Final Project in Networking

Page

Moshe Ofer

208821652

Matanya Barzilay

208982991

System Characterization

System Overview

This project's goal is to demonstrate how a client connects to several servers and carries out different activities. It demonstrates how a client connects to an application server via RUDP or TCP, acquires an IP address from a DHCP server, then asks the DNS for the FTP server IP, and uses an FTP server to read and download files by entering a username and password.

The architecture of the script is created to step-by-step demonstrate these procedures, giving a clear knowledge of each phase of the client-server interaction. The client first requests an IP address from a DHCP server, then gets the IP by creating a DNS request, after which it connects to the application server and then the FTP server.

How To Run the Code:

To run this code on Linux, open a terminal, navigate to the directory where the file is located, and run the following command:

```
$ python dhcp_server.py
```

```
$ python DNS_server.py
```

```
$ python FTP_server.py
```

This will run the 3 servers, with the default API.LO_HOST value, and 3 photos of 'animals' for the FTP server.

Alternatively, you can specify the server IP address or hostname using the -H/--host argument:

You can specify the prompt and the number of photos for the FTP server, as well.

```
$ python dhcp_server.py -H 192.168.1.1
```

```
$ python DNS_server.py -H 192.168.1.2
```

```
$ python FTP_server.py -p dogs -n 10
```

This will run the script with DHCP_server_IP set to 192.168.1.1 and DNS_server_ip set to 192.168.1.2

For the FTP server, it will download from "unsplash.com" API 10 random photos of dogs.

Finally run the client.py script which communicate with all 3 servers.

```
$ python client.py -[protocol]
```

In the protocol option you can choose between 'tcp' and 'rudp'.

```
$ python client.py -p rudp
```

System Functionality

Function	Purpose
Download()	Receives data over a reliable UDP or TCP connection and saves it to a file. Used for downloading files from a server.
connect_tp_FTP()	Connects to an FTP server, logs in, and downloads a file from the server. Used for retrieving files from an FTP server.
connect_to_DNS()	Sends a DNS query to a DNS server and receives a response. Used for resolving domain names to IP addresses. The client requests the FTP's IP.
connect_to_DHCP()	Sends a DHCP discover message, receives a DHCP offer message, and sends a DHCP request message to obtain an IP address from a DHCP server. Used for **dynamically** assigning IP addresses to devices on a network. The client gets from the server the DNS IP as well as the net mask etc.

The table above shows all the functions which the client has. The order which the client intrecte with those functions is from bottom to up.

Components and Subsystems

DHCP_server

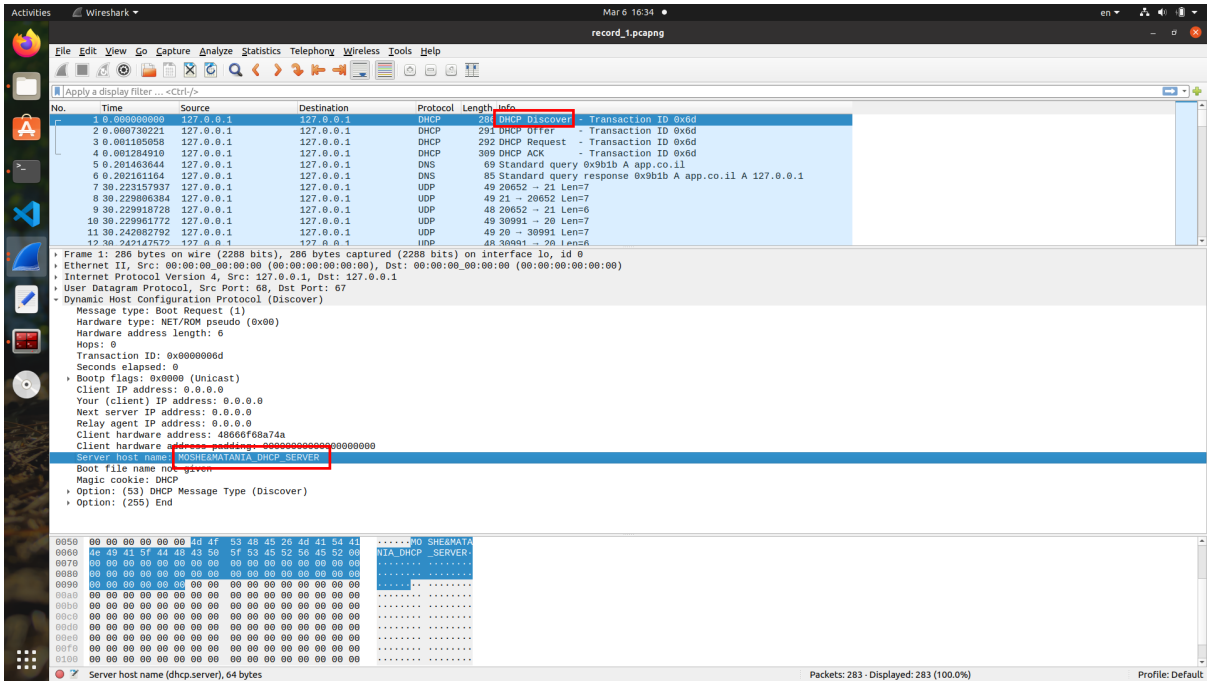
Software application that sends DHCP discover, offer, and request messages to obtain an IP address.

Properties:

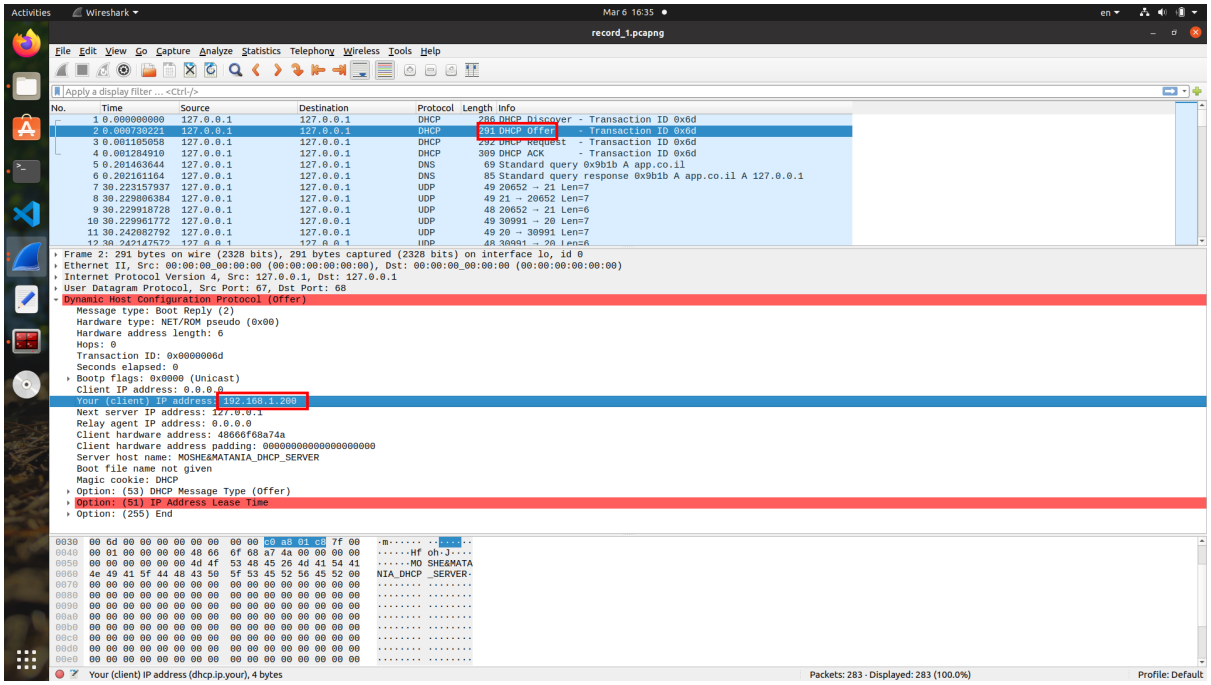
To prevent two clients from receiving the same IP, the server keeps a list of available and offered IPs. When a client sends it a DHCP discovery packet, it listens for it and sends back DHCP offer packets in which the client can find an IP address to use. The server sends a DHCP acknowledgement packet to the client to confirm the lease of the IP address after the client accepts the given IP address by sending a DHCP request packet to the server.

With a 24-hour lease time, the server dynamically assigns IP addresses from a pool of accessible IPs (86400 seconds). The server does not provide a fresh IP address if a client requests the same IP address more than once; instead, it delivers an acknowledgement packet. With the acknowledgement packet, the server also transmits the client the gateway, subnet mask, DNS server IP, and lease time options. In addition, each client must resend a request each half time of the lease time. So, the server has running thread which keep logged in clients by sending them acknowledge for each request. If the timeout occur the IP is return to the available IPs list and might be assigned to different device.

A timeout thread is defined in the script to remove the IP address from the list of available IPs after two seconds, and a Client Info class is defined to hold the transaction ID, MAC address, and assigned IP address of the client.



The DISCOVER packet contains the server's name.



The Offer packet contains the offered IP '192.168.1.200'.

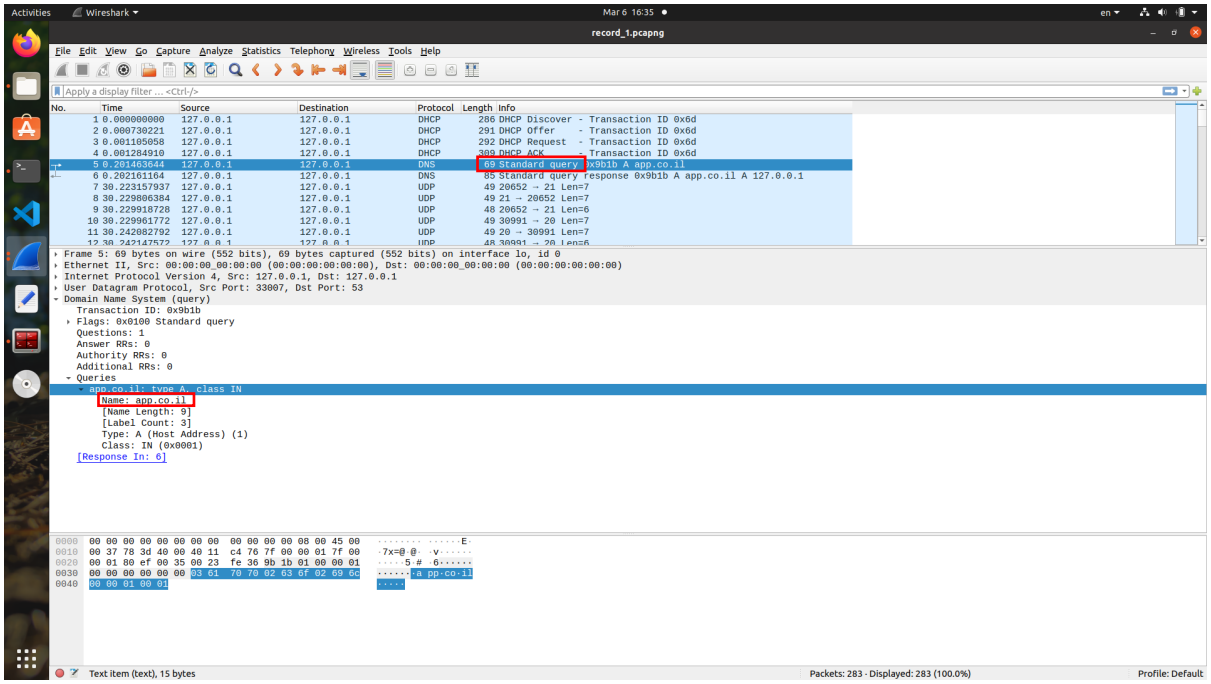
DNS_server

Software application that sends DNS responses and receives DNS queries.

