Final Project - SQL

Network Communications

Submitters:

Dor Shir, id: 308079797

Bar Alayof, id: 206840621

Abstract-

This paper is aimed to discuss over three main subject: **DHCP Server**, **DNS Server** and the main topic **SQL Server**.

Firstly, the DHCP Server is a network service that dynamically assigns IP addresses and other network configuration parameters to devices on a network. The server configuration includes the DHCP server IP address, subnet mask, lease time, and IP address pool. The DHCP server listens for DHCP Discover packets from clients and responds with DHCP Offer packets containing IP addresses from the available pool.

Secondly, the DNS Server is a network service that resolves domain names to IP addresses. The server maintains a cache of resolved domain names to reduce network traffic and speed up resolution times. The DNS server listens for DNS query packets from clients and responds with DNS response packets containing the resolved IP address.

Lastly, the SQL Server is a database management system that stores, retrieves, and manages data. The SQL server listens for SQL query packets from clients and responds with SQL query response packets containing the requested data. The server can also execute multiple SQL queries simultaneously and handle errors gracefully.

TCP and RUDP are two communication protocols used in computer networks. TCP (Transmission Control Protocol) is a reliable connection-oriented protocol that ensures the delivery of packets by providing flow control, error checking, and retransmission of lost packets. RUDP (Reliable User Datagram Protocol), on the other hand, is a connectionless protocol that guarantees delivery of packets with minimal overhead. While TCP is commonly used for applications that require reliable delivery of data, such as email and file transfers, RUDP is more suitable for real-time applications, such as video streaming and online gaming, where low latency is more important than reliability.

Keywords -

1. Introducing-

1.1 DHCP Server

DHCP Server: A DHCP server is a network server that automatically assigns IP addresses to devices on the network. The DHCP server manages a pool of IP addresses and assigns them to devices as they connect to the network. This eliminates the need for manual IP address configuration and greatly reduces the possibility of IP address conflicts. The DHCP server also provides additional configuration information to devices on the network, such as subnet masks, default gateways, and DNS servers. This additional information helps devices communicate with each other more efficiently.

DHCP Client: A DHCP client is a device that connects to a network and requests an IP address from the DHCP server. The client sends a broadcast message requesting an IP address, and the DHCP server responds with an available IP address from its pool. The client then uses the assigned IP address to communicate with other devices on the network. The use of DHCP simplifies network administration and allows devices to easily connect to a network without requiring manual IP address configuration.

Configuration stages between Server and Client:

1.Discovery: **DHCP client** broadcasts a message requesting network configuration information.

2. Offer: **DHCP server** responds with a message offering an IP address and other network configuration information

3. Request: **DHCP client** broadcasts a message requesting to use the offered IP address

4. Acknowledgement: **DHCP server** responds with a message acknowledging the client's use of the offered IP address

1.2 DNS Server

DNS Server is responsible for translating human-readable domain names (such as google.com) into IP addresses that can be understood by computers. it allows users to easily access websites and other online resources without needing to remember the underlying IP addresses.

In our code we first check if the request domain name is storage in its cache , if he didn't than the DNS Server builds a DNS query packet and sends it to a remote DNS server and he will circle the request through DNS hierarchy(Root ->TLD - > Authoritative DNS Server) until we receive the response with the corresponding ip and sent it to the client.

1.3 Application – SQL Server

SQL Server is a database management system used to store, organize, and retrieve data. It is designed to be a scalable, high-performance platform for managing data.

In general, SQL Server is a client-server application, with clients connecting to the server to query, update, or manipulate data.

In our code the SQL Server hold a data table with _____ values we create , which only the Server have a access to.

The client that wants to retrieve information from the table is able to send 10 queries to the Server and get back from the server response according the information storage inside the data table.

We implement to ways for communication between client and server:

1.TCP – protocol that fundamentally based on reliable delivery of packets.

2.RUDP – protocol that his base – UDP , isn't reliable but we make changes in order to make him reliable.

Q & A :

Question 1:

Named at least four major differences between the TCP and QUIC protocols.

Answer 1:

1. Connection Establishment:

TCP : Three-way handshake process to establish a connection between the client and the server.

Quick: Single round-trip time (RTT) handshake.

2. Multiplexing

TCP: Single channel for multiple streams of data.

Quick: Multiple channels for different data streams.

3. Packet Loss Recovery:

TCP: Complex retransmission mechanism for packet loss.

Quick: Simpler recovery mechanism that is based on error correction codes and retransmissions.

4. Congestion Control:

TCP: Slow-start congestion control mechanism.

QUICK: Aggressive congestion control algorithm in compared to Slow-start that can quickly adapt to changes in network conditions.

Question 2:

Named at least two major different between Cubic to Vegas protocols.

Answer 2:

1. Congestion window management:

Cubic: Use concave growth function to manage the congestion window.

Vegas: Linear growth function.

2. RTT measurement:

Cubic: Measures RTT based on the time it takes to transmit a packet and receive an acknowledgment.

Vegas: Measures RTT on a more accurate method compare to Cubic, based on the time between the transmission of two packets.

Question 3:

Explain what BGP protocol is , how it is differ from OSPF protocol and those he work according to short routes.

Answer 3:

The Border Gateway Protocol (BGP) is an inter-domain routing protocol used to exchange routing information between different Autonomous Systems on the Internet. It is used to route traffic between different Internet Service Providers (ISPs) and large organizations that have their own network infrastructure.

Unlike OSPF (Open Shortest Path First), which is an interior gateway protocol used for routing within a single autonomous system, BGP is an exterior gateway protocol used for routing between autonomous systems.

BGP does not work according to short routes. Instead, it uses a path-vector algorithm that takes into account several factors, such as the number of AS hops and the quality of the AS path, to determine the best path for a given network prefix. This allows BGP to make intelligent routing decisions that can take into account factors such as network performance, reliability, and policy constraints.

*AS - Autonomous System

Question 4:

Given the code you developed in this project, please add the data to this table on a process basis

The messages of your project. Explain how the messages will change if there is a NAT between the user to the servers and will you use the QUIC protocol

Answer 4:

Application	Port Src	Port Des	IP Src	IP Des	Mac SRC	Mac Des
DHCP Server	67	68	255.255.255.255	0.0.0.0	Server's NIC	ff:ff:ff:ff:ff:ff
DHCP Client	68	67	0.0.0.0	255.255.255.255	Client's NIC	ff:ff:ff:ff:ff:ff
Client Local DNS	53	53	127.0.0.1	127.0.0.1	Client's MAC	Server's MAC

Local DNS Server	53	53	Your public Ip	8.8.8.8	Server's MAC	Client's MAC
SQL Server	30797	20621	127.0.0.1	127.0.0.1	Server's MAC	Client's MAC
SQL Client	20621	30797	127.0.0.1	127.0.0.1	Client's MAC	Server's MAC

*NIC - Network Interface Car

How does using QUIC over TCP/RUDP will change the messages between client and servers?

DHCP Server:

1.QUIC's use of multiplexed streams can allow for multiple DHCP requests and responses to be sent simultaneously over a single connection, potentially improving overall network performance.

2. With QUIC in place, the messages between the client and server may be more efficient due to QUIC's built-in congestion control and packet loss recovery mechanisms.

DNS Server:

1. QUIC can potentially improve the speed and reliability of DNS lookups by reducing the number of round trips needed to establish a connection and retrieve a response.

2. The use of QUIC may also improve the security of DNS traffic, as QUIC includes built-in encryption and authentication mechanisms.

SQL Server:

The use of QUIC may also reduce the overhead of establishing and maintaining SQL connections, as QUIC's connection establishment process is typically faster than that of TCP.

How does adding NAT will change the messages between client and servers?

DHCP Server:

1.With NAT in place, the DHCP server may see requests coming from a single IP address, rather than from individual client IPs. This can make it more difficult to track down issues with specific clients or diagnose network problems.

2.NAT can also impact the ability of the DHCP server to assign IP addresses to clients, particularly if the DHCP server is located on a different subnet than the clients. Additional configuration may be needed to ensure that DHCP requests and responses are properly routed.

DNS Server:

1.NAT can impact the ability of the DNS server to accurately identify the source of DNS requests. If multiple clients are sharing a single public IP address via NAT, the DNS server may see all requests as coming from the same IP.

2.In some cases, NAT can also introduce additional latency into the DNS resolution process, particularly if the NAT device is under heavy load or not properly configured.

SQL Server:

1.Similar to the DHCP server, NAT can make it more difficult to track down specific client connections to the SQL server. Without proper configuration, it may be difficult to differentiate between connections coming from different clients behind the same NAT device.

