**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- 3](#_Toc129622376)

[*1.1 DHCP Server 3*](#_Toc129622377)

[*1.2 DNS Server 4*](#_Toc129622378)

[*1.3 Application – SQL Server 5*](#_Toc129622379)

[2. Methods- 9](#_Toc129622380)

[3. Wireshark- 14](#_Toc129622381)

[*3.1 DHCP 14*](#_Toc129622382)

[*3.2 DNS 15*](#_Toc129622383)

[*3.3 Application – Server start 16*](#_Toc129622384)

[*3.3.1 Application – RUDP 17*](#_Toc129622385)

[*3.3.2 Application – TCP 20*](#_Toc129622386)

[*3.3.3 Application – RUDP with packet loss of 10% 22*](#_Toc129622387)

[4. State Diagrams- 26](#_Toc129622388)

# *Introducing-*

## *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.

1. **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.

1. **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.

1. **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*

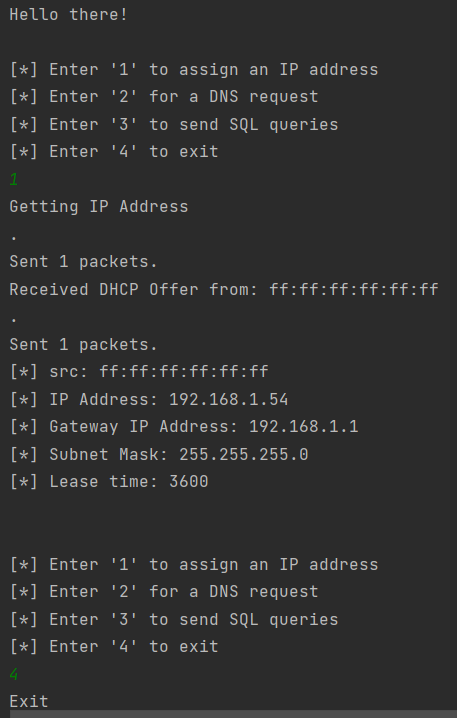
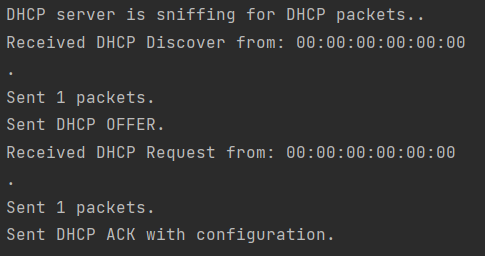
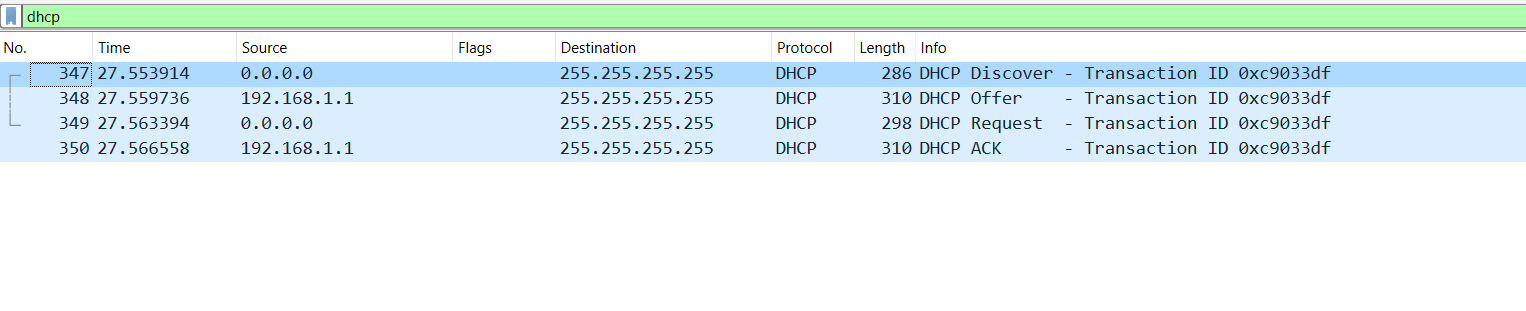
DHCP server

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.



Client

## *3.2 DNS*

Interface: loopback

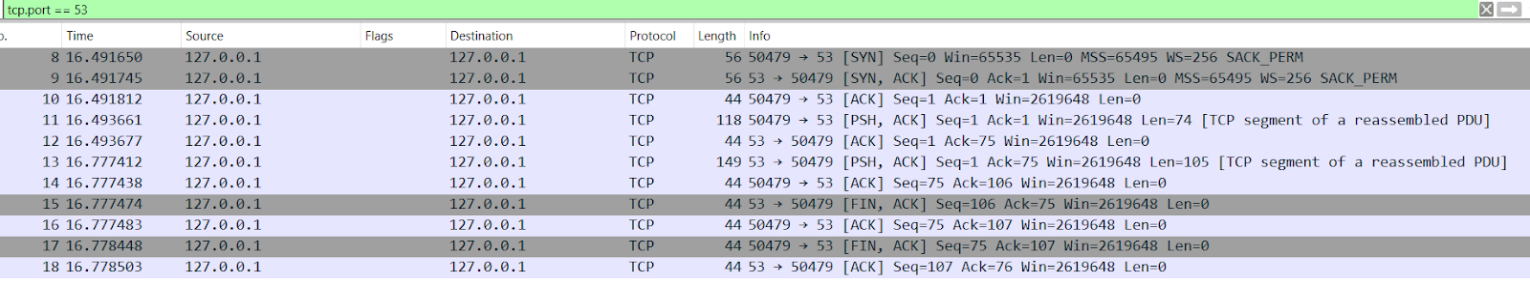
TCP connection, client with the local DNS server