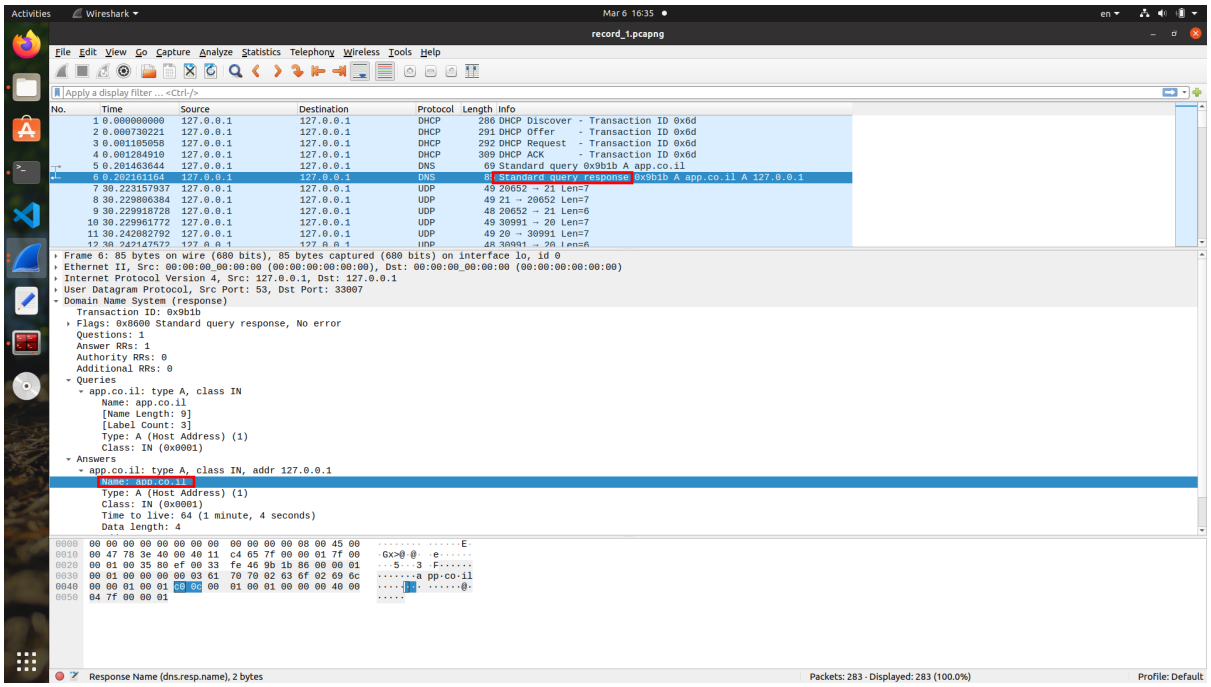
Properties:

The DNS gets a request from the client and check, the domain name is extracted from the request and checked to see if it is in the list of recognized domains, (in our case there is just "app.co.il", the app server).
If so, it generates a DNS response packet with the domain's IP address and forwards it back to the client. It adds the client to a list of pending clients.

If the domain is not known, it adds the client to a list of pending clients and forwards the request to Google's DNS server. This script offers a simple implementation of a DNS server that can resolve IP addresses for domain names. Nevertheless, it is not a fully functional DNS server solution.



The Query contains the domain name: 'app.co.il'.



The Response contains a pointer to the domain name from the Query.

FTP_Server

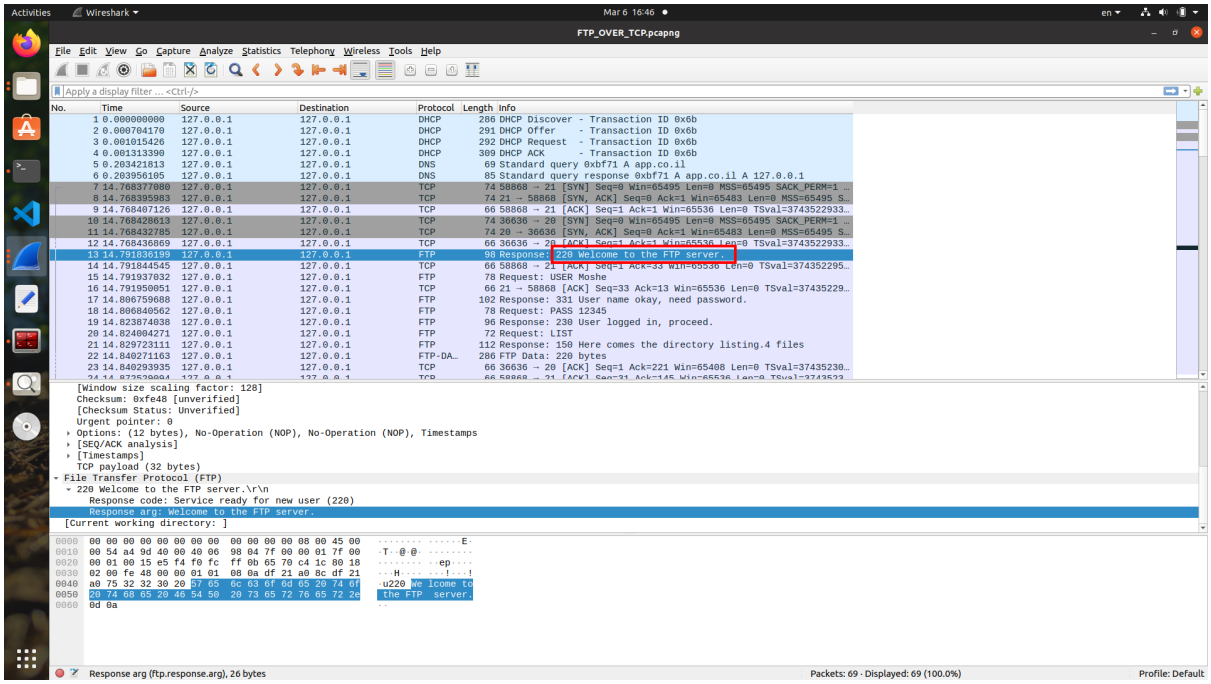
Software application that connects to an FTP server and downloads files.

Properties:

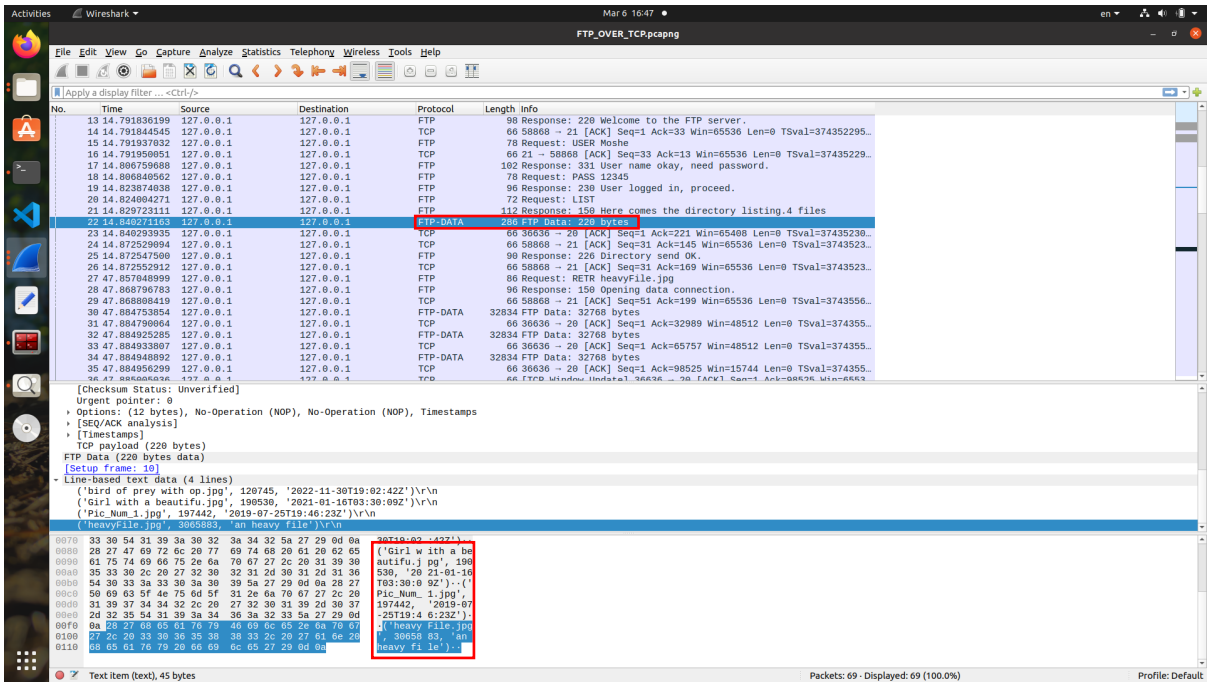
First the server downloads from "unsplash.com" free images API, couple of images into a new directory named 'images'. Then the server listens on TCP and UDP simultaneously on port 21 for incoming client connections and then accepts data connections on port 20 for file transfers.

The server implements several FTP commands, such as USER, PASS, LIST, RETR, and QUIT, using handlers in the HANDLERS lookup table. The connection_establish function is the main function that handles the communication with the client.

The remove_with_timeout function is a helper function used for removing a dictionary item after a specified timeout period. It is used in the handle_USER and handle_PASS functions to keep track of the data connection associated with a user after successful authentication. Similarly, the handle_QUIT function removes the data connection associated with a user after the user has disconnected.



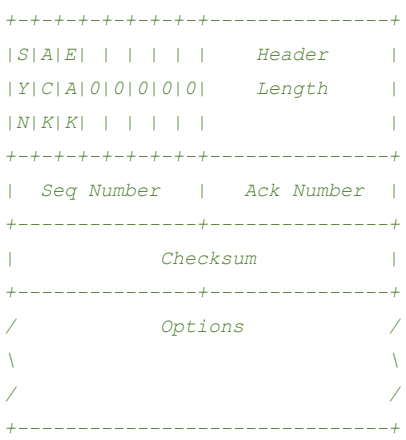
The FTP Packet contains the welcoming message.



This packet contains the list of the files from the server. Notice that this was sent via the data socket, on port 20, while the messages were sent on port 21.

Our RUDP Protocol

The RUDP header we have created:



SYN: Set for initialize a new connection. The header will include relevant data Like the threshold.

ACK: Set for acknowledge. The Ack Number indicates the next expected packet to be send.

EAK: Set for EACK message which includes the lost packets.

Checksum: Error Correction Technique. For Reliable transportation.

Options: Extend the header with more data.

SYN uses for adding connection information.
EACK uses for adding lost packets.

The Algorithm for sending the data:

1. Receive the data from the sender.
2. Divide the data into packets of the required size, and assign each packet an header with a sequence number.
3. Store each packet in a list and mark each packet's number in a set.
4. Send packets within the window size, removing the sent packet's number from the set.
5. Wait for ACK or EACK to confirm the receipt of packets.
    - If an ACK is received, **adjust the window size** according to the protocol's rules.
    - If an EACK is received, First add the **lost packets** into the set and **resend** them, second reduce the window size by halving it.
6. Repeat steps 4-5 until the set is empty which means all packets have been acknowledged.

The Algorithm for receiving the data:

TRHEAD-1 the receiver:

1. Create a **priority queue** q to store the received packet according to the sequence numbers. It's done to overcome the out of order problem.
2. Receive packet using recv() and store it in data.
3. If the packet is not a duplicate and the checksum is valid, add the packet to the priority queue q, update the size so far, and add the packet number to the shared set SharedSet.
4. Once the size of received data is equal to or greater than expected size, exit the loop.
5. Iterate through the priority queue q and append each packet's payload.

THREAD-2 sends Acknowledge.

1. Initialize the last_received variable to 0, which will store the sequence number of the last received packet.
2. While the thread-1 is still running:
    a. Wait for a time period equal to the window size multiplied by 0.07 (a heuristic value used to determine the waiting time).
    b. Check the shared set to identify which packets have been lost (i.e., not received yet).
    c. Update the last_received variable to the highest sequence number that has been received.
    d. If no packets have been lost, send an acknowledgment (ACK) packet to the sender to indicate the highest sequence number received successfully.
        i. If the window size is less than the threshold, double the window size.
        ii. Otherwise, increase the window size by one.
    e. If packets have been lost, send an extended acknowledgment (EACK) packet to the sender to indicate the lost packets.
        i. Reduce the window size by half.
        ii. If the receive thread is still running, create and send the EACK packet to the sender.
        iii. If the receive thread has completed, send an ACK packet to the sender instead.
3. Return the received data to the caller.

The receiver also sends ACKs to the sender in response to the received packets. The receiver maintains a sliding window of received packets using the SharedSet. The receiver

periodically sends ACKs to the sender based on the sliding window's status. If there are no lost packets, the receiver sends a regular ACK to the sender with the last received packet's sequence number plus one. If there are lost packets, the receiver sends an extended ACK (EACK) with the list of lost packet numbers.

Q1: How the system overcomes packet loss?

Both algorithms use acknowledgments and retransmissions to overcome lost packets. The sending algorithm keeps track of which packets have not yet been acknowledged using a set, while the receiving algorithm notifies the sender of lost packets using an EACK packet. In both cases, the window size is adjusted to optimize data transmission.

Q2: How the system overcomes latency problems?

The system does not directly address latency problems, but it is designed to minimize their impact.

Packets are sent using the sending algorithm inside a specified window size. The protocol's rules specify the window size, which can be modified in response to receiver feedback. The window size lowers the effect of latency on data transmission by allowing many packets to be delivered before waiting for acknowledgments.