2.Depending on the specifics of the NAT implementation, there may be additional overhead introduced into the SQL connection process. This could impact performance or increase latency for client connections.

3.If the SQL server is located on a different subnet than the clients, additional configuration may be needed to ensure that connections are properly routed through the NAT device.

Question 5:

Explain the differences between the ARP protocol and DNS.

Answer 5:

1. Purpose: The ARP protocol is used to map a network address (such as an IP address) to a physical address (such as a MAC address) on a local network. In contrast, DNS is used to translate domain names (such as www.example.com) into IP addresses.

2. Network Scope: ARP operates at the data link layer of the networking stack and is used for mapping physical addresses within a local network. DNS operates at the application layer of the networking stack and is used for translating domain names to IP addresses on the internet.

3. Request and Response: ARP requests are broadcasted to all devices on the local network, while DNS requests are typically sent to a DNS server which then responds with the requested information.

4. Frequency of use: ARP is used frequently by devices on a local network for efficient communication between devices, while DNS requests are generally infrequent and are made by end-user devices only when required for accessing specific resources on the internet.

5. Address Type: ARP maps network addresses (such as IP addresses) to physical addresses (such as MAC addresses) on the local network. DNS maps domain names to IP addresses.

- To answer how the system overcomes the loss of packages?

One approach to overcome the loss of packages in UDP is to implement a reliability mechanism on top of UDP. This can be done by implementing techniques such as retransmission of lost packets, flow control, congestion control, and error detection and correction.

Retransmission involves re-sending lost packets until they are successfully received by the receiver. Flow control limits the amount of data sent at a time to prevent the receiver from being overwhelmed. Congestion control controls the rate at which data is sent to prevent congestion on the network. Error detection and correction adds additional data to the packet to detect and correct errors that may occur during transmission.

- To answer how the system overcomes the latency problems?

Reliable Data Transfer: This technique involves implementing mechanisms such as sequence numbers, acknowledgements, and timers to ensure that packets are delivered reliably to the destination. One common protocol that uses this technique is the Stop-and-Wait Protocol.

Selective Repeat is a technique used in data communication to ensure reliable delivery of packets over an unreliable channel, such as UDP.

In Selective Repeat, the sender sends a stream of packets to the receiver and waits for an acknowledgement for each packet. If the sender does not receive an acknowledgement for a packet within a certain time period, it retransmits the packet. However, instead of waiting for all packets to be acknowledged before retransmitting any, the sender keeps track of the packets that have been acknowledged and only retransmits the packets that have not been acknowledged.

The receiver acknowledges each packet it receives by sending an acknowledgement message back to the sender. If a packet is lost or damaged in transit, the receiver sends a negative acknowledgement (NACK) to the sender, which requests the sender to retransmit the lost packet.

In this way, Selective Repeat can reduce the number of retransmissions required, and therefore reduce the overall delay in delivering the packets. By selectively retransmitting only the lost packets, the sender can minimize the number of redundant transmissions and maintain a high throughput even in the presence of a large number of lost packets.

To implement Selective Repeat in UDP, the sender and receiver need to maintain a window of packets. The window size is determined by the maximum number of packets that can be in transit at any given time. The sender sends packets within the window, and the receiver acknowledges the packets as they arrive. The sender can send new packets as old ones are acknowledged, up to the size of the window. If a packet is lost or damaged, the receiver sends a NACK to request the sender to retransmit it. The sender retransmits only the lost packets, while continuing to send new packets within the window.

2. Methods-

System: Windows

Machine code: Python

Files: App_server.py, Client.py, DHCP.py, DNS.py, mydatabase.db

To run the program:

(Make sure that mydatabase.db is at the same directory with the .py files)

Open any desirable server file via one terminal – for example App_server while 'files_directory' is the path in your computer where the python file are.

1) 'cd files_directory'
2) python App_server.py

Open in a new terminal the client file

1) 'cd files_directory'
2) python Client.py

Program:

1. **Client – DHCP**

The get_ip() method sends a DHCP discover packet over the network using the sendp() function.

The sniff() function is called to capture DHCP packets on the network that match the specified filter expression.

The sniff() function waits for a DHCP offer packet to be received within 5 seconds. If a DHCP offer packet is received, it calls the detect_dhcp() function to process the packet.

The detect_dhcp() function checks if the packet contains a DHCP offer message. If it does, it sends a DHCP request message to the DHCP server to request the offered IP address.

If the DHCP server responds with a DHCP ACK message, the detect_dhcp() function extracts the assigned IP address, gateway IP address, subnet mask, and lease time from the packet and update the client.

In summary, the get_ip() method uses DHCP to dynamically obtain an IP address from a DHCP server on the network. It sends a DHCP discover packet to initiate the process and waits for a DHCP offer packet from the DHCP server. Once the offer is received, it sends a DHCP request packet to request the offered IP address and waits for a DHCP ACK packet from the server. If the request is successful, the method returns the assigned IP address.

2. **Client – DNS**

send_dns_query() is called with a domain name as an argument.

The method creates a TCP socket and connects to the DNS server on port 53.

The DNS server listens for incoming TCP connections from clients on the specified IP address and port, and accepts the TCP connection from the client.

The method builds a DNS query packet and sends it to the DNS server over the TCP connection using the sendall() method.

The server receives the DNS query packet from the client over the TCP connection using the recv() method.

The server parses the DNS query packet and extracts the requested domain name. Then checks if the requested domain name is in **its cache and has not expired.**

If the domain name **is in** the cache and has not expired, the server constructs a DNS response packet using and returns the cached IP address to the client.

If the domain name **is not in** the cache or has expired, the server forwards the DNS query packet to the Google DNS server using a UDP socket and the sr1() method.

The server receives the DNS response packet from the Google DNS server over the UDP socket and extracts the resolved IP address.

The server stores the resolved IP address in its cache along with the current time.

The server constructs a DNS response packet using scapy and returns the resolved IP address to the client over the TCP connection. The server then closes the TCP connection to the client.

Meanwhile the client waits for a response from the DNS server using the recv() method of the socket object. This method blocks until data is received from the socket or the timeout occurs.

The response packet is stored in a byte string, which is then converted to a scapy packet object. This packet contains the DNS response packet sent by the DNS server.

The client checks **if** the response packet contains any answer records using the haslayer() method of scapy. If it does not have any answer records, the method prints an error message indicating that the domain name could not be resolved and returns.

If the response packet contains an answer record, the client extracts the resolved IP address from the packet.

The resolved IP address is printed to the console and the TCP connection to the DNS server is closed.

In summary, The client sending a domain name to a DNS server. The server then checks its cache and returns a resolved IP address to the client if it has one. If it doesn't, it forwards the query to a Google DNS server, receives the IP address, stores it in its cache, and returns it to the client. If the server is unable to resolve the domain name, the client receives an error message.

3. **Client – App server (TCP)**

The send_queries_tcp() method is called with a list of SQL queries as input.

The client creates a TCP socket and connects to the server at the specified address and port(localhost and port 5002).

The client iterates over the list of queries and sends each query to the server as a packet using the packet format defined.

The server receives each packet containing a query, decodes the query and stores it in a list of queries.

Once all 10 queries have been received, the server executes each query and sends the results back to the client as packets using the same packet format defined.

The client waits for a response to each query, receives the packet containing the response, decodes the response, and appends it to a list of responses.

After all responses have been received, the client checks that the number of responses received matches the number of queries sent. If the number of responses is different, an exception is raised.

The client and the server closes the TCP connection.

Overall, the client sends SQL queries to the server and receives responses back. The server handles the queries and sends the responses back to the client. The client waits for the responses and collects them into a list and print them.

4. Client – App server (RUDP)

The client creates a socket and binds it to a local address and port number(localhost and port 5002).

The server creates a socket and binds to the same address and port number.

The client constructs a packet with a sequence number, the query string, and a checksum. The packet is sent to the server using the sendto() method.

The client waits for a response from the server using the recvfrom() method. **If** the response is not received within a specified timeout period, the client resends the packet.

The server receives the packet and validates the checksum. **If** the checksum is invalid, the server sends a negative acknowledgement (**NACK**) to the client indicating that the packet was not received correctly.

If the checksum is valid, the server checks if the packet is within the **sliding window** of the RUDP protocol. **If** the packet is within the sliding window, the server sends a positive acknowledgement (**ACK**) to the client indicating that the packet was received correctly. The server also adds the packet to a dictionary of received packets and slides the window if possible.

If the packet is outside the sliding window, the server sends an ACK to the client for the received packet, but **does not process it** further.

The client receives the ACK from the server and proceeds to the next query string in the list of queries to be executed.

Once all queries have been sent and ACKed by the server, the client waits for a response from the server using the recvfrom() method.