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

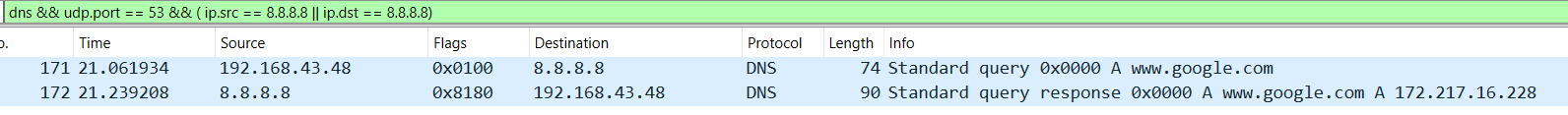


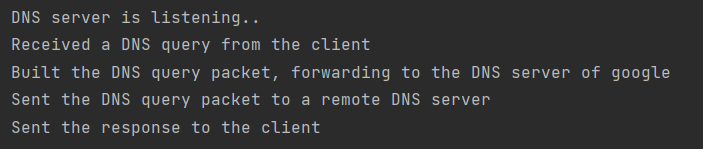
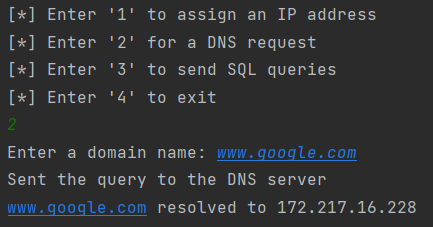
Interface: Wi-Fi