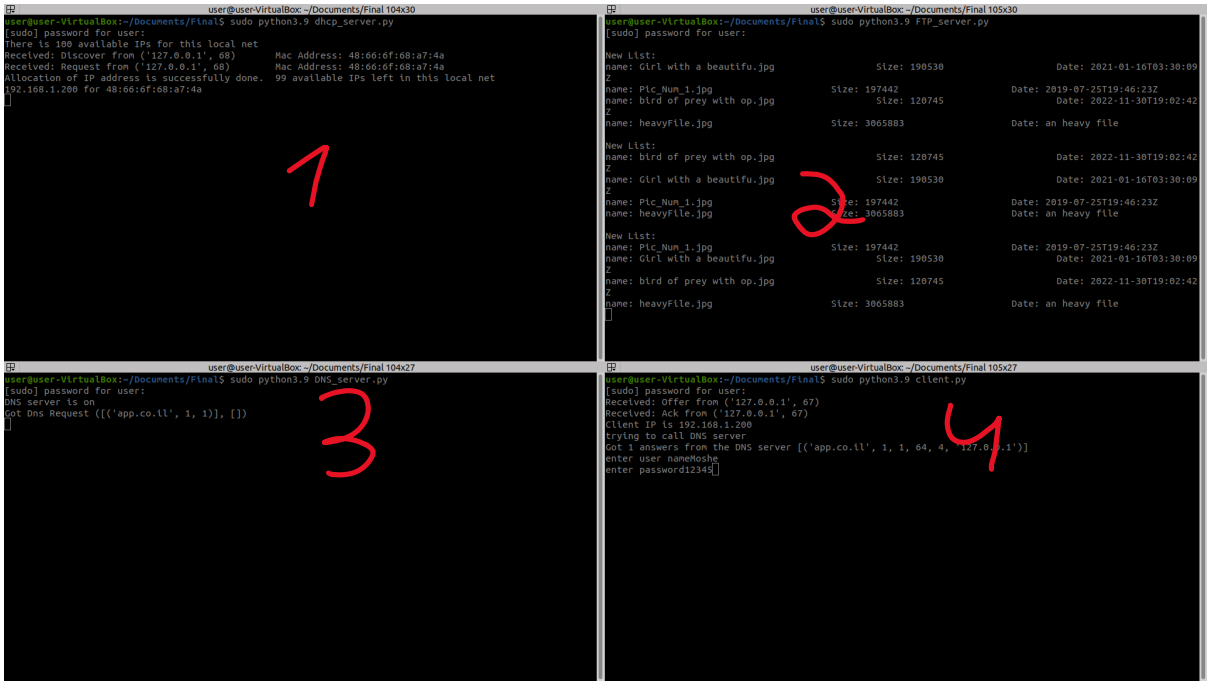
Overall, by using acknowledgments, retransmissions, and window sizes, the algorithms can adapt to different network conditions and overcome latency problems that may arise during data transmission.

RUDP vs TCP

In our project we would like to explore the RUDP protocol in an aspect of reliability, latency, dealing with congestion on the network. To do so we will compare our implementation to the well-known TCP protocol, which has built in congestion control and flow control.

So, we have two options to run our code, using TCP or our RUDP.
one can compare the two protocols, by running our code twice, each time with different protocol.
We will test our protocol in a situation where some packets lost, which demonstrates a bad condition network.

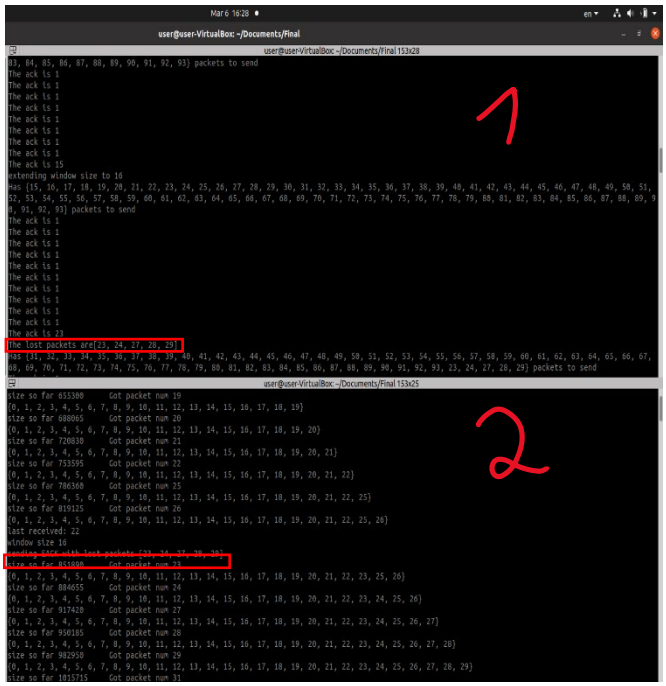
Sending via RUDP

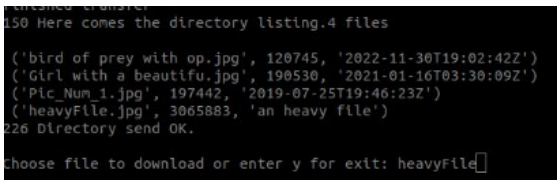


At this picture we can see running of our system.

Terminal:

1. The DHCP server. The IP '192.168.1.200' allocated for the client's MAC address.
2. The FTP server. 3 lists of the files which found in the server.
3. The DNS server. Got DNS request from the client for 'app.co.il'.
4. The Client. Got his IP and the DNS IP from the DHCP server, then Got IP address for the FTP server from the DNS server.

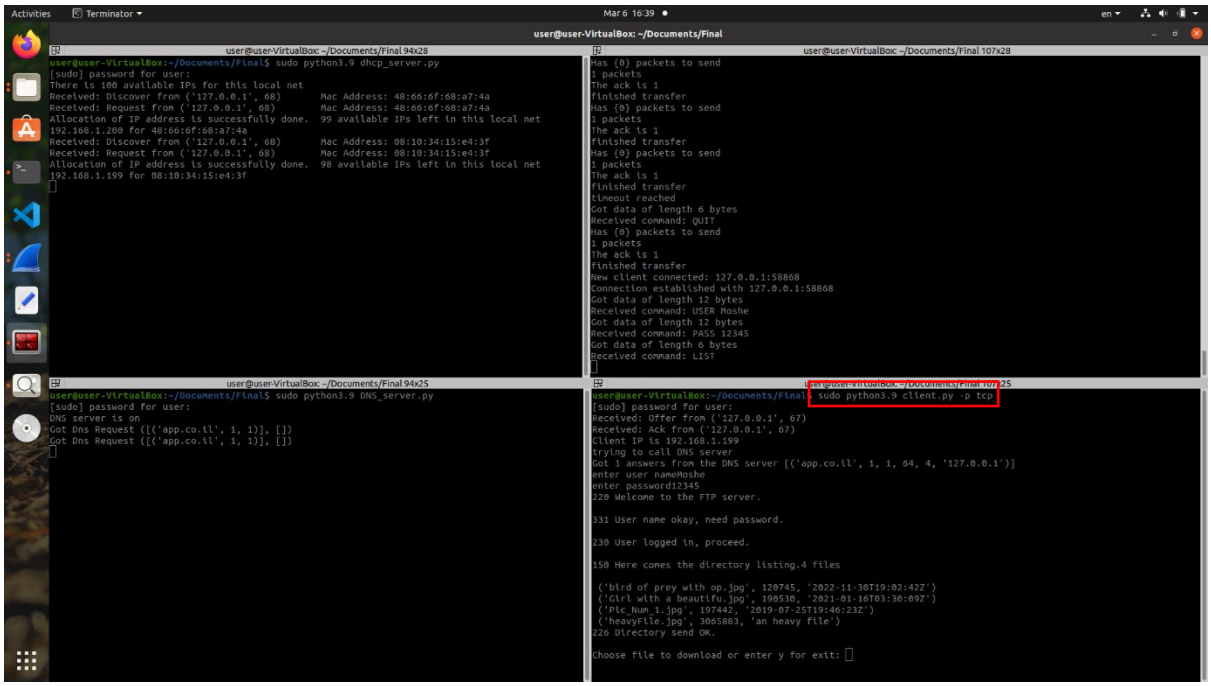




After logged in, the client Gets a list of all the files in the FTP server with their size and last changed date and need to pick one to download it.

1. The server which Sends the file. After sending 16 packets, Got an EACK of missing packets number, [23, 24, 27, 28, 28] for resending them.
2. The Client which receives the file. Sent the EACK.

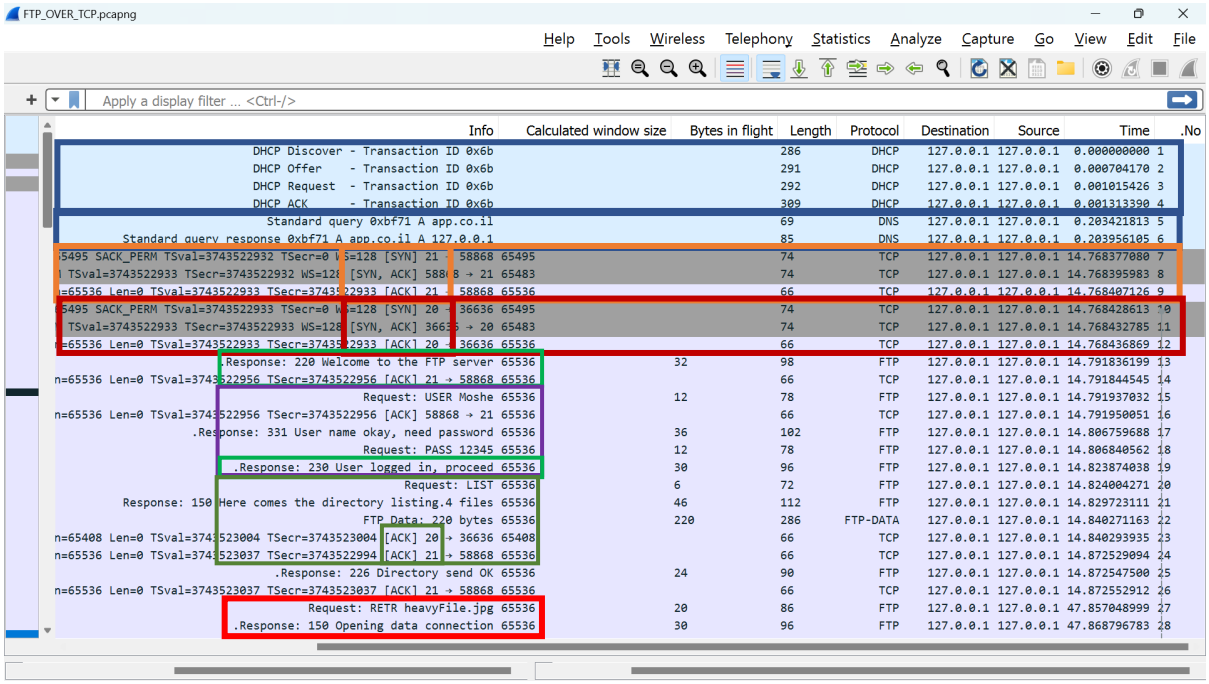
Sending via TCP



This time we run the client on TCP protocol.

Intercept

TCP



Let`s follow the Steps:

1. The client gets Ip address from DHCP server. (More details on DHCP chapter)

2. then the client sends request to DNS server to get the FTP server`s address.

3. the client send a SYN request to start connection with the ftp server over port 21.

After the Three-Handshakes succeeded, we can see that, the client sends a new SYN request to FTP server over port 20 to create a socket for data row. (Why? See the ftp chapter)

"yes.", we got a Welcome message from server.

4. Now all that left for the client to get access to server`s data is to verify credentials, so the client sends username and password.

"yes.", we got a User Logged in message from server.

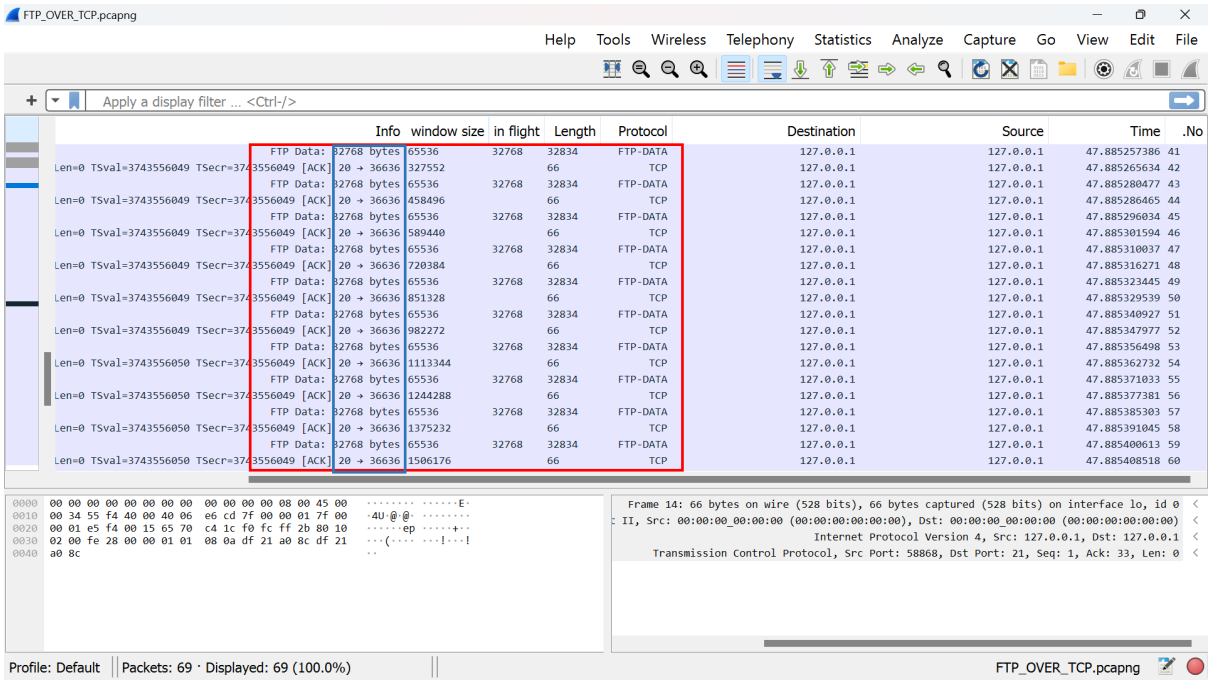
Notice:

Requests between server and client are done by specific command, we can see the USER command for username, PASS for password, LIST for getting the files option from server, RETR for downloading the specific file, etc.