The server receives the packets from the client, executes the queries, and sends the results back to the client using the RUDP protocol.

The client receives the results packets from the server, validates the checksums, and sends ACKs to the server indicating that the packets were received correctly.

If the client does not receive a response from the server within a specified timeout period, it resends the last unacknowledged packet.

Once all packets have been ACKed by the client, the server and the client closes the RUDP connection.

In summary, when the client calls the send_queries_rudp() method, it establishes a RUDP connection with the server, sends 10 queries to the server, and receives the results of the queries. The server receives the queries, executes them, and sends the results back to the client using RUDP protocol to ensure reliable delivery. Finally, the RUDP connection is closed.

3. Wireshark-

3.1 DHCP

Client sends a DHCP Discovery broadcast.

The server occurred the packet and send an offer to the client.

The client receive the offer and send a request packet to the server.

The server receive the request and sends an ACK packet.

dhcp

No.	Time	Source	Flags	Destination	Protocol	Length	Info
347	27.553914	0.0.0.0		255.255.255.255	DHCP	286	DHCP Discover - Transaction ID 0xc9033df
348	27.559736	192.168.1.1		255.255.255.255	DHCP	310	DHCP Offer - Transaction ID 0xc9033df
349	27.563394	0.0.0.0		255.255.255.255	DHCP	298	DHCP Request - Transaction ID 0xc9033df
350	27.566558	192.168.1.1		255.255.255.255	DHCP	310	DHCP ACK - Transaction ID 0xc9033df

DHCP server

```
Hello there!

[*] Enter '1' to assign an IP address
[*] Enter '2' for a DNS request
[*] Enter '3' to send SQL queries
[*] Enter '4' to exit
1
Getting IP Address
.
Sent 1 packets.
Received DHCP Offer from: ff:ff:ff:ff:ff:ff
.
Sent 1 packets.
[*] src: ff:ff:ff:ff:ff:ff
[*] IP Address: 192.168.1.54
[*] Gateway IP Address: 192.168.1.1
[*] Subnet Mask: 255.255.255.0
[*] Lease time: 3600


[*] Enter '1' to assign an IP address
[*] Enter '2' for a DNS request
[*] Enter '3' to send SQL queries
[*] Enter '4' to exit
4
Exit
```

Client

```
DHCP server is sniffing for DHCP packets..
Received DHCP Discover from: 00:00:00:00:00:00
.
Sent 1 packets.
Sent DHCP OFFER.
Received DHCP Request from: 00:00:00:00:00:00
.
Sent 1 packets.
Sent DHCP ACK with configuration.
```

3.2 DNS

Interface: loopback

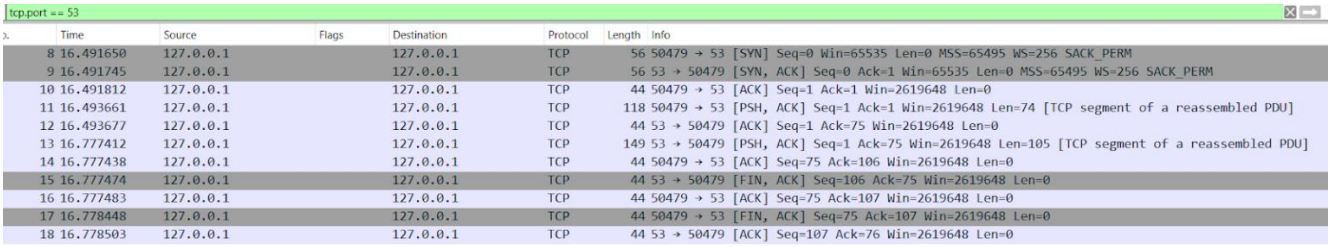
TCP connection, client with the local DNS server

TCP handshake client-local DNS server.

The client sends a DNS query to the server. The server then passing it forward to the DNS server of Google(will be shown in the Wi-Fi interface below) and receive the response.

The server send the DNS response to the client

The client receive the DNS response and they close the TCP connection



tcp.port == 53

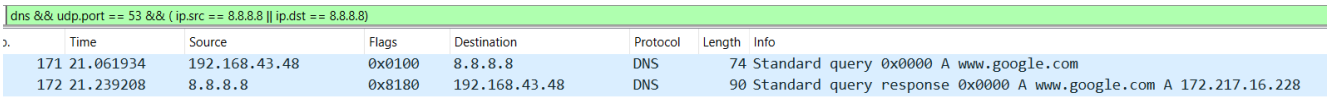
No.	Time	Source	Flags	Destination	Protocol	Length	Info
8	16.491650	127.0.0.1		127.0.0.1	TCP	56	50479 → 53 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
9	16.491745	127.0.0.1		127.0.0.1	TCP	56	53 → 50479 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
10	16.491812	127.0.0.1		127.0.0.1	TCP	44	50479 → 53 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
11	16.493661	127.0.0.1		127.0.0.1	TCP	118	50479 → 53 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=74 [TCP segment of a reassembled PDU]
12	16.493677	127.0.0.1		127.0.0.1	TCP	44	53 → 50479 [ACK] Seq=1 Ack=75 Win=2619648 Len=0
13	16.777412	127.0.0.1		127.0.0.1	TCP	149	53 → 50479 [PSH, ACK] Seq=1 Ack=75 Win=2619648 Len=105 [TCP segment of a reassembled PDU]
14	16.777438	127.0.0.1		127.0.0.1	TCP	44	50479 → 53 [ACK] Seq=75 Ack=106 Win=2619648 Len=0
15	16.777474	127.0.0.1		127.0.0.1	TCP	44	53 → 50479 [FIN, ACK] Seq=106 Ack=75 Win=2619648 Len=0
16	16.777483	127.0.0.1		127.0.0.1	TCP	44	50479 → 53 [ACK] Seq=75 Ack=107 Win=2619648 Len=0
17	16.778448	127.0.0.1		127.0.0.1	TCP	44	50479 → 53 [FIN, ACK] Seq=75 Ack=107 Win=2619648 Len=0
18	16.778503	127.0.0.1		127.0.0.1	TCP	44	53 → 50479 [ACK] Seq=107 Ack=76 Win=2619648 Len=0

Interface: Wi-Fi

UDP connection, local DNS server with the DNS server of Google

The local DNS server sends the client's DNS query to the google DNS server(query is 'www.google.com').

The Google DNS server receive with a response to the query



dns && udp.port == 53 && (ip.src == 8.8.8.8 || ip.dst == 8.8.8.8)

No.	Time	Source	Flags	Destination	Protocol	Length	Info
171	21.061934	192.168.43.48	0x0100	8.8.8.8	DNS	74	Standard query 0x0000 A www.google.com
172	21.239208	8.8.8.8	0x8180	192.168.43.48	DNS	90	Standard query response 0x0000 A www.google.com A 172.217.16.228

Client

```
[*] Enter '1' to assign an IP address
[*] Enter '2' for a DNS request
[*] Enter '3' to send SQL queries
[*] Enter '4' to exit
2
Enter a domain name: www.google.com
Sent the query to the DNS server
www.google.com resolved to 172.217.16.228
```

```
DNS server is listening..
Received a DNS query from the client
Built the DNS query packet, forwarding to the DNS server of google
Sent the DNS query packet to a remote DNS server
Sent the response to the client
```

Local DNS server

3.3 Application – Server start

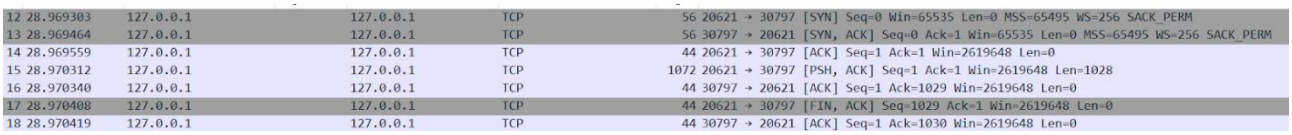
Interface: loopback

TCP connection, client with the application server

TCP handshake client-app server.

The client sends a packet including a string with the desirable protocol(TCP or RUDP).