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





Local DNS server

Client

## *3.3 Application – Server start*

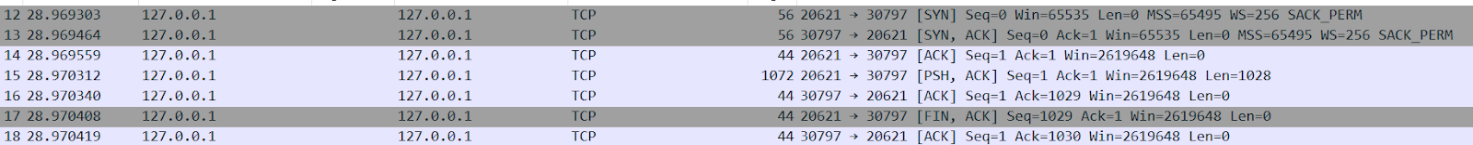
Interface: loopback

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



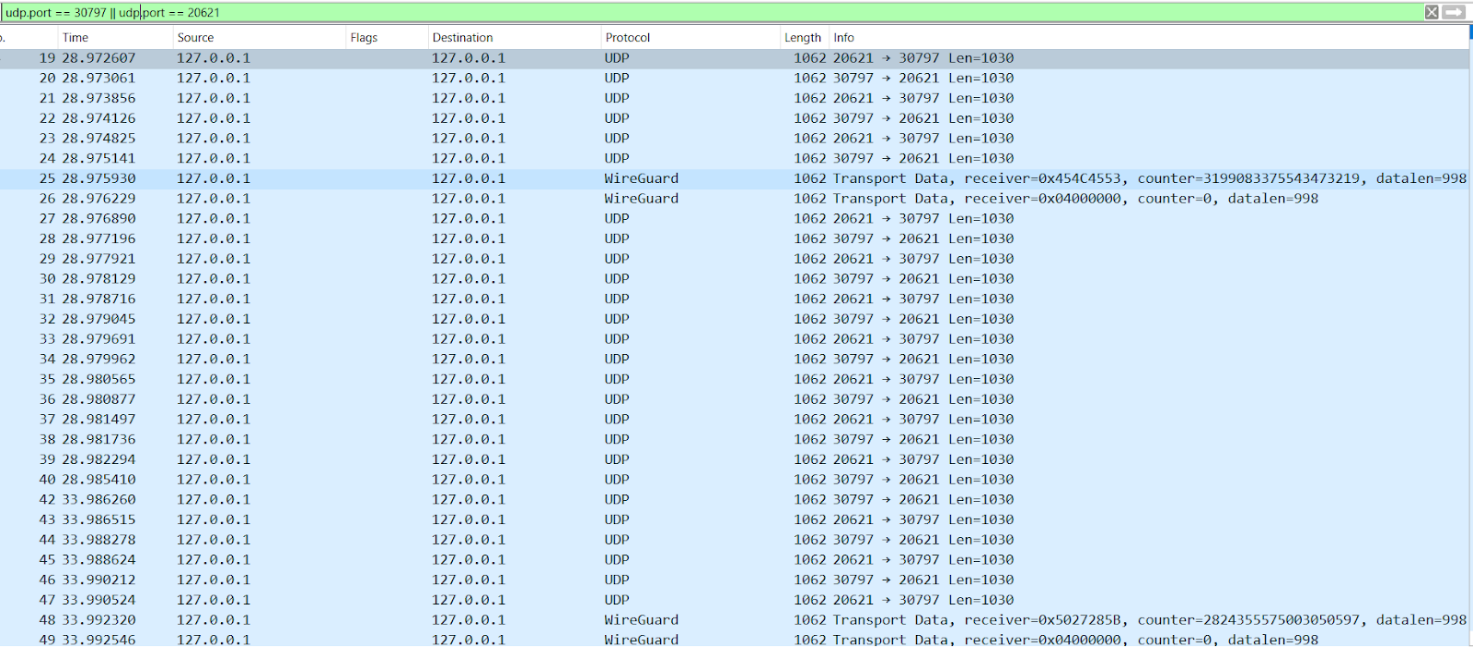
## *3.3.1 Application – RUDP*

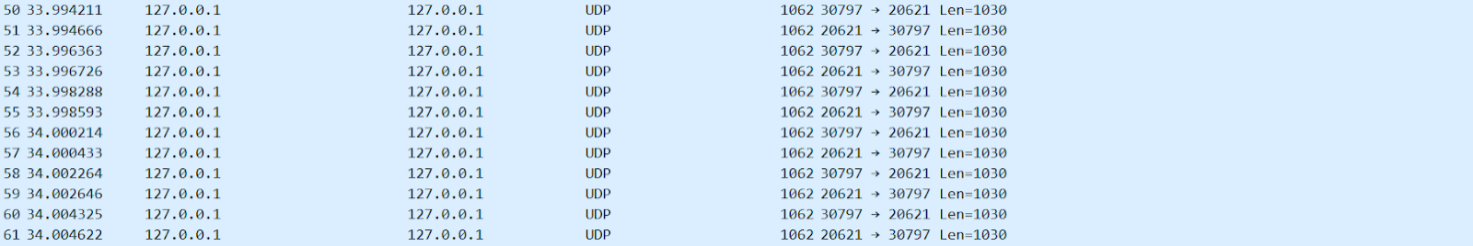
As RUDP is on top of UDP protocol, there is no early data transfer.

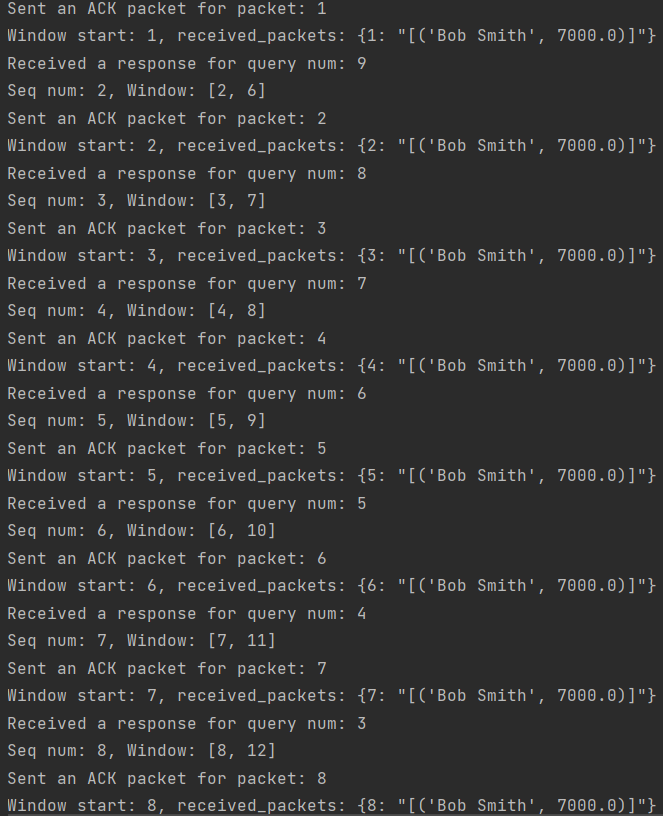
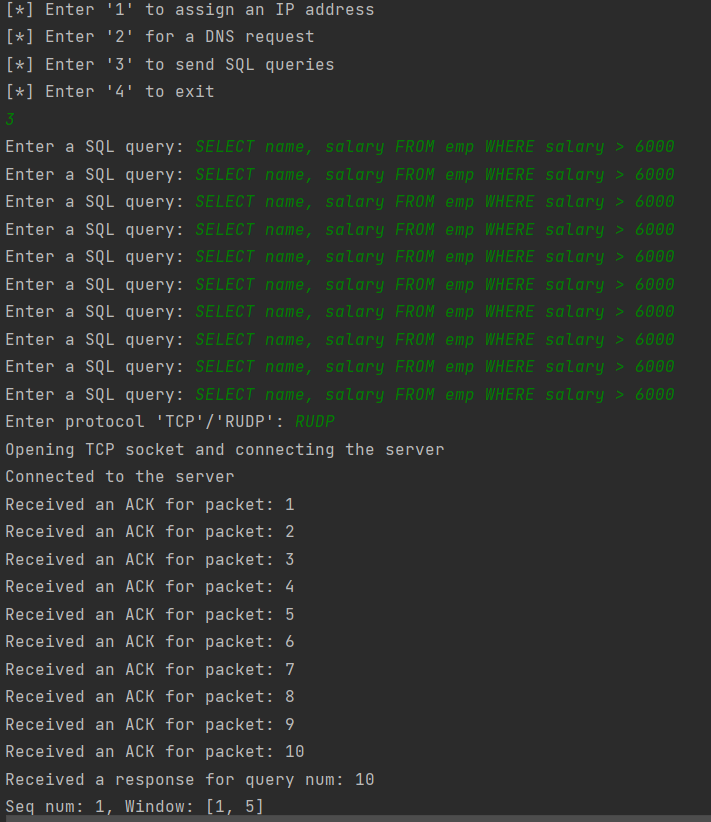
The client sends the queries packets to the server.

The server response with ACK packets for the queries packets.