Reply is made by magic code like, 150 for opening data connection or file status OK, 226 for requested file action successful, 230 for User logged on, etc.

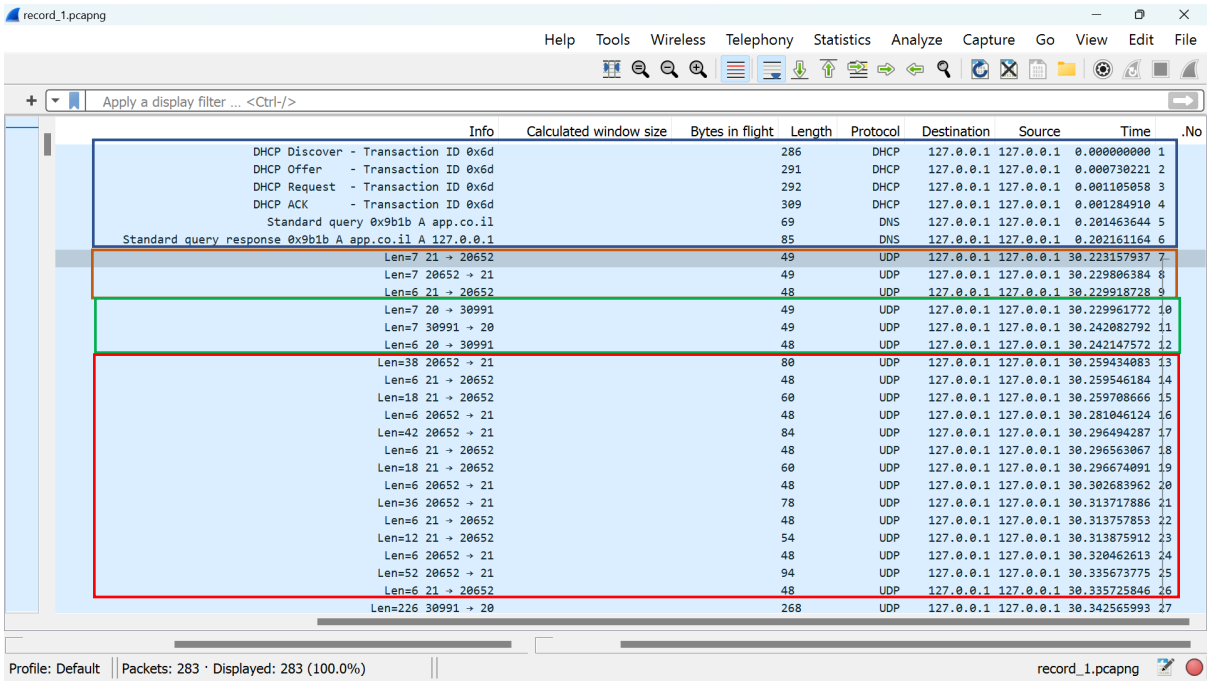
5. we can see the client sends the LIST command for the file list on server and get reply 150 for getting the request over port 20 and reply 220 for directory successfully sent over port 21.

and finally, the RETR command for Downloading specific file.



6. The important thing to notice here is that communication with server has moved from port 21 to port 20, to split the sending and receiving socket.

Rudp



Steps:

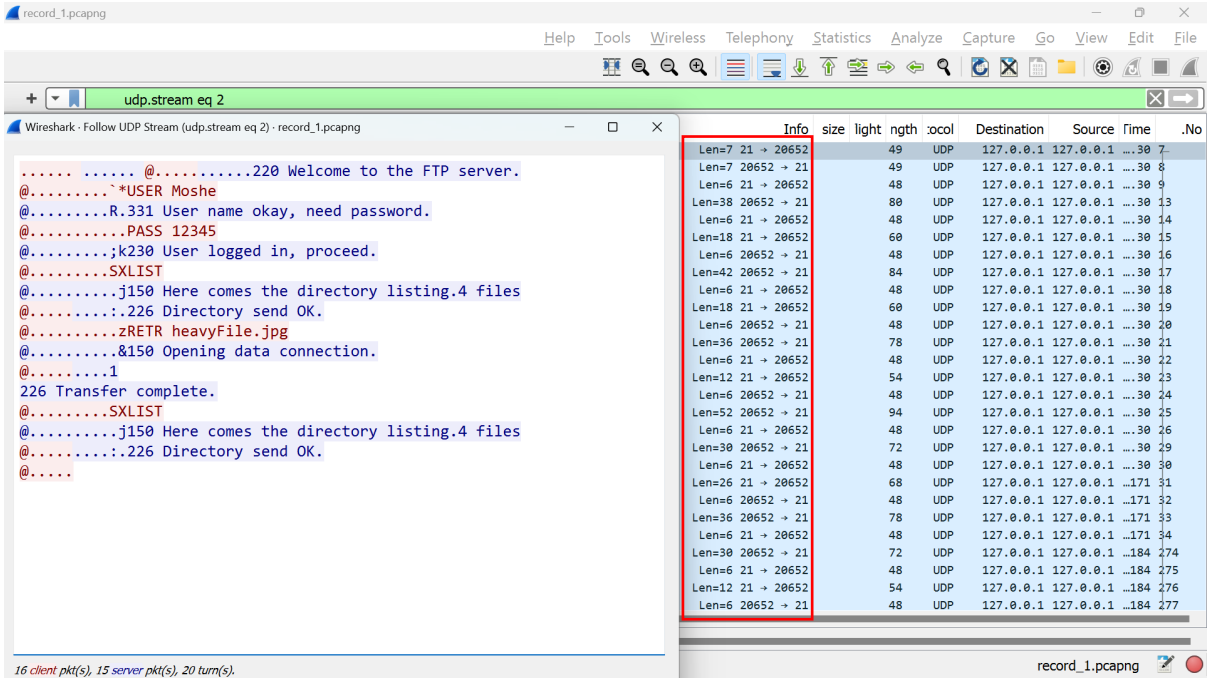
1.The client gets an Ip address (DHCP) and FTP server`s Ip (DNS), like tcp

2.the client send a SYN request with the info for communication over port 21.

3. after the server send an ack, the client initials a new socket over port 20.

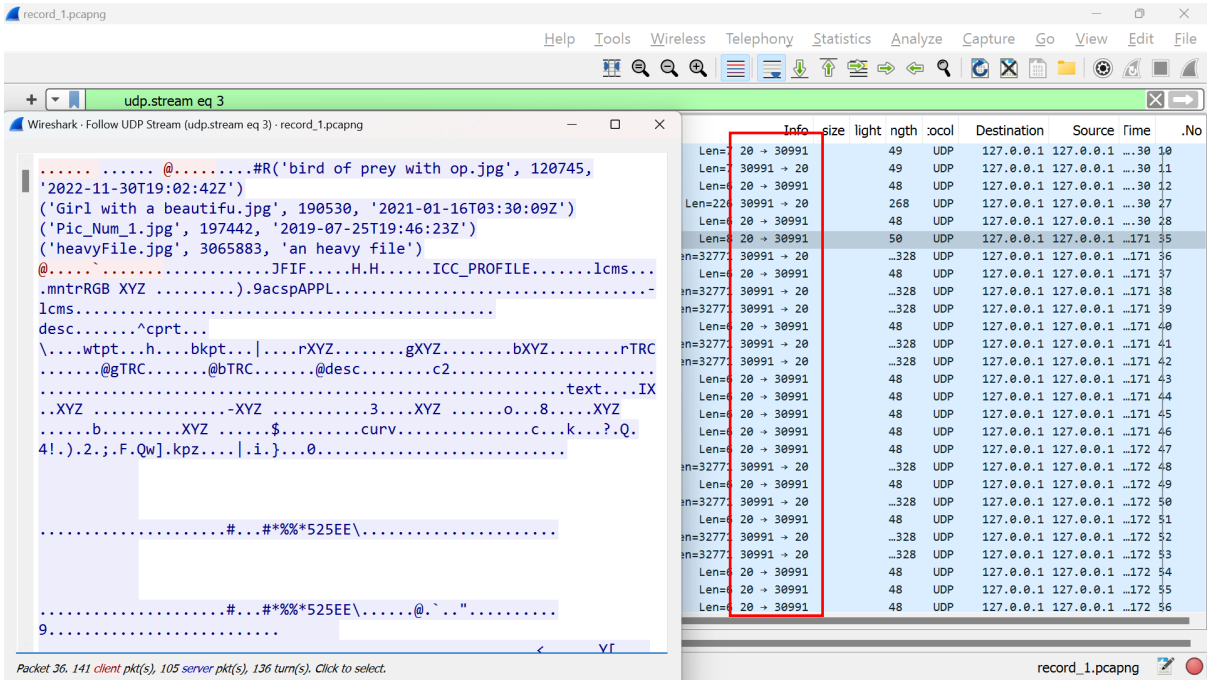
4. then the client communicates with server over port 21 and received the data over port 20.

More detail on the following pictures.



Stream eq 2: is the data over port 21, as we can see in screenshot

On the left we can see the data between server and client, ack package which received every time data sent. (Red for client message, and blue for tcp message).

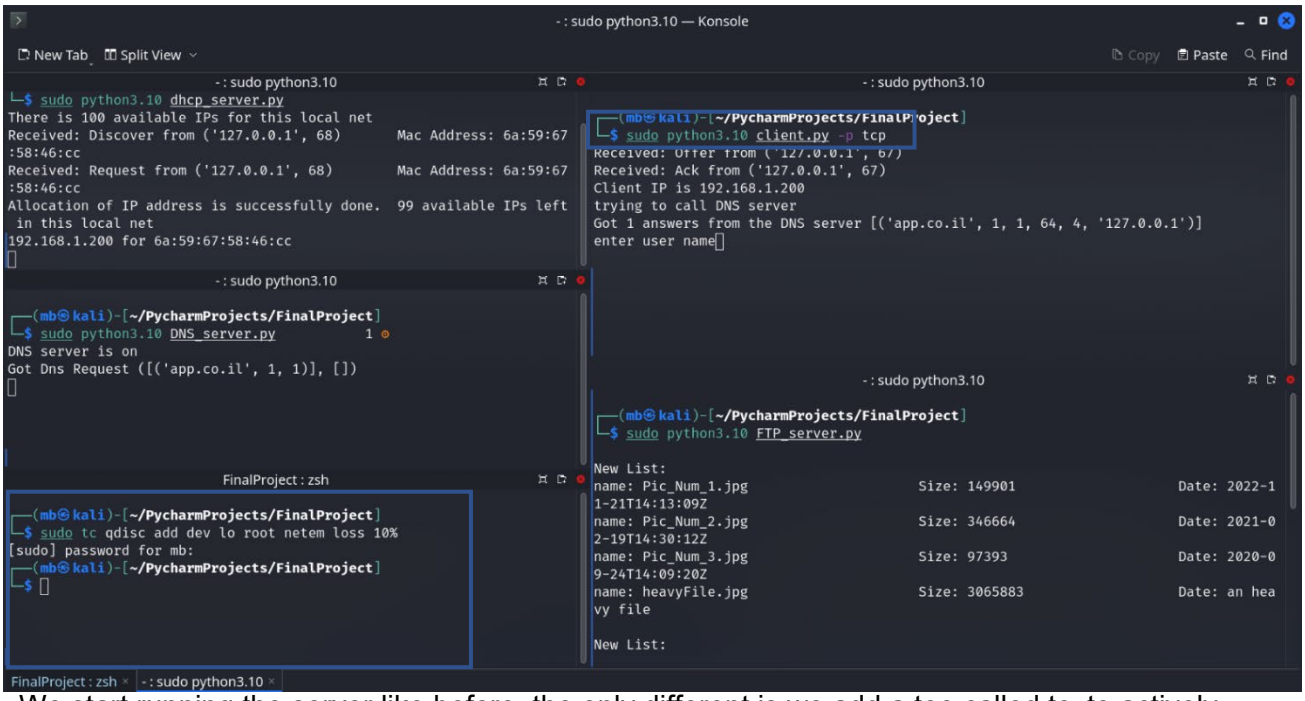


Stream eq 2: is the data over port 20, after the client send the RETR command he waits on port 20 to receive the data of the requested file.

