer train as the final packet, indicating the completion of the transmission.

Furthermore, the project analyzed the impact of different window sizes on the transmission performance. The findings revealed that a smaller window size of 1 resulted in the fastest response, with quicker acknowledgment and transmission of packets. Conversely, larger window sizes introduced congestion and potential delays, resulting in longer transfer times. This highlighted the significance of selecting an optimal window size for efficient and reliable data transmission.

Overall, this project provided valuable hands-on experience in implementing a Go-Back-N protocol, understanding its key components, and exploring the effects of different parameters on transmission performance. It demonstrated the importance of reliable data transmission protocols in network communication and offered insights into optimizing protocol parameters for enhanced efficiency.