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



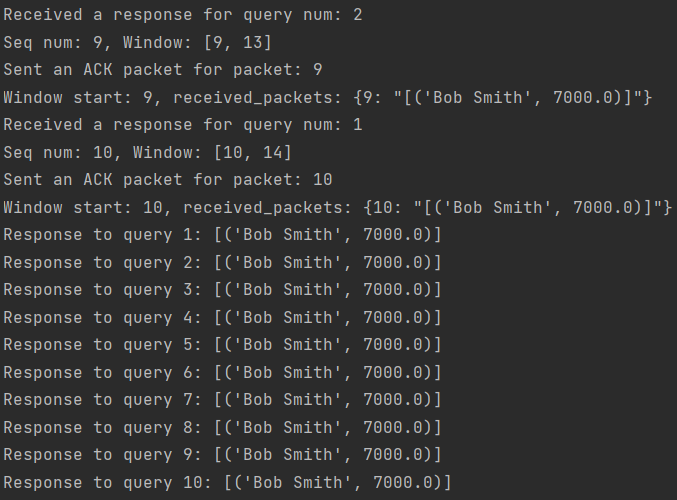




**2**

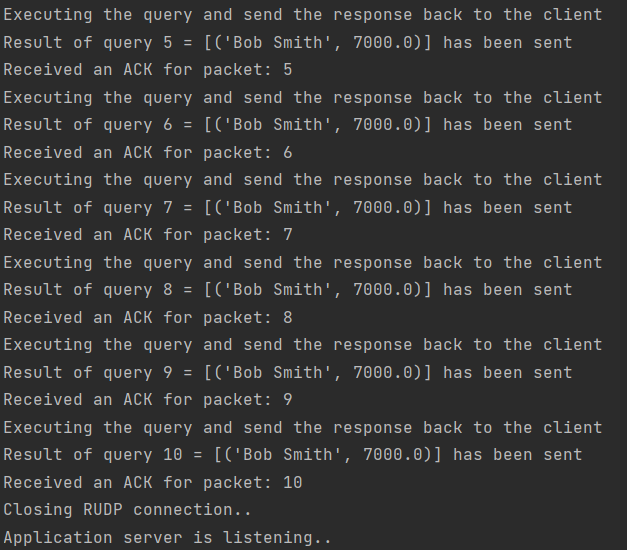
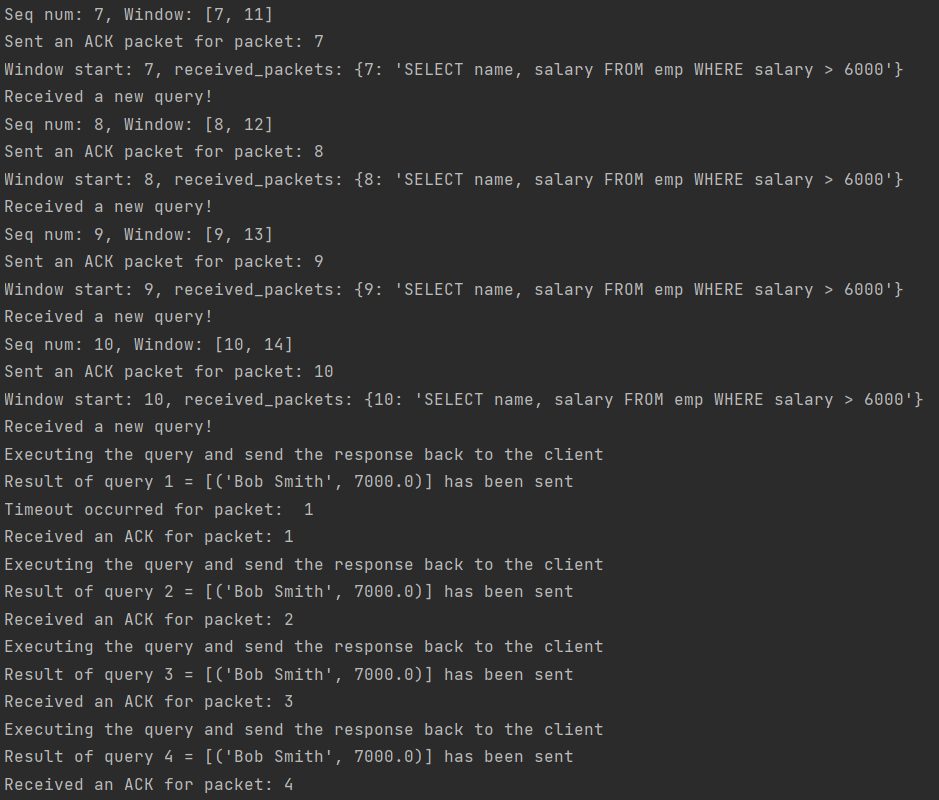
Client

**1**



**1**

**1**



**3**

**2**

Server

## *3.3.2 Application – TCP*

TCP handshake between the client and the application server.

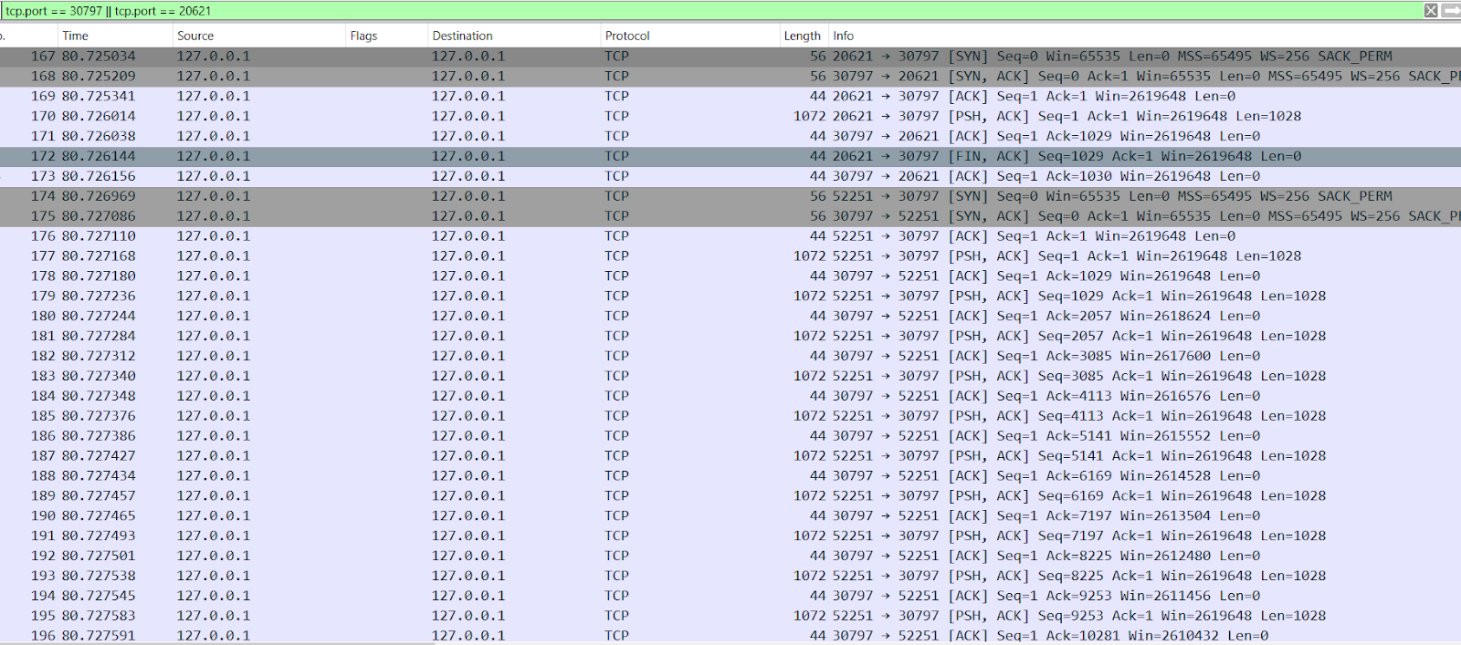
The client sends queries packets to the server.