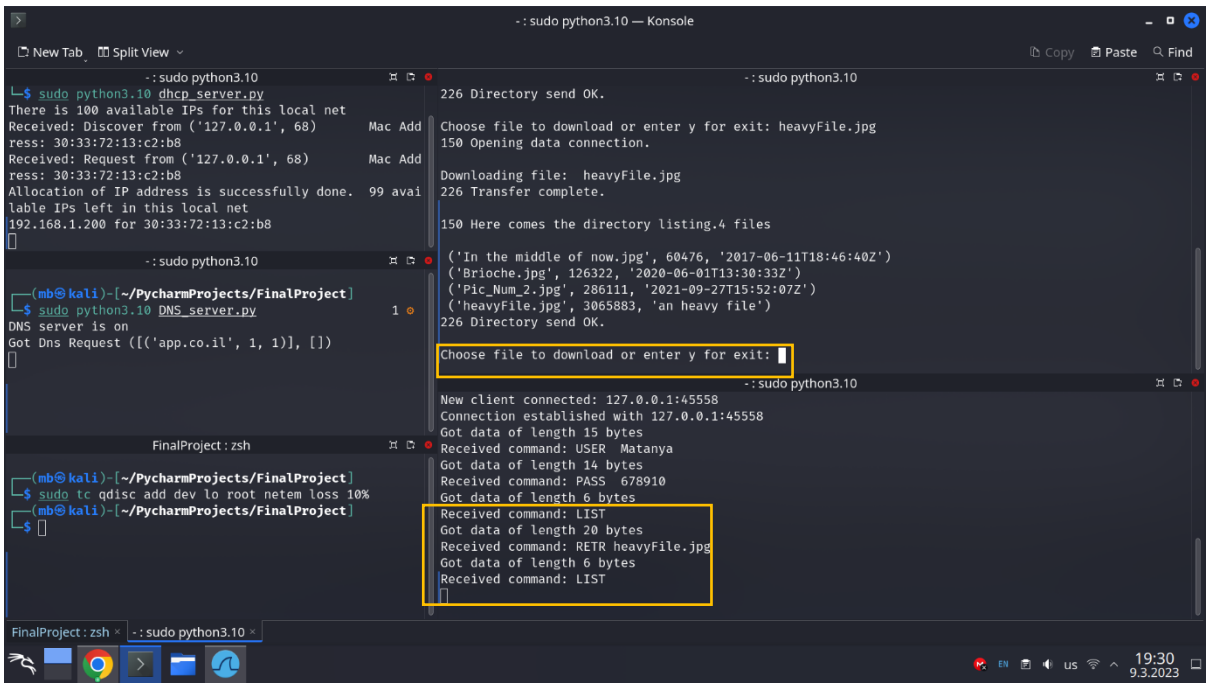
On the left wo see the actual data which sent from server, and the red sign for the ack`s data which sent by client.

Handle Packet loss

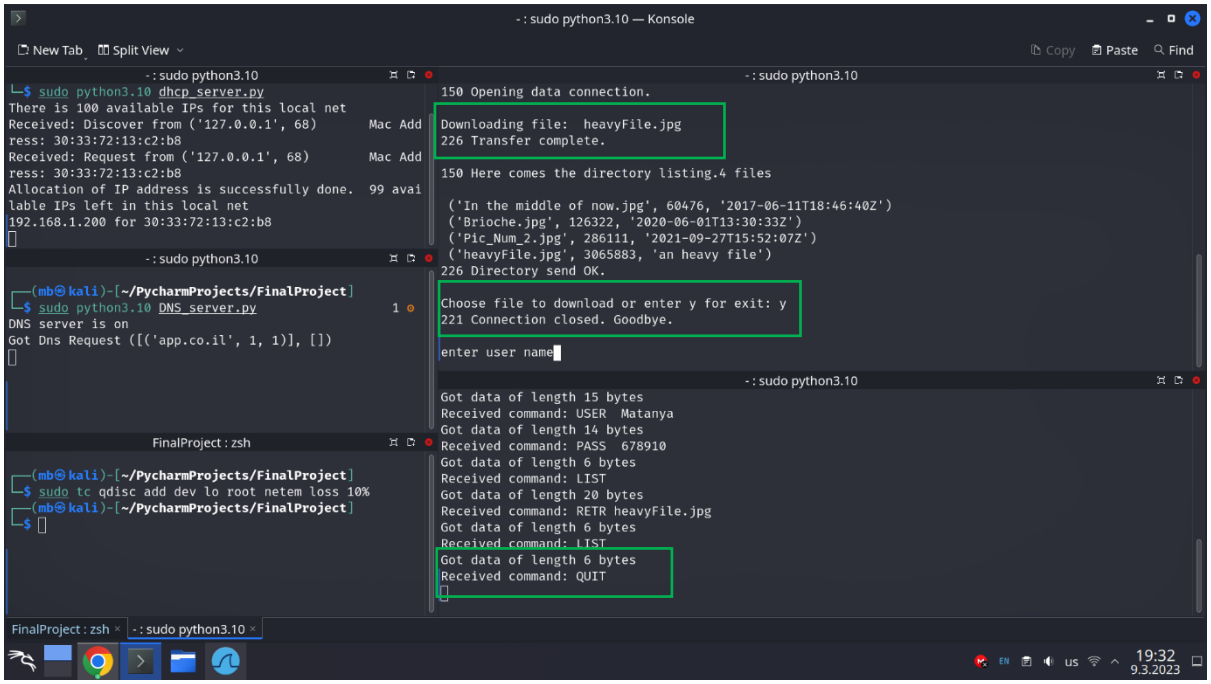
sending via TCP



We start running the server like before, the only different is we add a too called tc, to actively loss packets, and sees how the tcp protocol handles.

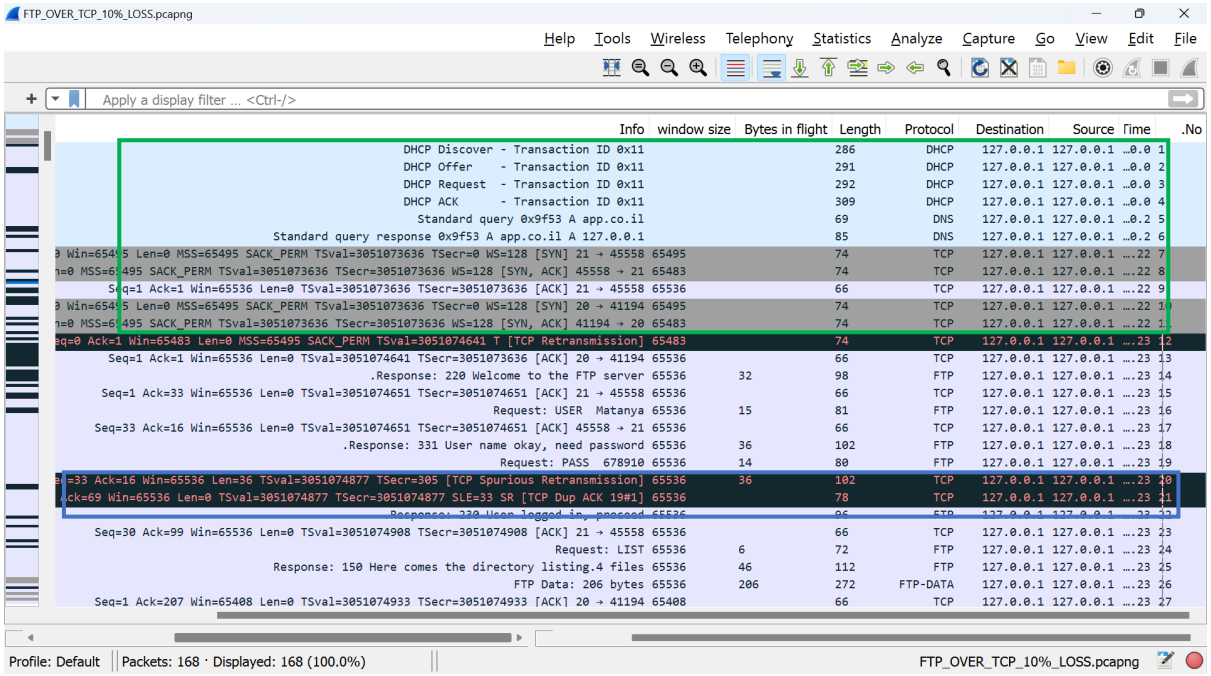


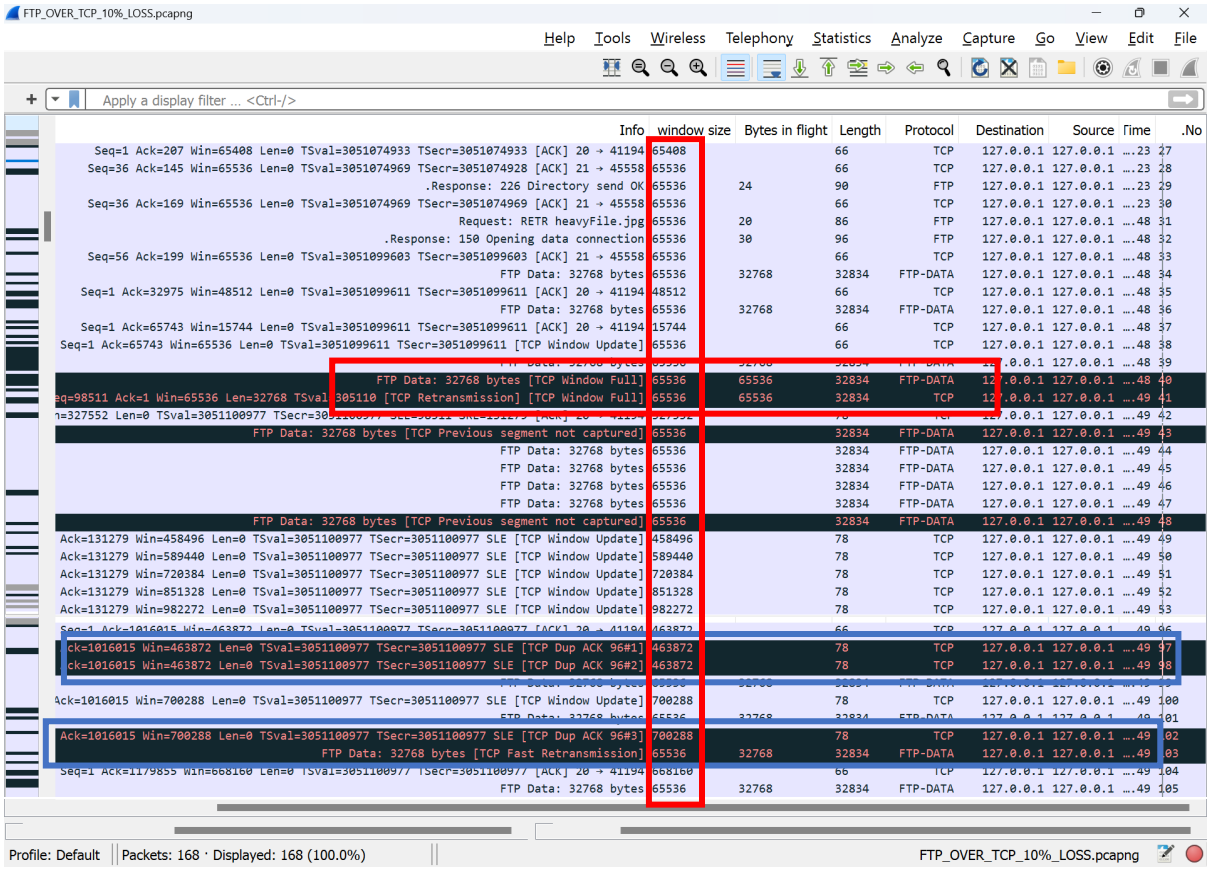
The servers run as we want, we enter the credentials, the server accept and ask for image to download or exit.



Although the packet loss, we managed to get the file from server, we aske the server for quit, and the connection close corrrectly.

Intercept

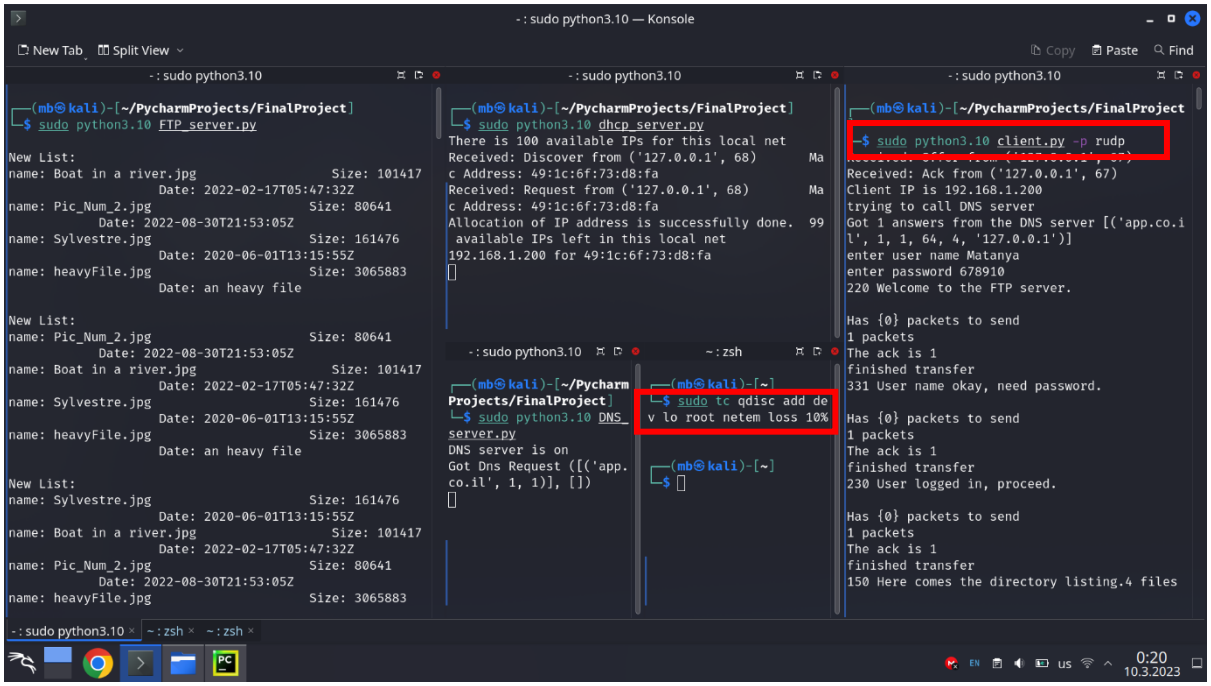




We can see the connection initially as expected, and tcp protocol managed the packets loss for us.