The server receive the packet. Then close the TCP connection



12	28.969303	127.0.0.1	127.0.0.1	TCP	56 20621 → 30797 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
13	28.969464	127.0.0.1	127.0.0.1	TCP	56 30797 → 20621 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
14	28.969559	127.0.0.1	127.0.0.1	TCP	44 20621 → 30797 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
15	28.970312	127.0.0.1	127.0.0.1	TCP	1072 20621 → 30797 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=1028
16	28.970340	127.0.0.1	127.0.0.1	TCP	44 30797 → 20621 [ACK] Seq=1 Ack=1029 Win=2619648 Len=0
17	28.970408	127.0.0.1	127.0.0.1	TCP	44 20621 → 30797 [FIN, ACK] Seq=1029 Ack=1 Win=2619648 Len=0
18	28.970419	127.0.0.1	127.0.0.1	TCP	44 30797 → 20621 [ACK] Seq=1 Ack=1030 Win=2619648 Len=0

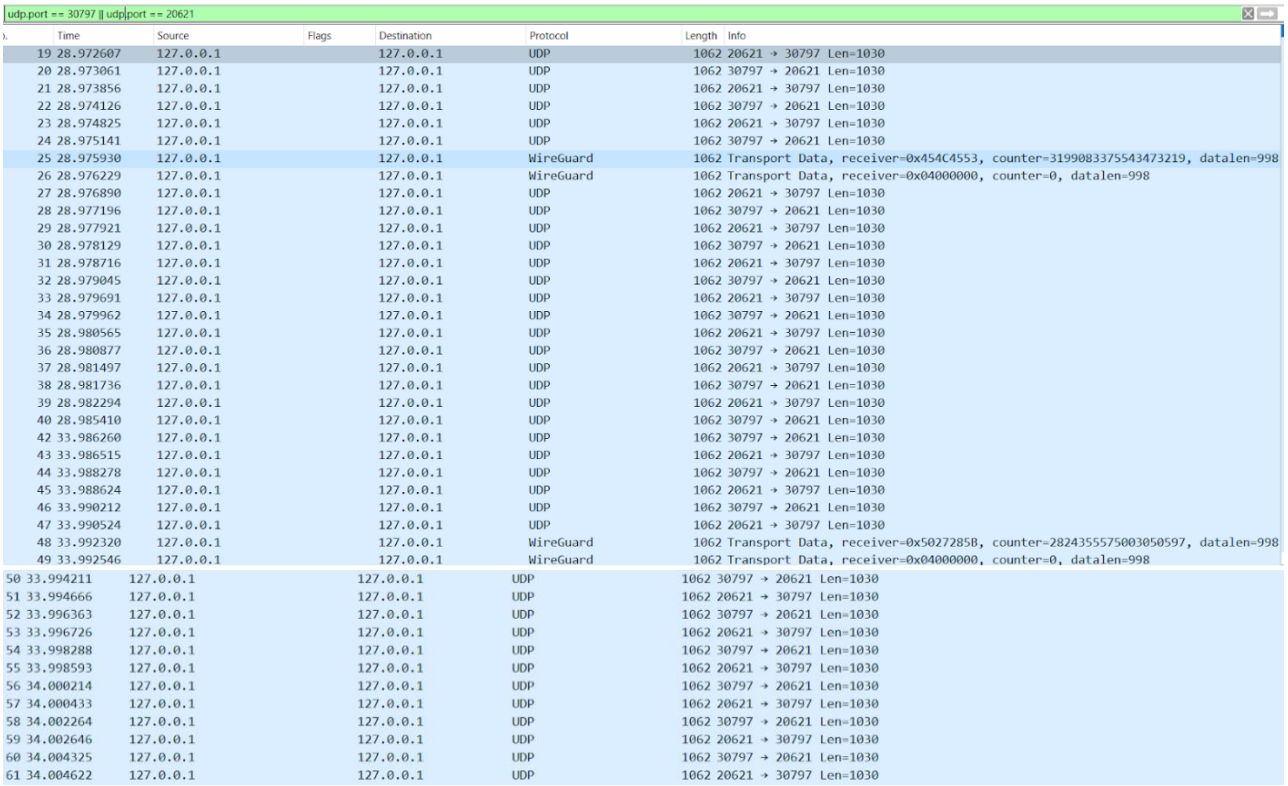
3.3.1 Application – RUDP

As RUDP is on top of UDP protocol, there is no early data transfer.

The client sends the queries packets to the server.

The server response with ACK packets for the queries packets.

The server received 10 queries, then sends the responses for them to the client



udp.port == 30797 || udp.port == 20621

No.	Time	Source	Flags	Destination	Protocol	Length	Info
19	28.972607	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
20	28.973061	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
21	28.973856	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
22	28.974126	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
23	28.974825	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
24	28.975141	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
25	28.975930	127.0.0.1		127.0.0.1	WireGuard	1062	Transport Data, receiver=0x454C4553, counter=3199083375543473219, datalen=998
26	28.976229	127.0.0.1		127.0.0.1	WireGuard	1062	Transport Data, receiver=0x04000000, counter=0, datalen=998
27	28.976890	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
28	28.977196	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
29	28.977921	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
30	28.978129	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
31	28.978716	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
32	28.979045	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
33	28.979691	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
34	28.979962	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
35	28.980565	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
36	28.980877	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
37	28.981497	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
38	28.981736	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
39	28.982294	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
40	28.985410	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
42	33.986260	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
43	33.986515	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
44	33.988278	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
45	33.988624	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
46	33.990212	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
47	33.990524	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
48	33.992320	127.0.0.1		127.0.0.1	WireGuard	1062	Transport Data, receiver=0x5027285B, counter=2824355575003050597, datalen=998
49	33.992546	127.0.0.1		127.0.0.1	WireGuard	1062	Transport Data, receiver=0x04000000, counter=0, datalen=998
50	33.994211	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
51	33.994666	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
52	33.996363	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
53	33.996726	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
54	33.998288	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
55	33.998593	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
56	34.000214	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
57	34.000433	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
58	34.002264	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
59	34.002646	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030
60	34.004325	127.0.0.1		127.0.0.1	UDP	1062	30797 → 20621 Len=1030
61	34.004622	127.0.0.1		127.0.0.1	UDP	1062	20621 → 30797 Len=1030

Client

1

```
[*] Enter '1' to assign an IP address
[*] Enter '2' for a DNS request
[*] Enter '3' to send SQL queries
[*] Enter '4' to exit
3
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter protocol 'TCP'/'RUDP': RUDP
Opening TCP socket and connecting the server
Connected to the server
Received an ACK for packet: 1
Received an ACK for packet: 2
Received an ACK for packet: 3
Received an ACK for packet: 4
Received an ACK for packet: 5
Received an ACK for packet: 6
Received an ACK for packet: 7
Received an ACK for packet: 8
Received an ACK for packet: 9
Received an ACK for packet: 10
Received a response for query num: 10
Seq num: 1, Window: [1, 5]
```

```
Sent an ACK packet for packet: 1
Window start: 1, received_packets: {1: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 9
Seq num: 2, Window: [2, 6]
Sent an ACK packet for packet: 2
Window start: 2, received_packets: {2: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 8
Seq num: 3, Window: [3, 7]
Sent an ACK packet for packet: 3
Window start: 3, received_packets: {3: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 7
Seq num: 4, Window: [4, 8]
Sent an ACK packet for packet: 4
Window start: 4, received_packets: {4: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 6
Seq num: 5, Window: [5, 9]
Sent an ACK packet for packet: 5
Window start: 5, received_packets: {5: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 5
Seq num: 6, Window: [6, 10]
Sent an ACK packet for packet: 6
Window start: 6, received_packets: {6: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 4
Seq num: 7, Window: [7, 11]
Sent an ACK packet for packet: 7
Window start: 7, received_packets: {7: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 3
Seq num: 8, Window: [8, 12]
Sent an ACK packet for packet: 8
Window start: 8, received_packets: {8: "[('Bob Smith', 7000.0)]"}
```

2

```
Received a response for query num: 2
Seq num: 9, Window: [9, 13]
Sent an ACK packet for packet: 9
Window start: 9, received_packets: {9: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 1
Seq num: 10, Window: [10, 14]
Sent an ACK packet for packet: 10
Window start: 10, received_packets: {10: "[('Bob Smith', 7000.0)]"}
Response to query 1: [('Bob Smith', 7000.0)]
Response to query 2: [('Bob Smith', 7000.0)]
Response to query 3: [('Bob Smith', 7000.0)]
Response to query 4: [('Bob Smith', 7000.0)]
Response to query 5: [('Bob Smith', 7000.0)]
Response to query 6: [('Bob Smith', 7000.0)]
Response to query 7: [('Bob Smith', 7000.0)]
Response to query 8: [('Bob Smith', 7000.0)]
Response to query 9: [('Bob Smith', 7000.0)]
Response to query 10: [('Bob Smith', 7000.0)]
```