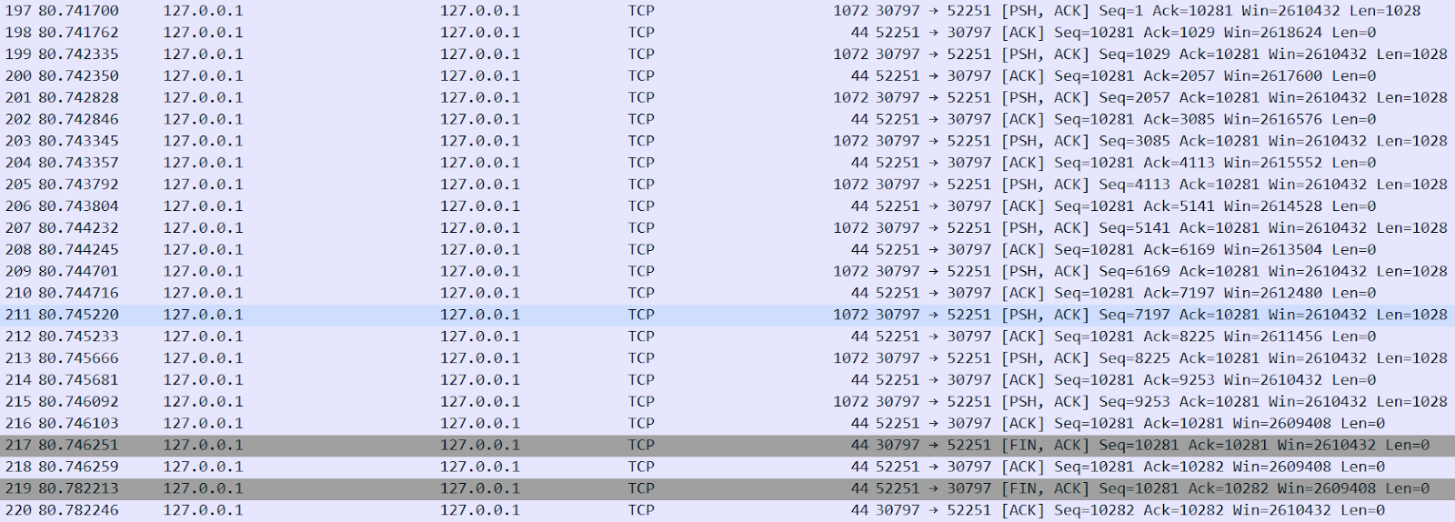
The server response with ACK packets for the queries packets.

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

The client response with ACK packets.

Closing TCP connection.

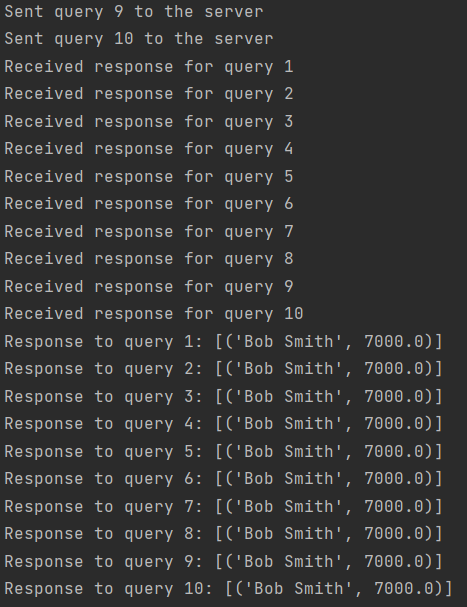
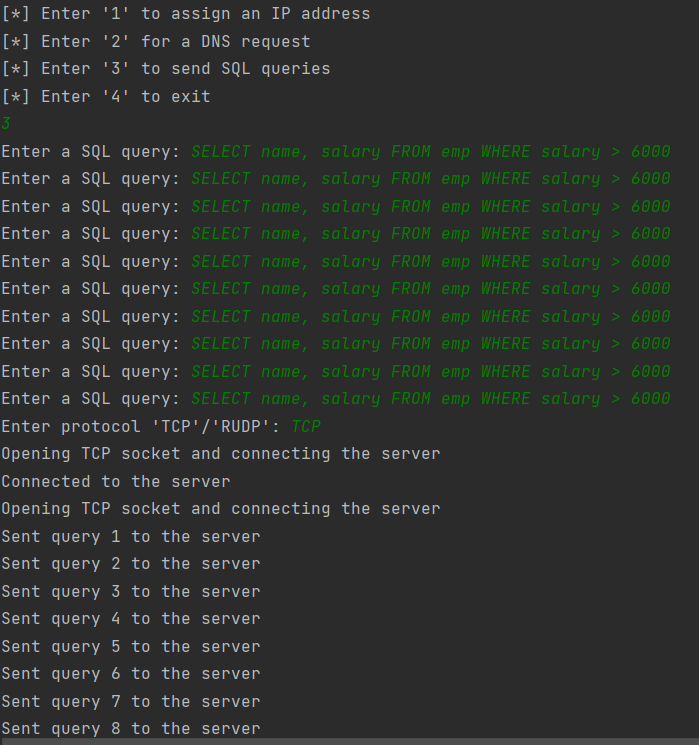