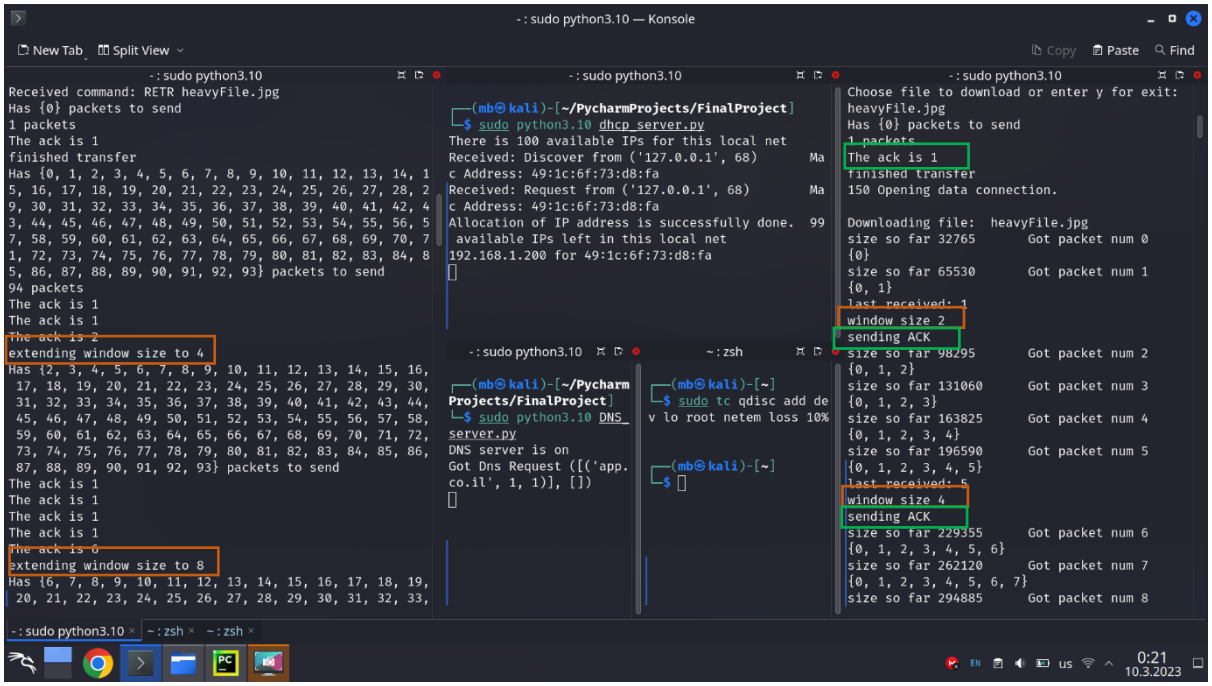
Using windows size changes and retransmitted the packet loss according to ack`s the server received

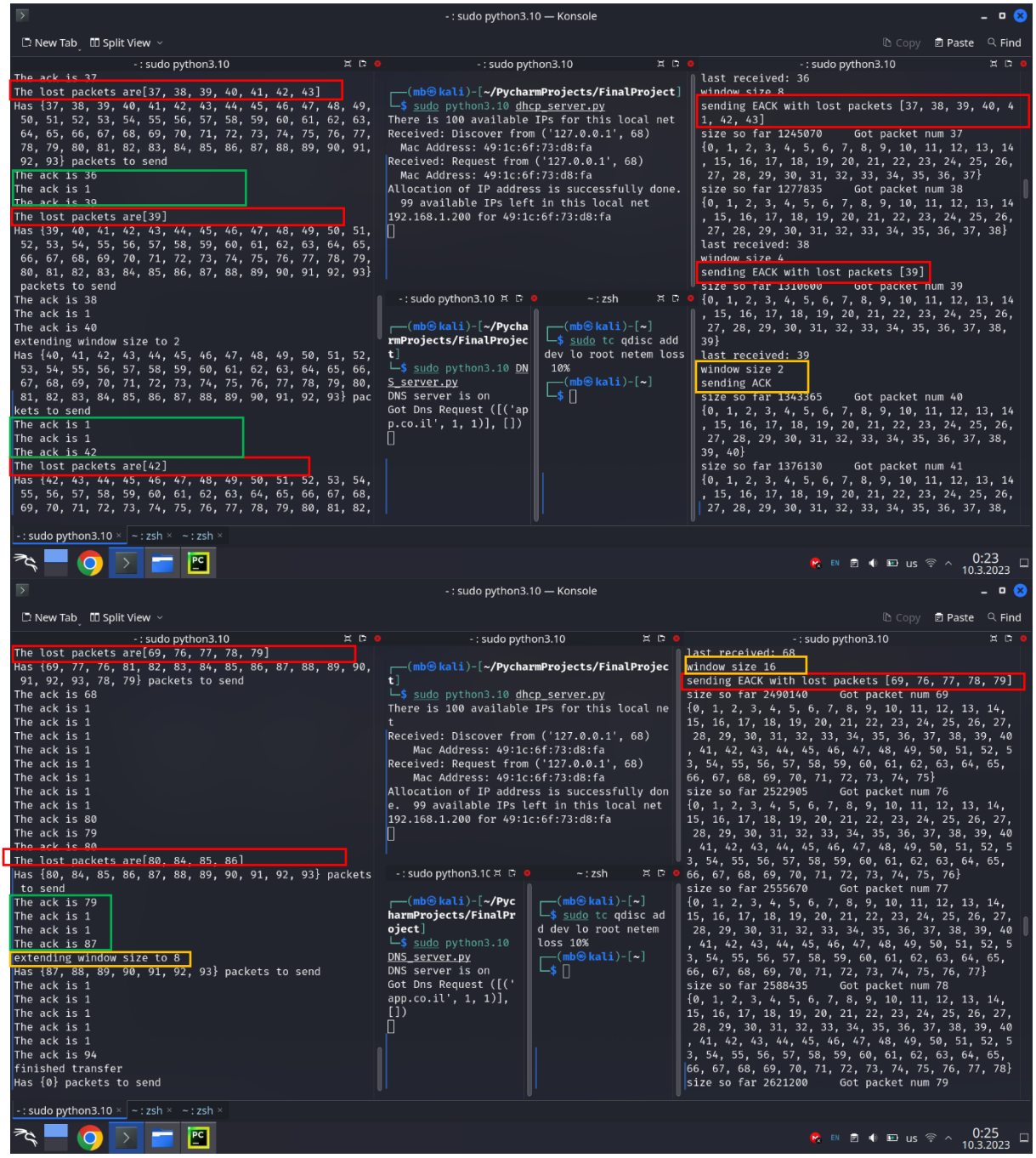
sending via RUDP

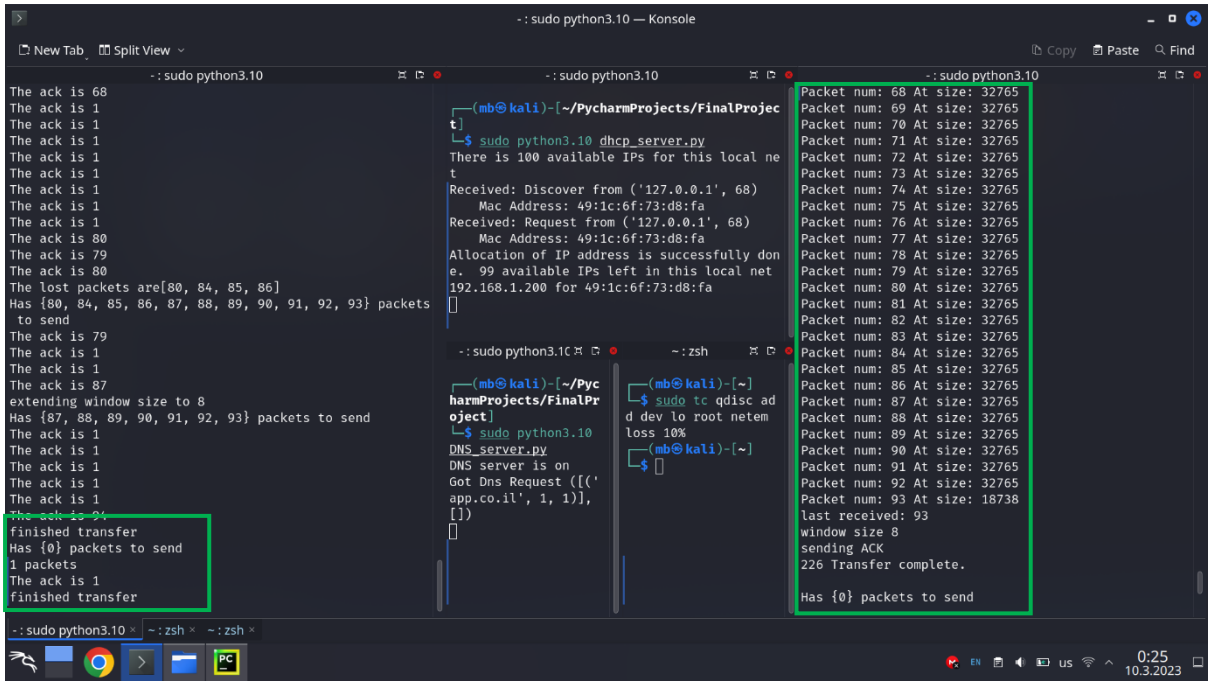


We ran our client over rudp protocol and tc tool for loosing packets.

In the following images we will see the server and client are communicate each other and updating the missing packets through EACK packet, also we will see the windows size updating and the ACK packet the client and server sends for reliability.