1

Server

1

```
Application server started
Application server is listening..
Accept occurred.
Opening RUDP connection
Received a new query!
Seq num: 1, Window: [1, 5]
Sent an ACK packet for packet: 1
Window start: 1, received_packets: {1: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 2, Window: [2, 6]
Sent an ACK packet for packet: 2
Window start: 2, received_packets: {2: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 3, Window: [3, 7]
Sent an ACK packet for packet: 3
Window start: 3, received_packets: {3: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 4, Window: [4, 8]
Sent an ACK packet for packet: 4
Window start: 4, received_packets: {4: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 5, Window: [5, 9]
Sent an ACK packet for packet: 5
Window start: 5, received_packets: {5: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 6, Window: [6, 10]
Sent an ACK packet for packet: 6
Window start: 6, received_packets: {6: 'SELECT name, salary FROM emp WHERE salary > 6000'}
```

2

```
Seq num: 7, Window: [7, 11]
Sent an ACK packet for packet: 7
Window start: 7, received_packets: {7: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 8, Window: [8, 12]
Sent an ACK packet for packet: 8
Window start: 8, received_packets: {8: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 9, Window: [9, 13]
Sent an ACK packet for packet: 9
Window start: 9, received_packets: {9: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 10, Window: [10, 14]
Sent an ACK packet for packet: 10
Window start: 10, received_packets: {10: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Executing the query and send the response back to the client
Result of query 1 = [('Bob Smith', 7000.0)] has been sent
Timeout occurred for packet:  1
Received an ACK for packet: 1
Executing the query and send the response back to the client
Result of query 2 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 2
Executing the query and send the response back to the client
Result of query 3 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 3
Executing the query and send the response back to the client
Result of query 4 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 4
```

3

```
Executing the query and send the response back to the client
Result of query 5 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 5
Executing the query and send the response back to the client
Result of query 6 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 6
Executing the query and send the response back to the client
Result of query 7 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 7
Executing the query and send the response back to the client
Result of query 8 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 8
Executing the query and send the response back to the client
Result of query 9 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 9
Executing the query and send the response back to the client
Result of query 10 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 10
Closing RUDP connection..
Application server is listening..
```

3.3.2 Application – TCP

TCP handshake between the client and the application server.

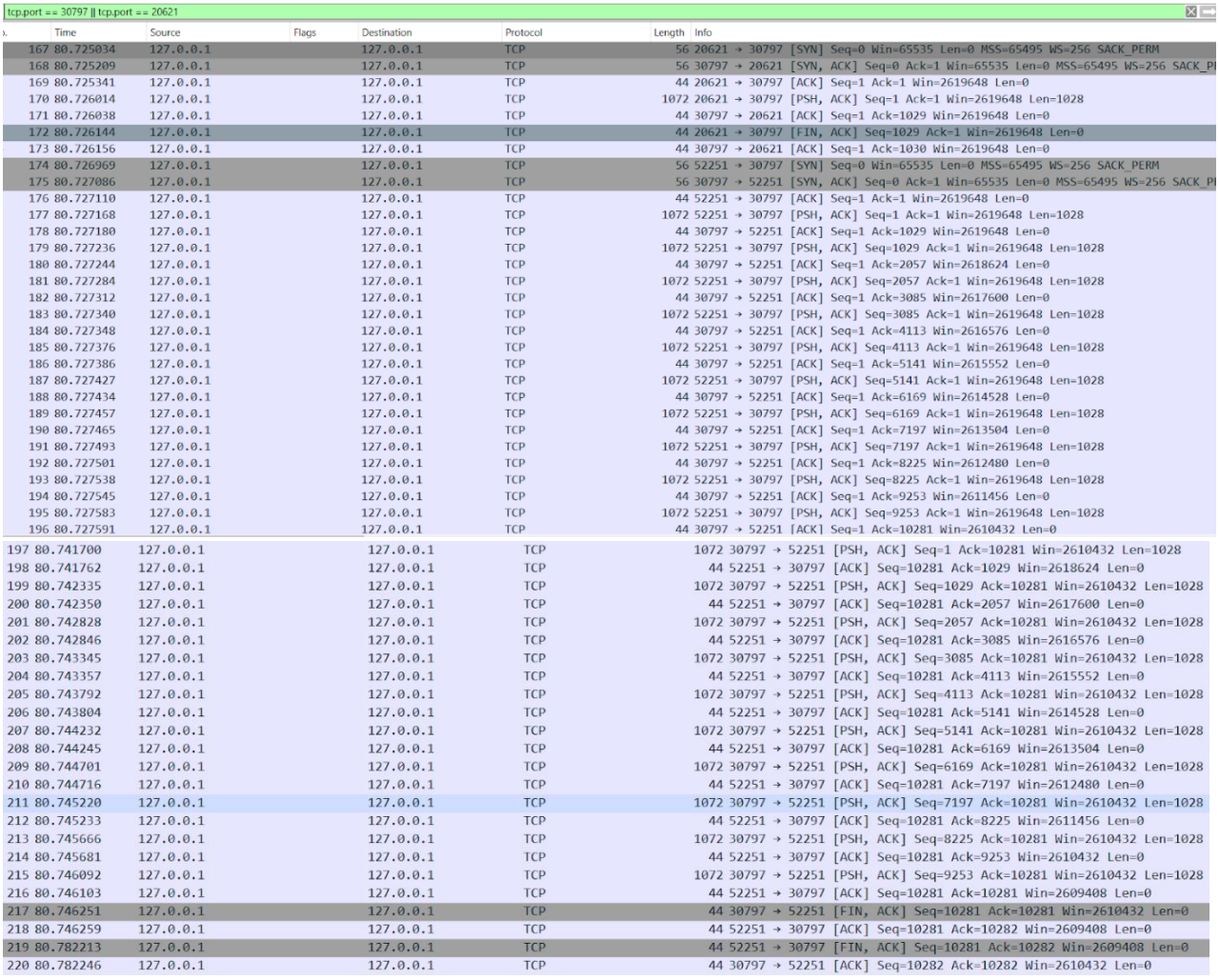
The client sends queries packets to the server.

The server response with ACK packets for the queries packets.

10 query packets transmitted, then the server sends the responses packets.

The client response with ACK packets.

Closing TCP connection.