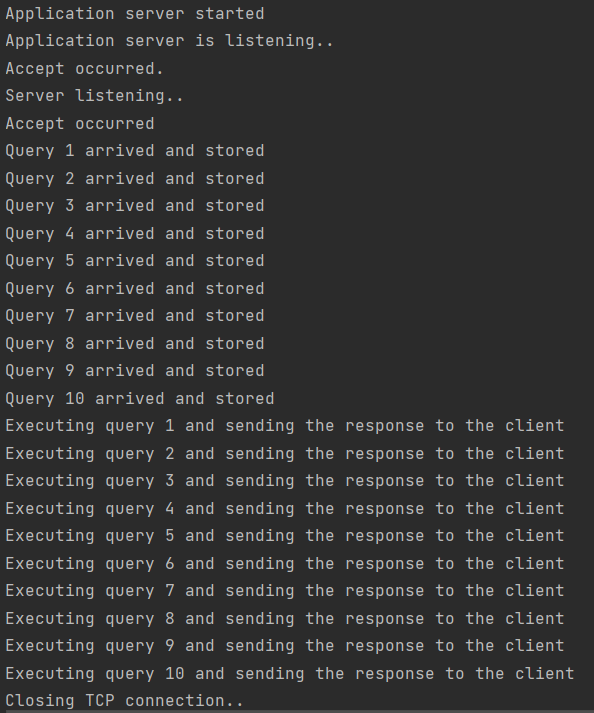


**2**

**1**

Client





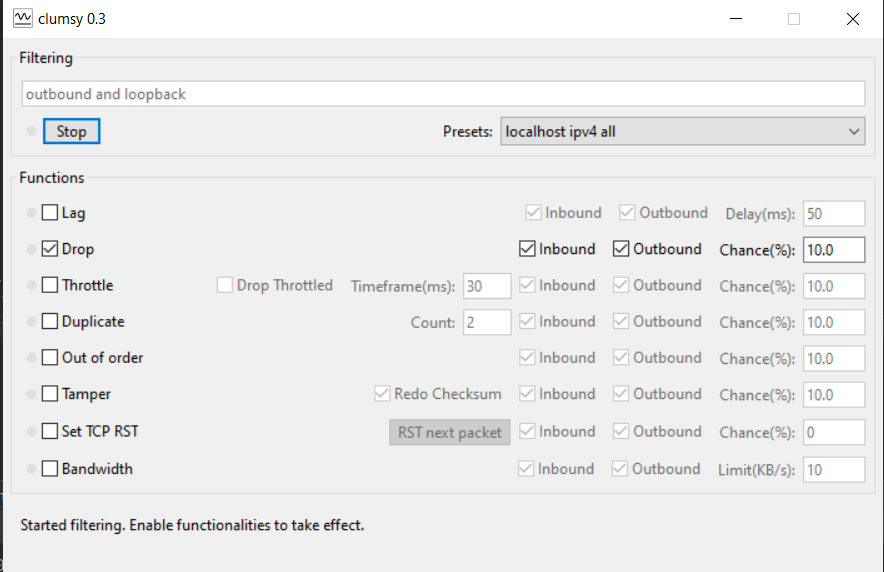
**3**

Server

## *3.3.3 Application – RUDP with packet loss of 10%*

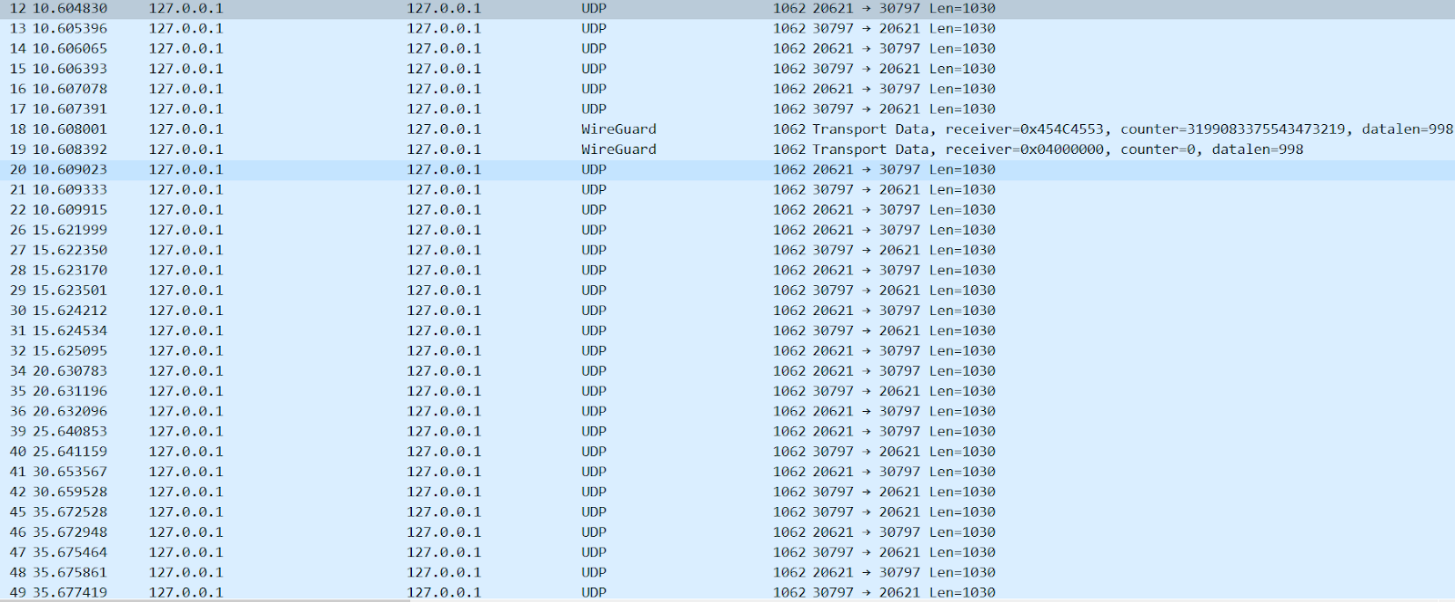
We used clumsy 0.3 program to illustrate packet loss, in this case of 10%.