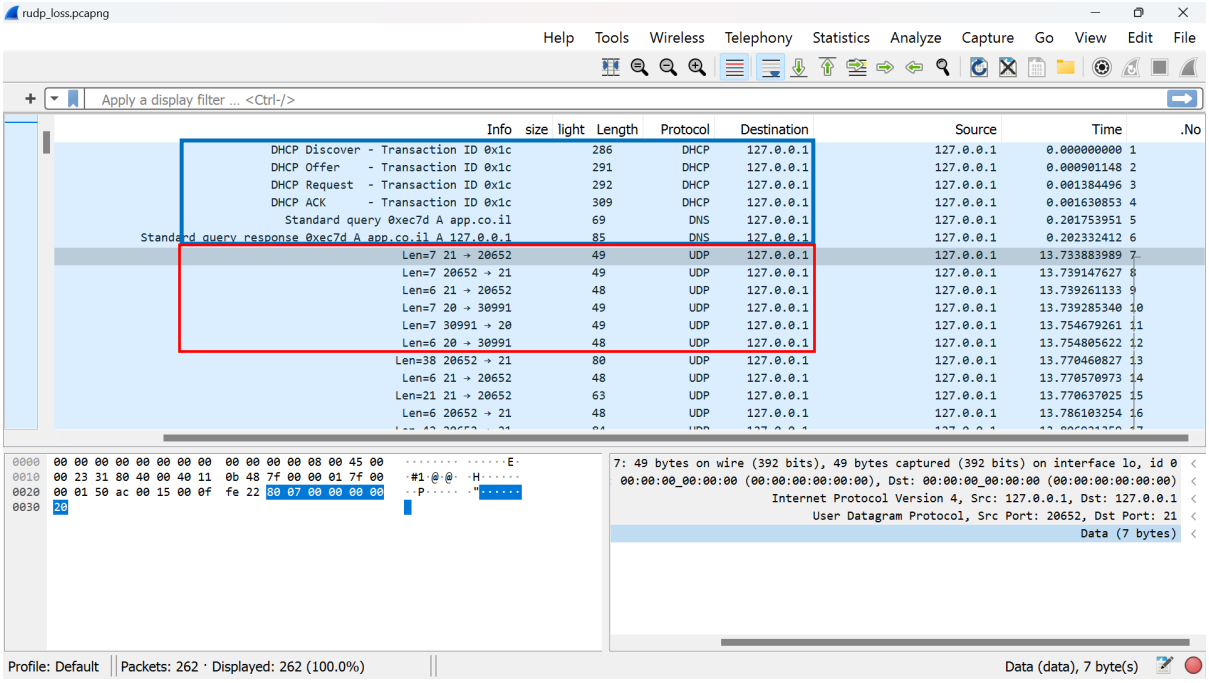



-: sudo python3.10 — Konsole
New Tab
Split View
Copy
Paste
Find
-: sudo python3.10
The lost packets are[37, 38, 39, 40, 41, 42, 43]
Has {37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93} packets to send
The ack is 36
The ack is 1
The lost packets are[39]
Has {39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93} packets to send
The ack is 38
The ack is 1
The ack is 40
extending window size to 2
Has {40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93} packets to send
The ack is 1
The ack is 1
The ack is 42
The lost packets are[42]
Has {42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
-: sudo python3.10
(mb㉿kali)-[~/PycharmProjects/FinalProject]
$ sudo python3.10 dhcp_server.py
There is 100 available IPs for this local net
Received: Discover from ('127.0.0.1', 68)
Mac Address: 49:1c:6f:73:d8:fa
Received: Request from ('127.0.0.1', 68)
Mac Address: 49:1c:6f:73:d8:fa
Allocation of IP address is successfully done.
99 available IPs left in this local net
192.168.1.200 for 49:1c:6f:73:d8:fa
-: sudo python3.10
(mb㉿kali)-[~/PycharmProjects/FinalProject]
$ sudo python3.10 DNS_server.py
DNS server is on
Got Dns Request ([('app.co.il', 1, 1)], [])
~: zsh
(mb㉿kali)-[~]
$ sudo tc qdisc add dev lo root netem loss 10%
(mb㉿kali)-[~]
$
-: sudo python3.10
last received: 36
window size 8
sending EACK with lost packets [37, 38, 39, 40, 41, 42, 43]
size so far 1245070 Got packet num 37
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37}
size so far 1277835 Got packet num 38
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38}
last received: 38
window size 4
sending EACK with lost packets [39]
size so far 1310600 Got packet num 39
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39}
last received: 39
window size 2
sending ACK
size so far 1343365 Got packet num 40
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40}
size so far 1376130 Got packet num 41
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
-: sudo python3.10
~: zsh
~: zsh
0:23
10.3.2023
-: sudo python3.10 — Konsole
New Tab
Split View
Copy
Paste
Find
-: sudo python3.10
The lost packets are[69, 76, 77, 78, 79]
Has {69, 77, 76, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 78, 79} packets to send
The ack is 68
The ack is 1
The ack is 1
The ack is 1
The ack is 1
The ack is 1
The ack is 1
The ack is 1
The ack is 1
The ack is 80
The ack is 79
The lost packets are[80, 84, 85, 86]
Has {80, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93} packets to send
The ack is 79
The ack is 1
The ack is 1
The ack is 87
extending window size to 8
Has {87, 88, 89, 90, 91, 92, 93} packets to send
The ack is 1
The ack is 1
The ack is 1
The ack is 1
The ack is 1
The ack is 94
finished transfer
Has {0} packets to send
-: sudo python3.10
(mb㉿kali)-[~/PycharmProjects/FinalProject]
$ sudo python3.10 dhcp_server.py
There is 100 available IPs for this local net
Received: Discover from ('127.0.0.1', 68)
Mac Address: 49:1c:6f:73:d8:fa
Received: Request from ('127.0.0.1', 68)
Mac Address: 49:1c:6f:73:d8:fa
Allocation of IP address is successfully done. 99 available IPs left in this local net
192.168.1.200 for 49:1c:6f:73:d8:fa
-: sudo python3.10
(mb㉿kali)-[~/PycharmProjects/FinalProject]
$ sudo python3.10 DNS_server.py
DNS server is on
Got Dns Request ([('app.co.il', 1, 1)], [])
~: zsh
(mb㉿kali)-[~]
$ sudo tc qdisc add dev lo root netem loss 10%
(mb㉿kali)-[~]
$
-: sudo python3.10
last received: 68
window size 16
sending EACK with lost packets [69, 76, 77, 78, 79]
size so far 2490140 Got packet num 69
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75}
size so far 2522905 Got packet num 76
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76}
size so far 2555670 Got packet num 77
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77}
size so far 2588435 Got packet num 78
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78}
size so far 2621200 Got packet num 79
-: sudo python3.10
~: zsh
~: zsh
0:25
10.3.2023

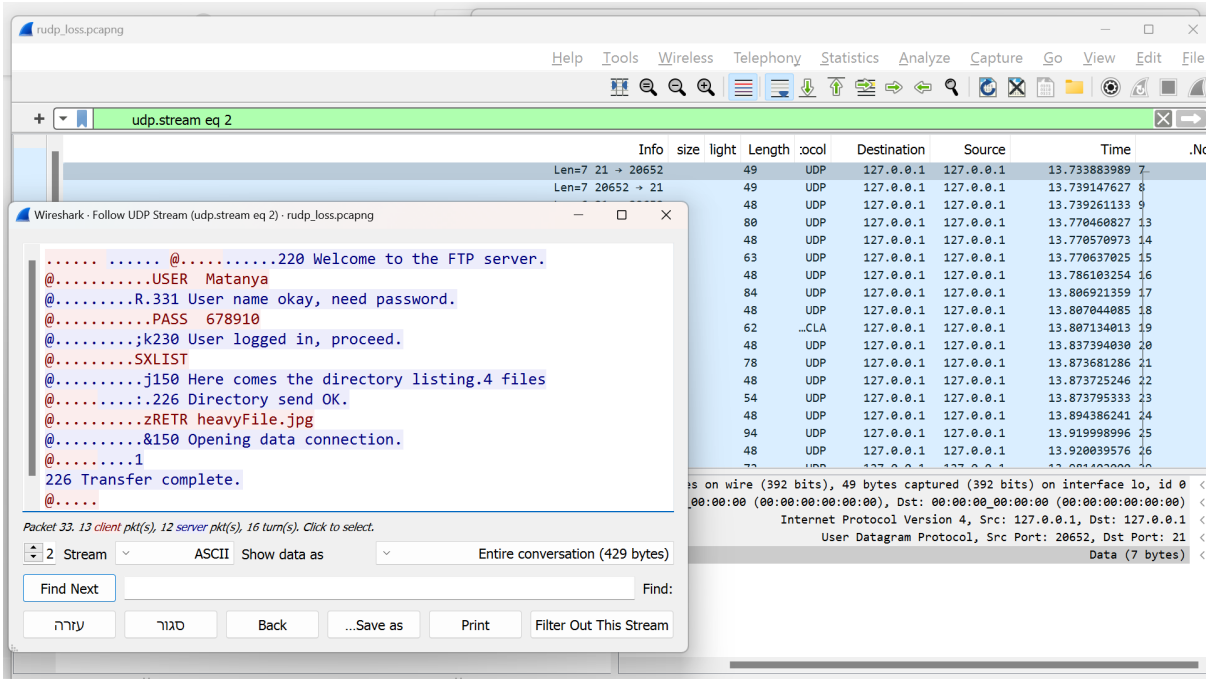


As we can see the image downloaded correctly although the loss packets.

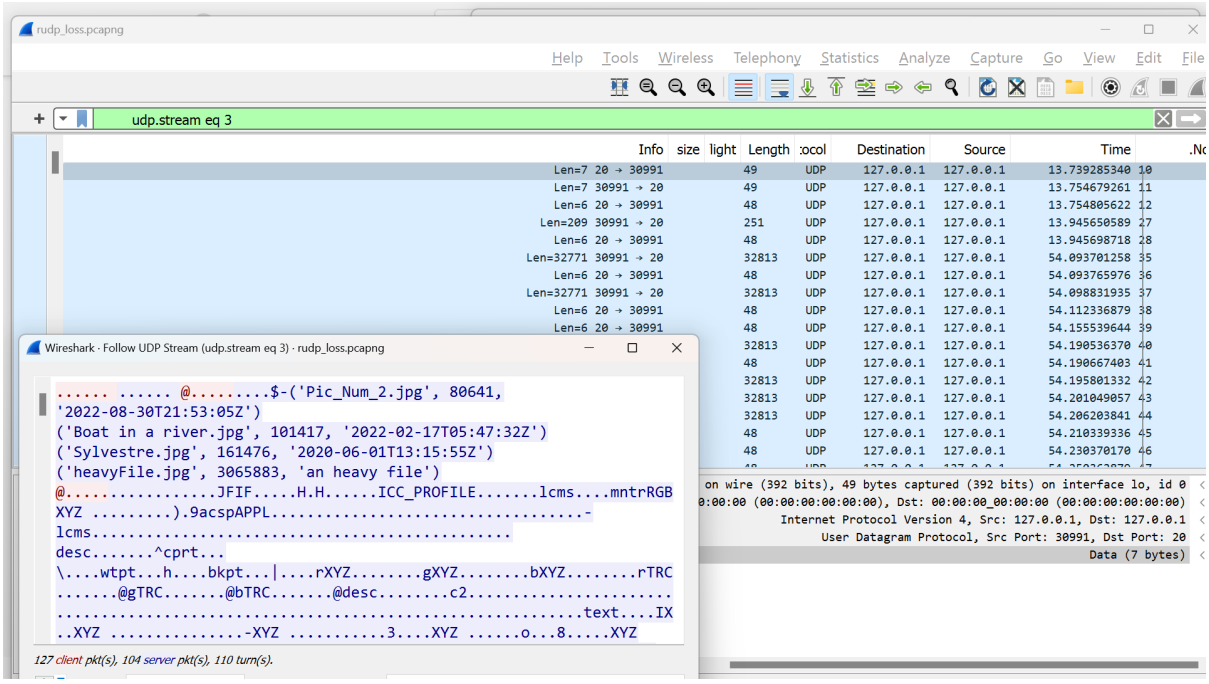
intercept



we can see the connection initially as expected with DHCP and DNS server. Also, the starting socket through port 21 and then port 20, as ftp server configure to.



Stream 2: we see the message the client send over port 21, such as credentials, USER, PASS,LIST,RETR commands, and acknowledge packets(at the starting line, before the ascii data)



Stream 3: for the server sending data, the first data is reply for LIST command, and after the acknowledgement from the client, the server start to send the file.

We only see the data the server sent, because the receiving from client is through port 21, on stream 2 (at the picture above).

So, even the packets lost, the server and client exceeded to manage the data they were expecting to get.

Additional Questions

1) List four main differences between TCP and QUIC protocol

1. **Connection Establishment:** TCP requires a three-way handshake to establish a connection, while QUIC uses a zero-round-trip-time connection setup, which allows a client to send data in the first message it sends to a server, without waiting for a response.

2. **Multiplexing:** QUIC permits the transmission of many data streams over a single connection, while TCP only uses one connection for each data transfer.

3. **Security:** QUIC provides Transport Layer Security (TLS) encryption and a secure handshake protocol to establish a connection securely. TCP does not offer built-in authentication or encryption.

4. **Packet Loss Recovery:** Congestion algorithms are used by TCP to recover from packet loss, which can result in substantial delays. While TCP takes longer to recover from packet loss than QUIC, the latter has its own congestion control technique that performs well with unreliable network connections.

2) List two main differences between Cubic and Vegas algorithms

- Cubic measures congestion based on packet loss, while Vegas measures it based on packet delay.
- Cubic uses a control mechanism that adjusts the sending rate based on a cubic function of the window size, resulting in a fast increase and slow decrease of sending rate. Vegas uses a smoother and more adaptive control mechanism based on the gradient of the delay function and has an adaptive timeout mechanism that reacts quickly to changes in network conditions.

3) what the BGP protocol is, how it differs from OSPF, and does it work according to short paths?

BGP or Border Gateway Protocol, is a protocol used to exchange routing information between **different networks** on the internet. It is different from **OSPF**, which is used within a **single network**. BGP allows network administrators to control how traffic is routed, which can be used for traffic engineering. It does not always choose the shortest path.

4) Fill the table and explain how the messages will change if there is a NAT between the user to the servers and whether you will use the QUIC protocol?

Table:

Application	Port Src	Port Des	IP Src	IP Des	Mac Src	Mac Des
FTP Server	30991 20652	20000 21000	127.0.0.1	127.0.0.1	00:00:00:00:00:00	00:00:00:00:00:00

The reason of the IP and MAC value is because we use 'loopback' interface, which is a virtual interface for internal communication, the IP Src and IP Dst is the same, to indicate they both running on 'lo' interface, the mac address use to identify physical component over the network. because it is a virtual interface we don’t need a physical identity, and so we don’t need to use physical address (MAC),

In case there is a NAT between the user and the servers:

When the packet sends from the client:

Mac src: is the client`s mac address.

Mac dst: is the touter mac address, or broadcast.

Ip src : is the internal client`s ip, which we get from DNS

Ip dst : is the server`s external ip, which we got from DHCP.

When the packet reaches to NAT:

Ip dst: remain the same.

Ip src: the NAT device replaces the private IP address in the packet with its public IP address.