tcp.port == 30797 || tcp.port == 20621

	Time	Source	Flags	Destination	Protocol	Length	Info
167	80.725034	127.0.0.1		127.0.0.1	TCP	56	20621 → 30797 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
168	80.725209	127.0.0.1		127.0.0.1	TCP	56	30797 → 20621 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_P
169	80.725341	127.0.0.1		127.0.0.1	TCP	44	20621 → 30797 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
170	80.726014	127.0.0.1		127.0.0.1	TCP	1072	20621 → 30797 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=1028
171	80.726038	127.0.0.1		127.0.0.1	TCP	44	30797 → 20621 [ACK] Seq=1 Ack=1029 Win=2619648 Len=0
172	80.726144	127.0.0.1		127.0.0.1	TCP	44	20621 → 30797 [FIN, ACK] Seq=1029 Ack=1 Win=2619648 Len=0
173	80.726156	127.0.0.1		127.0.0.1	TCP	44	30797 → 20621 [ACK] Seq=1 Ack=1030 Win=2619648 Len=0
174	80.726969	127.0.0.1		127.0.0.1	TCP	56	52251 → 30797 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
175	80.727086	127.0.0.1		127.0.0.1	TCP	56	30797 → 52251 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_P
176	80.727110	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
177	80.727168	127.0.0.1		127.0.0.1	TCP	1072	52251 → 30797 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=1028
178	80.727180	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=1 Ack=1029 Win=2619648 Len=0
179	80.727236	127.0.0.1		127.0.0.1	TCP	1072	52251 → 30797 [PSH, ACK] Seq=1029 Ack=1 Win=2619648 Len=1028
180	80.727244	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=1 Ack=2057 Win=2618624 Len=0
181	80.727284	127.0.0.1		127.0.0.1	TCP	1072	52251 → 30797 [PSH, ACK] Seq=2057 Ack=1 Win=2619648 Len=1028
182	80.727312	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=1 Ack=3085 Win=2617600 Len=0
183	80.727340	127.0.0.1		127.0.0.1	TCP	1072	52251 → 30797 [PSH, ACK] Seq=3085 Ack=1 Win=2619648 Len=1028
184	80.727348	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=1 Ack=4113 Win=2616576 Len=0
185	80.727376	127.0.0.1		127.0.0.1	TCP	1072	52251 → 30797 [PSH, ACK] Seq=4113 Ack=1 Win=2619648 Len=1028
186	80.727386	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=1 Ack=5141 Win=2615552 Len=0
187	80.727427	127.0.0.1		127.0.0.1	TCP	1072	52251 → 30797 [PSH, ACK] Seq=5141 Ack=1 Win=2619648 Len=1028
188	80.727434	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=1 Ack=6169 Win=2614528 Len=0
189	80.727457	127.0.0.1		127.0.0.1	TCP	1072	52251 → 30797 [PSH, ACK] Seq=6169 Ack=1 Win=2619648 Len=1028
190	80.727465	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=1 Ack=7197 Win=2613504 Len=0
191	80.727493	127.0.0.1		127.0.0.1	TCP	1072	52251 → 30797 [PSH, ACK] Seq=7197 Ack=1 Win=2619648 Len=1028
192	80.727501	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=1 Ack=8225 Win=2612480 Len=0
193	80.727538	127.0.0.1		127.0.0.1	TCP	1072	52251 → 30797 [PSH, ACK] Seq=8225 Ack=1 Win=2619648 Len=1028
194	80.727545	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=1 Ack=9253 Win=2611456 Len=0
195	80.727583	127.0.0.1		127.0.0.1	TCP	1072	52251 → 30797 [PSH, ACK] Seq=9253 Ack=1 Win=2619648 Len=1028
196	80.727591	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=1 Ack=10281 Win=2610432 Len=0
197	80.741700	127.0.0.1		127.0.0.1	TCP	1072	30797 → 52251 [PSH, ACK] Seq=1 Ack=10281 Win=2610432 Len=1028
198	80.741762	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=1029 Win=2618624 Len=0
199	80.742335	127.0.0.1		127.0.0.1	TCP	1072	30797 → 52251 [PSH, ACK] Seq=1029 Ack=10281 Win=2610432 Len=1028
200	80.742350	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=2057 Win=2617600 Len=0
201	80.742828	127.0.0.1		127.0.0.1	TCP	1072	30797 → 52251 [PSH, ACK] Seq=2057 Ack=10281 Win=2610432 Len=1028
202	80.742846	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=3085 Win=2616576 Len=0
203	80.743345	127.0.0.1		127.0.0.1	TCP	1072	30797 → 52251 [PSH, ACK] Seq=3085 Ack=10281 Win=2610432 Len=1028
204	80.743357	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=4113 Win=2615552 Len=0
205	80.743792	127.0.0.1		127.0.0.1	TCP	1072	30797 → 52251 [PSH, ACK] Seq=4113 Ack=10281 Win=2610432 Len=1028
206	80.743804	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=5141 Win=2614528 Len=0
207	80.744232	127.0.0.1		127.0.0.1	TCP	1072	30797 → 52251 [PSH, ACK] Seq=5141 Ack=10281 Win=2610432 Len=1028
208	80.744245	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=6169 Win=2613504 Len=0
209	80.744701	127.0.0.1		127.0.0.1	TCP	1072	30797 → 52251 [PSH, ACK] Seq=6169 Ack=10281 Win=2610432 Len=1028
210	80.744716	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=7197 Win=2612480 Len=0
211	80.745220	127.0.0.1		127.0.0.1	TCP	1072	30797 → 52251 [PSH, ACK] Seq=7197 Ack=10281 Win=2610432 Len=1028
212	80.745233	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=8225 Win=2611456 Len=0
213	80.745666	127.0.0.1		127.0.0.1	TCP	1072	30797 → 52251 [PSH, ACK] Seq=8225 Ack=10281 Win=2610432 Len=1028
214	80.745681	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=9253 Win=2610432 Len=0
215	80.746092	127.0.0.1		127.0.0.1	TCP	1072	30797 → 52251 [PSH, ACK] Seq=9253 Ack=10281 Win=2610432 Len=1028
216	80.746103	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=10281 Win=2609408 Len=0
217	80.746251	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [FIN, ACK] Seq=10281 Ack=10281 Win=2610432 Len=0
218	80.746259	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [ACK] Seq=10281 Ack=10282 Win=2609408 Len=0
219	80.782213	127.0.0.1		127.0.0.1	TCP	44	52251 → 30797 [FIN, ACK] Seq=10281 Ack=10282 Win=2609408 Len=0
220	80.782246	127.0.0.1		127.0.0.1	TCP	44	30797 → 52251 [ACK] Seq=10282 Ack=10282 Win=2610432 Len=0

Client

1

```
[*] Enter '1' to assign an IP address
[*] Enter '2' for a DNS request
[*] Enter '3' to send SQL queries
[*] Enter '4' to exit
3
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter protocol 'TCP'/'RUDP': TCP
Opening TCP socket and connecting the server
Connected to the server
Opening TCP socket and connecting the server
Sent query 1 to the server
Sent query 2 to the server
Sent query 3 to the server
Sent query 4 to the server
Sent query 5 to the server
Sent query 6 to the server
Sent query 7 to the server
Sent query 8 to the server
```

2

```
Sent query 9 to the server
Sent query 10 to the server
Received response for query 1
Received response for query 2
Received response for query 3
Received response for query 4
Received response for query 5
Received response for query 6
Received response for query 7
Received response for query 8
Received response for query 9
Received response for query 10
Response to query 1: [('Bob Smith', 7000.0)]
Response to query 2: [('Bob Smith', 7000.0)]
Response to query 3: [('Bob Smith', 7000.0)]
Response to query 4: [('Bob Smith', 7000.0)]
Response to query 5: [('Bob Smith', 7000.0)]
Response to query 6: [('Bob Smith', 7000.0)]
Response to query 7: [('Bob Smith', 7000.0)]
Response to query 8: [('Bob Smith', 7000.0)]
Response to query 9: [('Bob Smith', 7000.0)]
Response to query 10: [('Bob Smith', 7000.0)]
```

Server

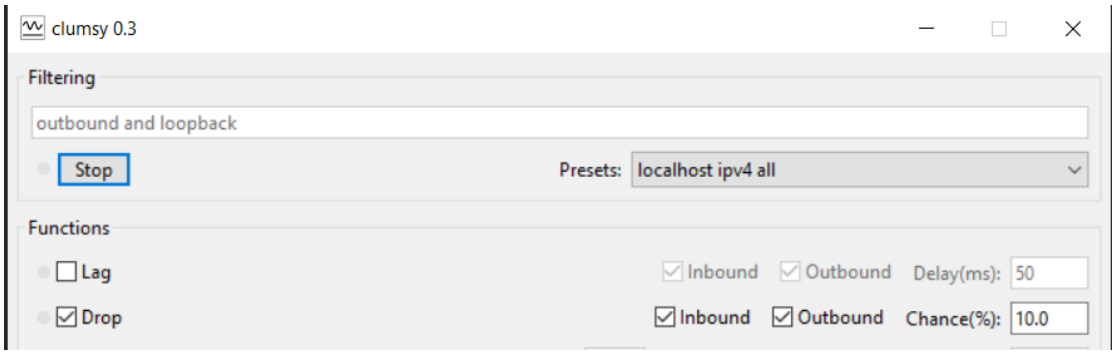
3

```
Application server started
Application server is listening..
Accept occurred.
Server listening..
Accept occurred
Query 1 arrived and stored
Query 2 arrived and stored
Query 3 arrived and stored
Query 4 arrived and stored
Query 5 arrived and stored
Query 6 arrived and stored
Query 7 arrived and stored
Query 8 arrived and stored
Query 9 arrived and stored
Query 10 arrived and stored
Executing query 1 and sending the response to the client
Executing query 2 and sending the response to the client
Executing query 3 and sending the response to the client
Executing query 4 and sending the response to the client
Executing query 5 and sending the response to the client
Executing query 6 and sending the response to the client
Executing query 7 and sending the response to the client
Executing query 8 and sending the response to the client
Executing query 9 and sending the response to the client
Executing query 10 and sending the response to the client
Closing TCP connection..
```

3.3.3 Application – RUDP with packet loss of 10%

We used clumsy 0.3 program to illustrate packet loss, in this case of 10%.

(clumsy 0.3 - https://github.com/jagt/clumsy/releases/tag/0.3rc4)



Same process as described before about the RUDP traffic (10 query packets from the client to the server, 10 response packets from the server to the client).

However, since there are lost packets, there is more traffic with the same packets, as the sender doesn't receive an ACK packet for the sent packet, then retransmit the same packet again.