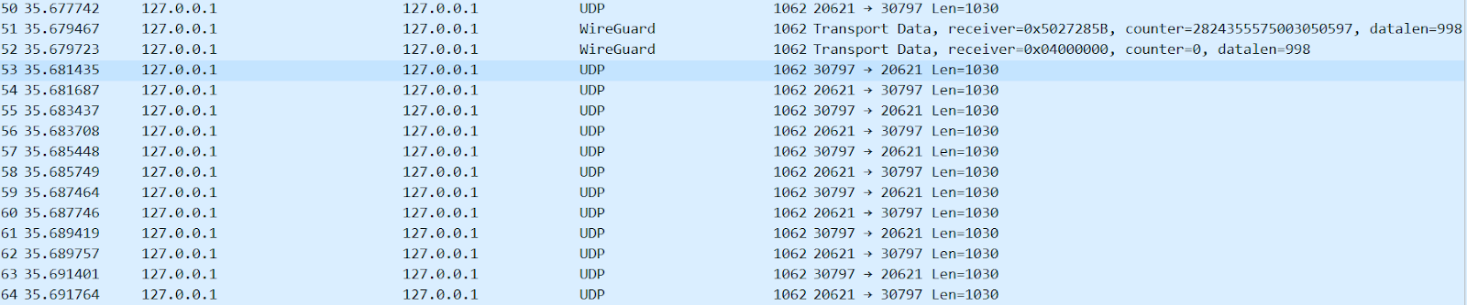
(clumsy 0.3 - <https://github.com/jagt/clumsy/releases/tag/0.3rc4>)

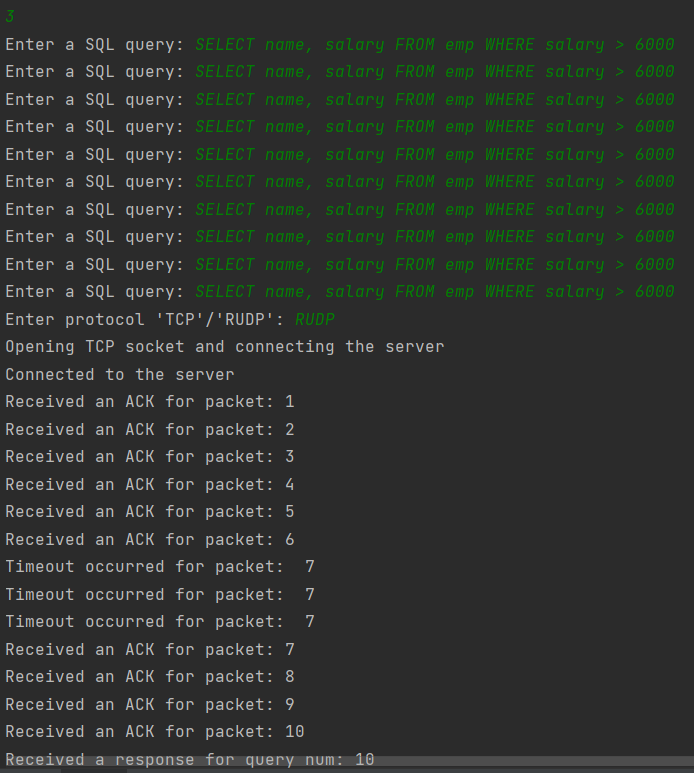


Same process as described before about the RUDP traffic (10 query packets from the client to the server, 10 response packets from the server to the client).

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.