Mac src : router mac address

Mac dst: client`s mac address

Will we use QUIC for our server? Short answer is yes, due to security reason.

QUIC (Quick UDP Internet Connections) is a transport layer network protocol developed by Google. It is designed to provide a secure and efficient transport protocol for web traffic over the internet.

we have an FTP server that supports both TCP and RUDP protocols, so we will want to use QUIC to improve the efficiency and security of our connections.

However, the Disadvantages of using QUIC, if we are working in an environment where there is limited support for QUIC, such as an older network infrastructure, we may not be able to take advantage of its benefits. Additionally, if we are working with applications or protocols that have been specifically optimized for TCP, such as some legacy applications or protocols, you may not see significant improvements by switching to QUIC.

5) explain the different between DNS to ARP protocols?

DNS (Domain Name System) and ARP (Address Resolution Protocol) are both network protocols, but they serve different functions.

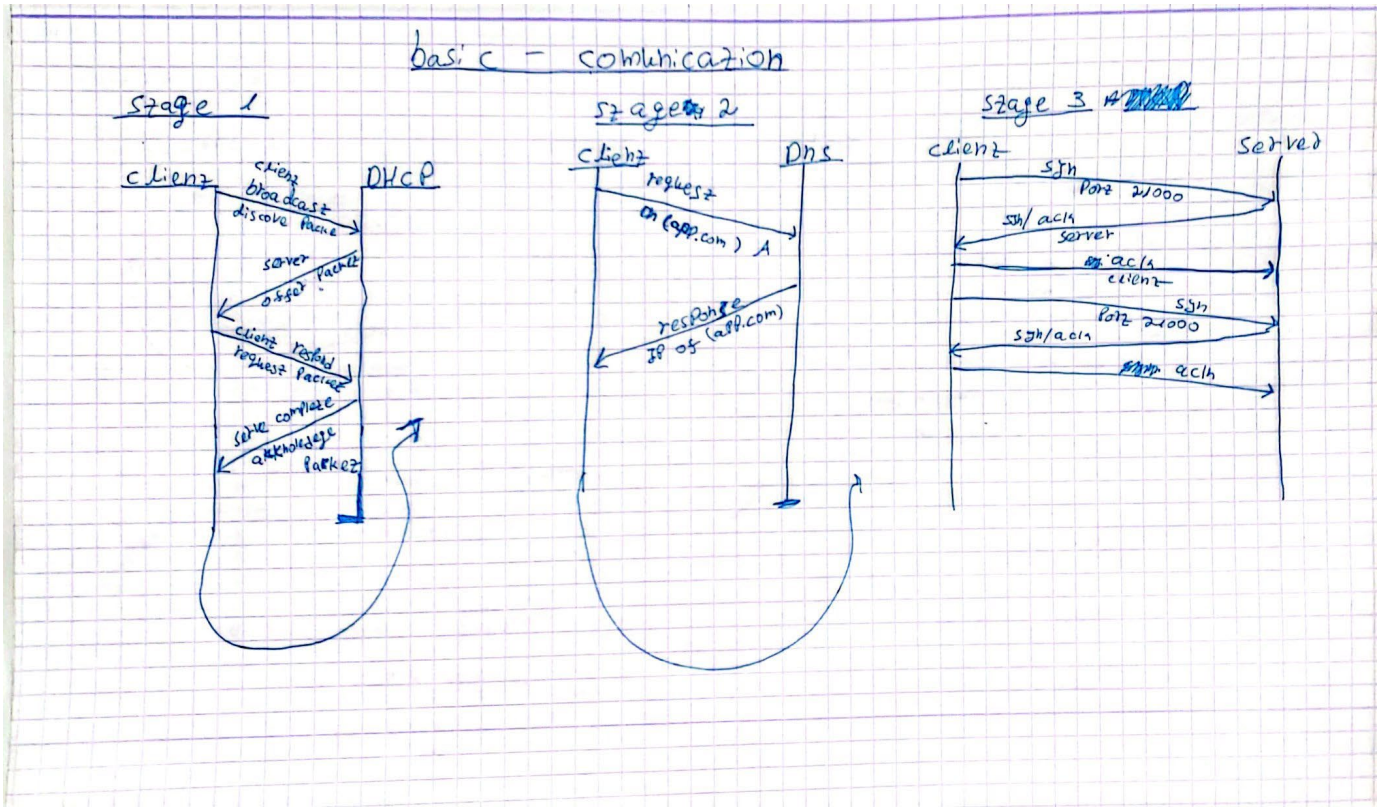
DNS and ARP operate at different layers of the network. DNS operates at the application layer, while ARP operates at the data link layer.

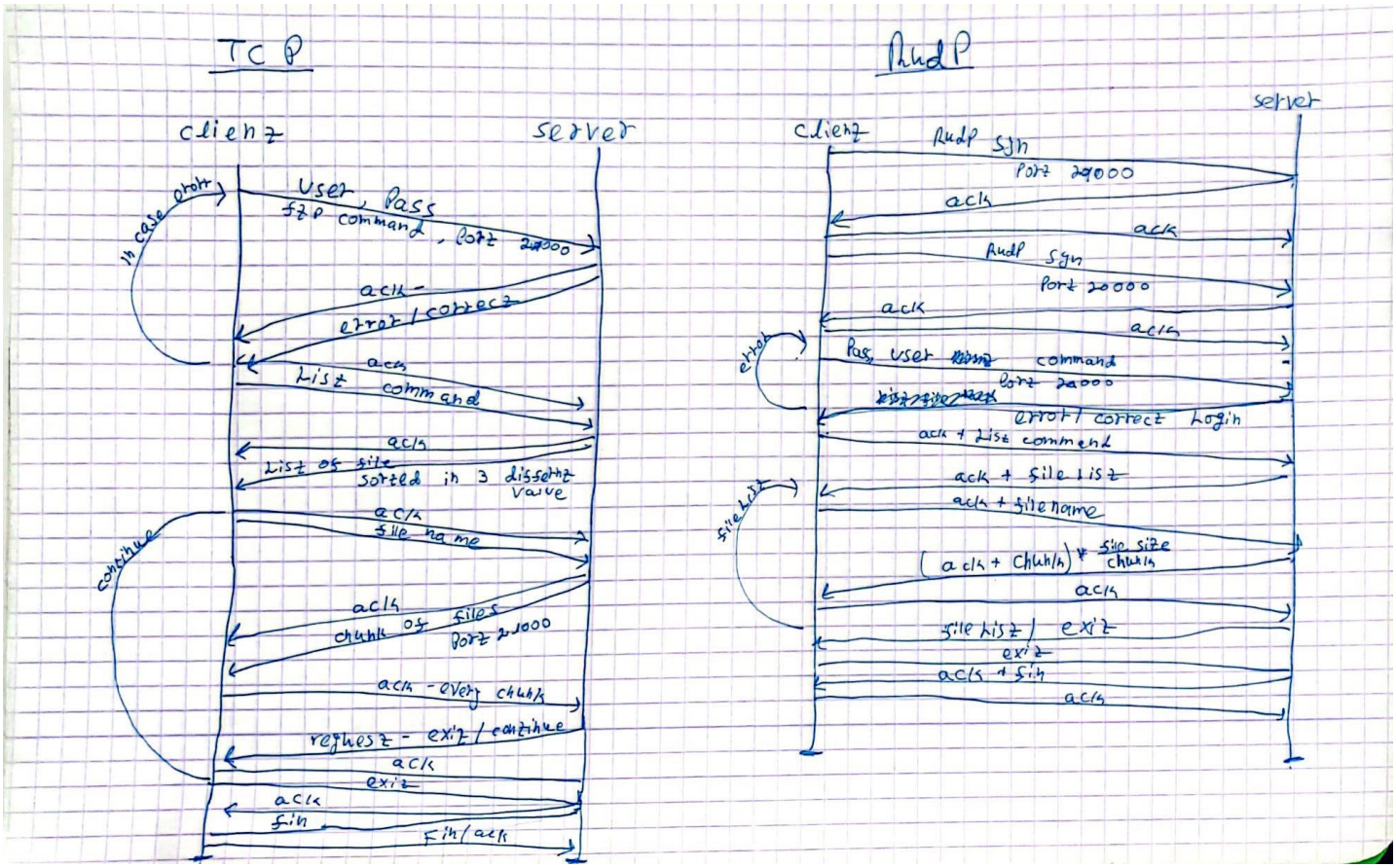
DNS is used to translate domain names into IP addresses. When a user types a domain name into a web browser, the browser sends a request to a DNS server to get the IP address associated with that domain name. Once the browser has the IP address, it can initiate a connection to the server associated with that address.

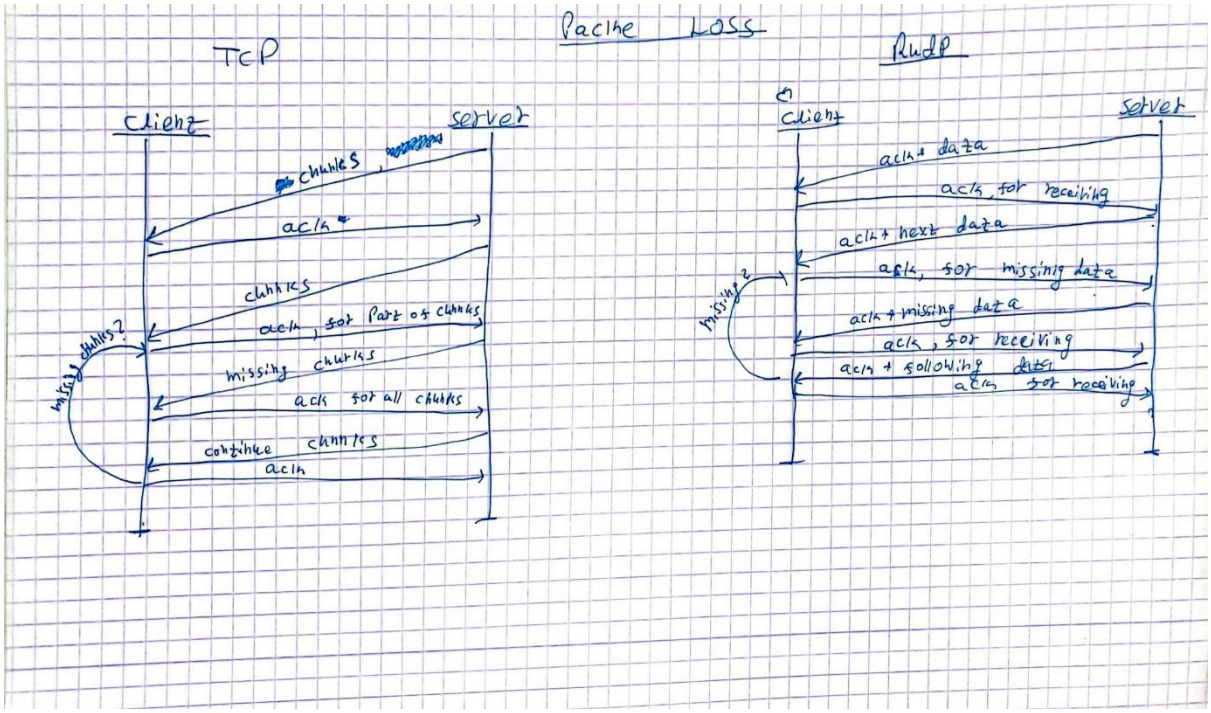
ARP, on the other hand, is used to translate IP addresses into MAC addresses. When a device on a network wants to communicate with another device on the same network, it needs to know the MAC address of that device. The device sends an ARP request to the network asking for the MAC address associated with a specific IP address. The device with that IP address responds with its MAC address, allowing the two devices to communicate with each other.

The main use of mac address is over LAN, when two components over the network can initial communication p2p, switch is the device which responsible to take a request from one client and send it physically to another device according to mac destination in the request.

Diagram State






Packet Loss
TCP
client
server
chunks
ack
chunks
ack, for Part of chunks
missing chunks?
missing chunks
ack for all chunks
combine chunks
ack
RUDP
client
server
ack + data
ack, for receiving
ack + next data
ask, for missing data
missing?
ack + missing data
ack, for receiving
ack + following data
ack, for receiving

Bibliography

- DHCP:
  - https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol
  - https://techhub.hpe.com/eginfolib/networking/docs/switches/5120si/cg/5998-8491_l3-ip-svcs_cg/content/436042653.htm
- DNS:
  - https://en.wikipedia.org/wiki/Domain_Name_System
  - https://mislove.org/teaching/cs4700/spring11/handouts/project1-primer.pdf
- FTP:
  - https://en.wikipedia.org/wiki/File_Transfer_Protocol
  - https://www.rfc-editor.org/rfc/rfc959
- RUDP:
  - https://en.wikipedia.org/wiki/Reliable_User_Datagram_Protocol
  - https://datatracker.ietf.org/doc/html/draft-ietf-sigtran-reliable-udp-00/
- QUIC:
  - https://en.wikipedia.org/wiki/QUIC
  - https://www.chromium.org/quic/
- CUBIC & VEGAS:
  - https://www.speedguide.net/articles/tcp-congestion-control-algorithms-comparison-7423