12	10.604830	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
13	10.605396	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
14	10.606065	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
15	10.606393	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
16	10.607078	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
17	10.607391	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
18	10.608001	127.0.0.1	127.0.0.1	WireGuard	1062 Transport Data, receiver=0x454C4553, counter=3199083375543473219, datalen=998
19	10.608392	127.0.0.1	127.0.0.1	WireGuard	1062 Transport Data, receiver=0x04000000, counter=0, datalen=998
20	10.609023	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
21	10.609333	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
22	10.609915	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
26	15.621999	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
27	15.622350	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
28	15.623170	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
29	15.623501	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
30	15.624212	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
31	15.624534	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
32	15.625095	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
34	20.630783	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
35	20.631196	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
36	20.632096	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
39	25.640853	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
40	25.641159	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
41	30.653567	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
42	30.659528	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
45	35.672528	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
46	35.672948	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
47	35.675464	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
48	35.675861	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
49	35.677419	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
50	35.677742	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
51	35.679467	127.0.0.1	127.0.0.1	WireGuard	1062 Transport Data, receiver=0x5027285B, counter=2824355575003050597, datalen=998
52	35.679723	127.0.0.1	127.0.0.1	WireGuard	1062 Transport Data, receiver=0x04000000, counter=0, datalen=998
53	35.681435	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
54	35.681687	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
55	35.683437	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
56	35.683708	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
57	35.685448	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
58	35.685749	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
59	35.687464	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
60	35.687746	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
61	35.689419	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
62	35.689757	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030
63	35.691401	127.0.0.1	127.0.0.1	UDP	1062 30797 → 20621 Len=1030
64	35.691764	127.0.0.1	127.0.0.1	UDP	1062 20621 → 30797 Len=1030

1

Client

```
3
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter a SQL query: SELECT name, salary FROM emp WHERE salary > 6000
Enter protocol 'TCP'/'RUDP': RUDP
Opening TCP socket and connecting the server
Connected to the server
Received an ACK for packet: 1
Received an ACK for packet: 2
Received an ACK for packet: 3
Received an ACK for packet: 4
Received an ACK for packet: 5
Received an ACK for packet: 6
Timeout occurred for packet:  7
Timeout occurred for packet:  7
Timeout occurred for packet:  7
Received an ACK for packet: 7
Received an ACK for packet: 8
Received an ACK for packet: 9
Received an ACK for packet: 10
Received a response for query num: 10
```

Timeout

```
Seq num: 1, Window: [1, 5]
Sent an ACK packet for packet: 1
Window start: 1, received_packets: {1: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 9
Seq num: 2, Window: [2, 6]
Sent an ACK packet for packet: 2
Window start: 2, received_packets: {2: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 8
Seq num: 3, Window: [3, 7]
Sent an ACK packet for packet: 3
Window start: 3, received_packets: {3: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 7
Seq num: 4, Window: [4, 8]
Sent an ACK packet for packet: 4
Window start: 4, received_packets: {4: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 6
Seq num: 5, Window: [5, 9]
Sent an ACK packet for packet: 5
Window start: 5, received_packets: {5: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 5
Seq num: 6, Window: [6, 10]
Sent an ACK packet for packet: 6
Window start: 6, received_packets: {6: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 4
Seq num: 7, Window: [7, 11]
Sent an ACK packet for packet: 7
Window start: 7, received_packets: {7: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 3
```

2

```
Seq num: 8, Window: [8, 12]
Sent an ACK packet for packet: 8
Window start: 8, received_packets: {8: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 2
Seq num: 9, Window: [9, 13]
Sent an ACK packet for packet: 9
Window start: 9, received_packets: {9: "[('Bob Smith', 7000.0)]"}
Received a response for query num: 1
Seq num: 10, Window: [10, 14]
Sent an ACK packet for packet: 10
Window start: 10, received_packets: {10: "[('Bob Smith', 7000.0)]"}
Response to query 1: [('Bob Smith', 7000.0)]
Response to query 2: [('Bob Smith', 7000.0)]
Response to query 3: [('Bob Smith', 7000.0)]
Response to query 4: [('Bob Smith', 7000.0)]
Response to query 5: [('Bob Smith', 7000.0)]
Response to query 6: [('Bob Smith', 7000.0)]
Response to query 7: [('Bob Smith', 7000.0)]
Response to query 8: [('Bob Smith', 7000.0)]
Response to query 9: [('Bob Smith', 7000.0)]
Response to query 10: [('Bob Smith', 7000.0)]
```

3

1

Server

```
Application server started
Application server is listening..
Accept occurred.
Opening RUDP connection
Received a new query!
Seq num: 1, Window: [1, 5]
Sent an ACK packet for packet: 1
Window start: 1, received_packets: {1: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 2, Window: [2, 6]
Sent an ACK packet for packet: 2
Window start: 2, received_packets: {2: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 3, Window: [3, 7]
Sent an ACK packet for packet: 3
Window start: 3, received_packets: {3: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 4, Window: [4, 8]
Sent an ACK packet for packet: 4
Window start: 4, received_packets: {4: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 5, Window: [5, 9]
Sent an ACK packet for packet: 5
Window start: 5, received_packets: {5: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 6, Window: [6, 10]
Sent an ACK packet for packet: 6
```

```
Window start: 6, received_packets: {6: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 7, Window: [7, 11]
Sent an ACK packet for packet: 7
Window start: 7, received_packets: {7: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 7, Window: [8, 12]
Sent an ACK packet for packet: 7
Received a new query!
Seq num: 8, Window: [8, 12]
Sent an ACK packet for packet: 8
Window start: 8, received_packets: {8: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 9, Window: [9, 13]
Sent an ACK packet for packet: 9
Window start: 9, received_packets: {9: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Seq num: 10, Window: [10, 14]
Sent an ACK packet for packet: 10
Window start: 10, received_packets: {10: 'SELECT name, salary FROM emp WHERE salary > 6000'}
Received a new query!
Executing the query and send the response back to the client
Result of query 1 = [('Bob Smith', 7000.0)] has been sent
Timeout occurred for packet:  1
Timeout occurred for packet:  1
Received an ACK for packet: 1
Executing the query and send the response back to the client
Result of query 2 = [('Bob Smith', 7000.0)] has been sent
```

2

Timeout

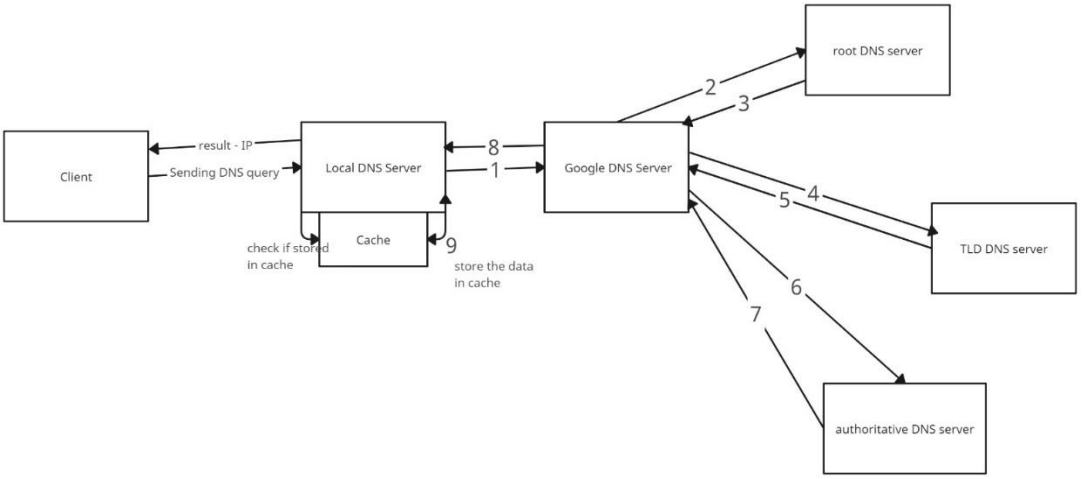
```
Timeout occurred for packet:  2
Received an ACK for packet: 2
Executing the query and send the response back to the client
Result of query 3 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 3
Executing the query and send the response back to the client
Result of query 4 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 4
Executing the query and send the response back to the client
Result of query 5 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 5
Executing the query and send the response back to the client
Result of query 6 = [('Bob Smith', 7000.0)] has been sent
Timeout occurred for packet:  6
Received an ACK for packet: 6
Executing the query and send the response back to the client
Result of query 7 = [('Bob Smith', 7000.0)] has been sent
Timeout occurred for packet:  7
Received an ACK for packet: 7
Executing the query and send the response back to the client
Result of query 8 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 8
Executing the query and send the response back to the client
Result of query 9 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 9
Executing the query and send the response back to the client
Result of query 10 = [('Bob Smith', 7000.0)] has been sent
Received an ACK for packet: 10
Closing RUDP connection..
```

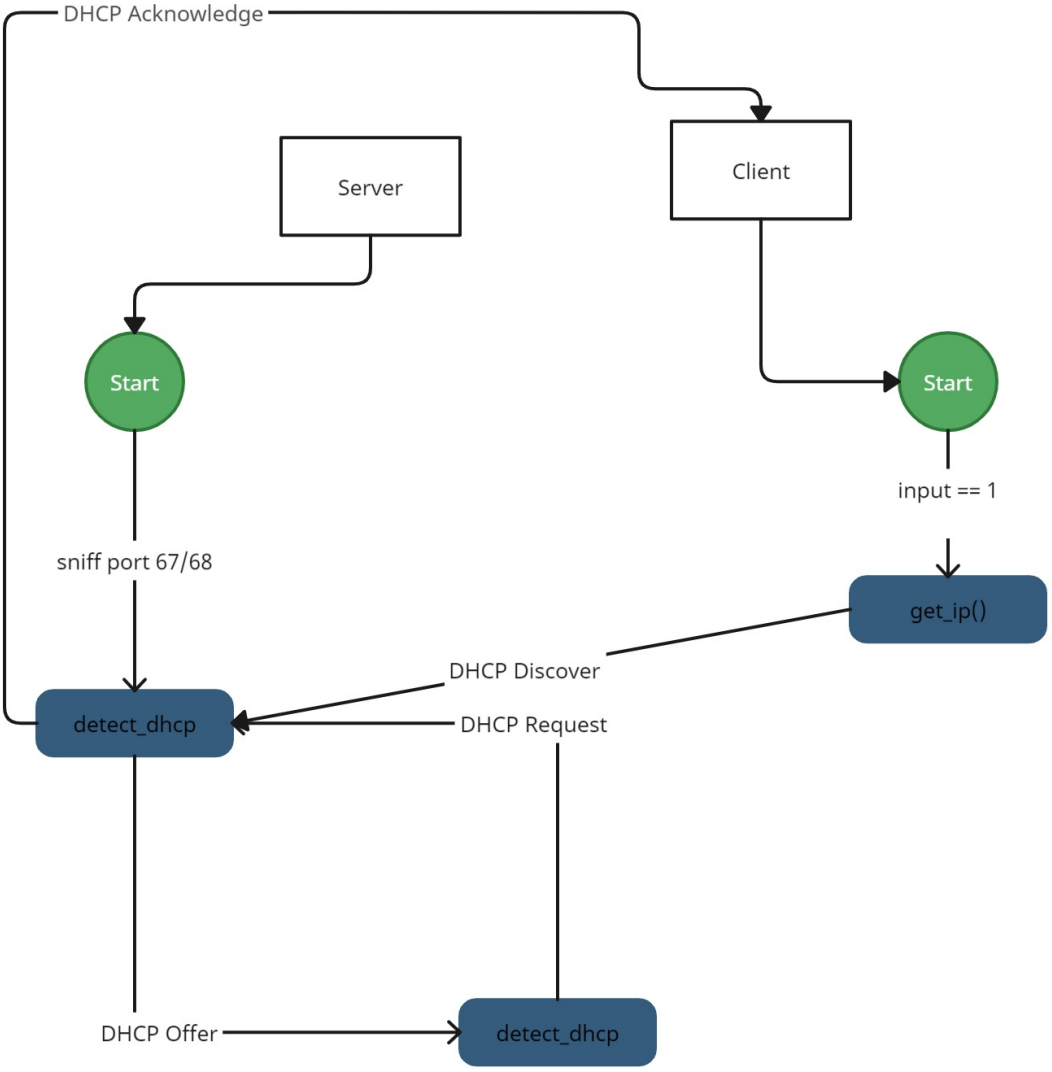
3

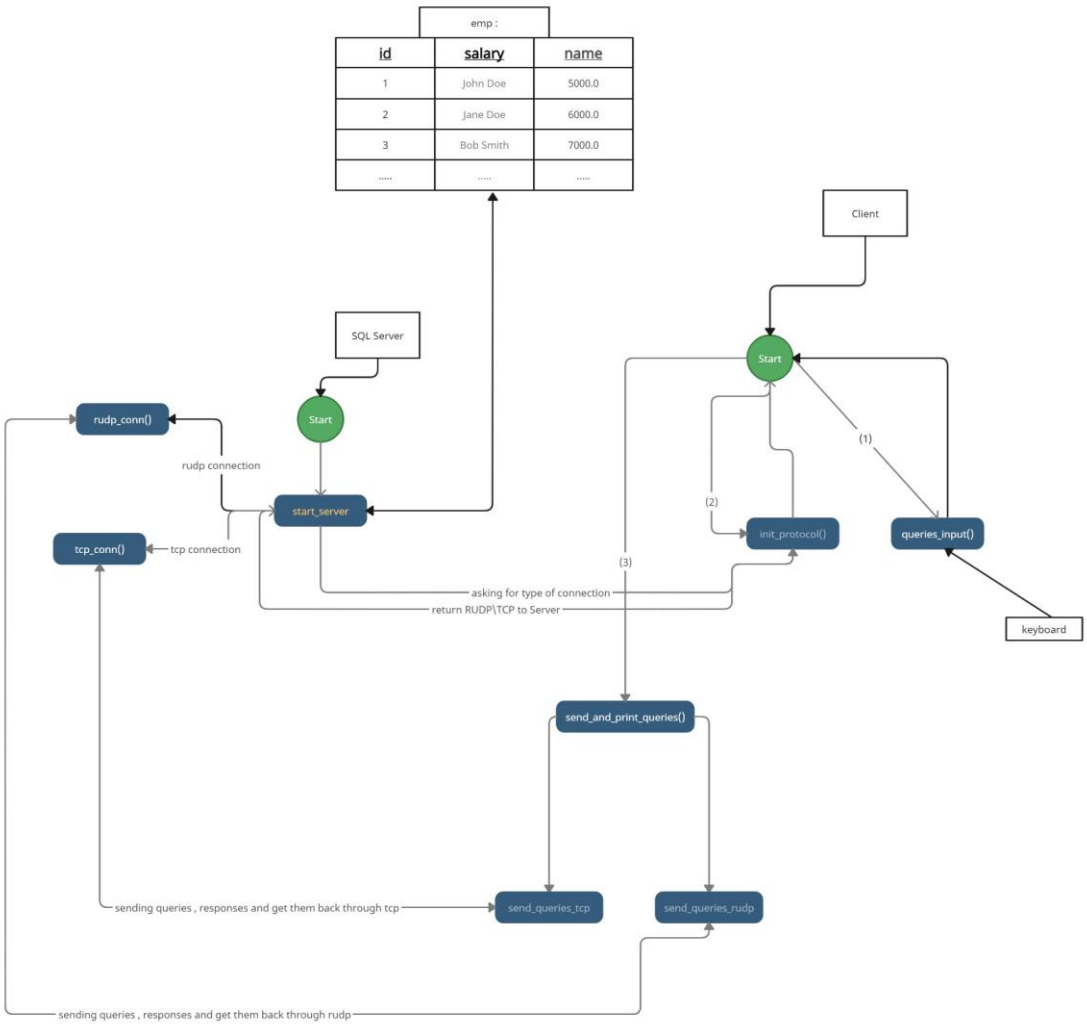
Timeout

Timeout

4. State Diagrams-






emp :
id
salary
name
1
John Doe
5000.0
2
Jane Doe
6000.0
3
Bob Smith
7000.0
.....
.....
.....
Client
SQL Server
Start
Start
rudp_conn()
rudp connection
start_server
tcp_conn()
tcp connection
(1)
(2)
(3)
init_protocol()
queries_input()
keyboard
asking for type of connection
return RUDP\TCP to Server
send_and_print_queries()
sending queries , responses and get them back through tcp
send_queries_tcp
send_queries_rudp
sending queries , responses and get them back through rudp

References-

[1] "DNS and ARP Security Vulnerabilities and Mitigation Techniques" by A. Hussain, R. Singh, and D. K. Singh. (published in 2017)

[2] "Comparative Analysis of ARP, DNS and DHCP Security Vulnerabilities in Computer Networks" by R. K. Vatsa and R. K. Jangra. (published in 2018)

[3] "The Differences Between OSPF and BGP" by J. Davis. (Published in 2019)

[4] "Does BGP Really Work? An Empirical Assessment" by M. Luckie, B. Huffaker, and K. claffy. (Published in 2013)

[5] "A Comparison of Cubic and Vegas TCP Congestion Control" by J. Leddy and R. Sivakumar. (Published in 2012)

[6] "Comparing the Performance of Cubic and Veno over High Bandwidth-Delay Product Networks" by P. Balakrishnan and V. S. Pai. (Published in 2008)

[7] "QUIC vs TCP+TLS: Analyzing the tradeoff between security and performance" by J. C. Cardona and S. S. Roy. (Published in 2020)

[8] "A Comparative Study of TCP and QUIC for HTTP/2 Server Push" by F. Khalid, J. H. Lee, and B. Rhee. (Published in 2017)