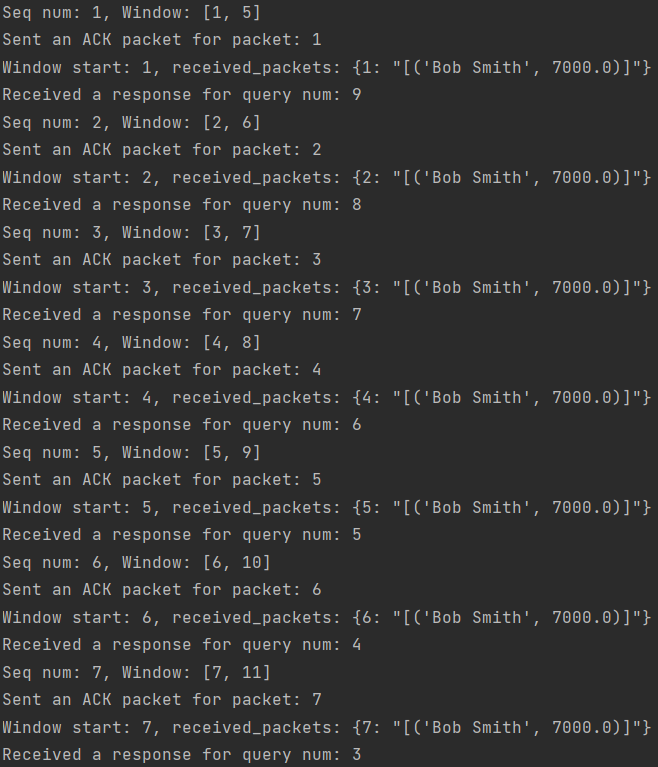




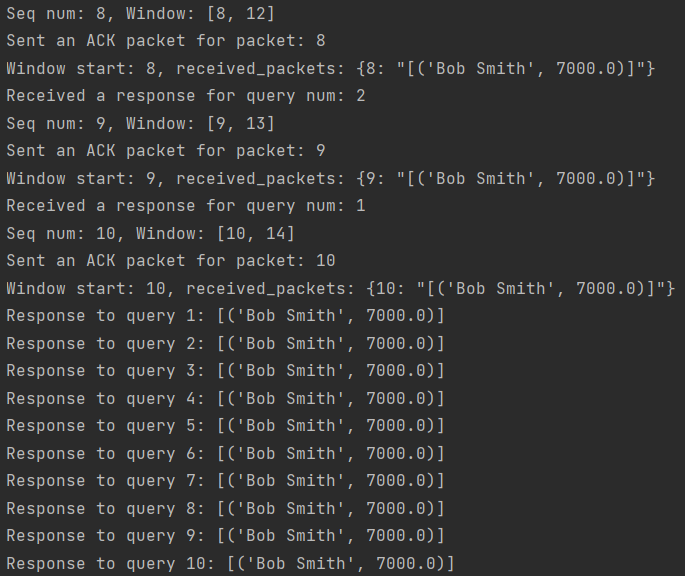
**1**

Client

**Timeout**

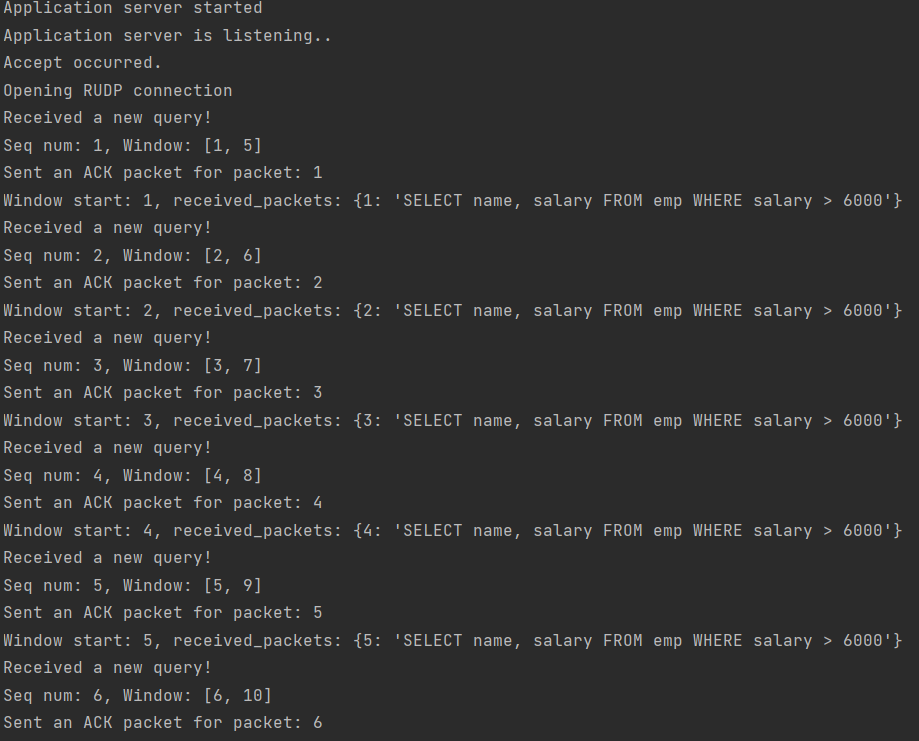


**2**



**3**

**1**

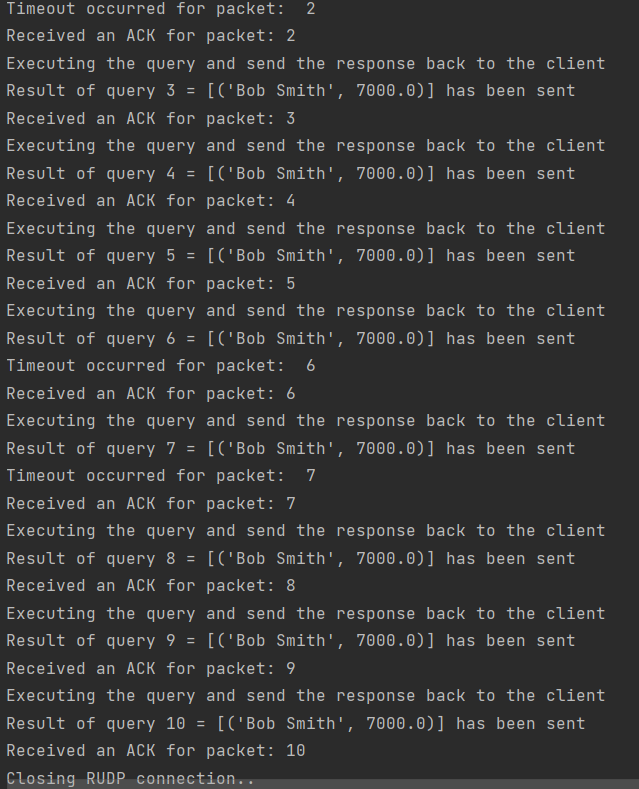


Server



**2**

**Timeout**

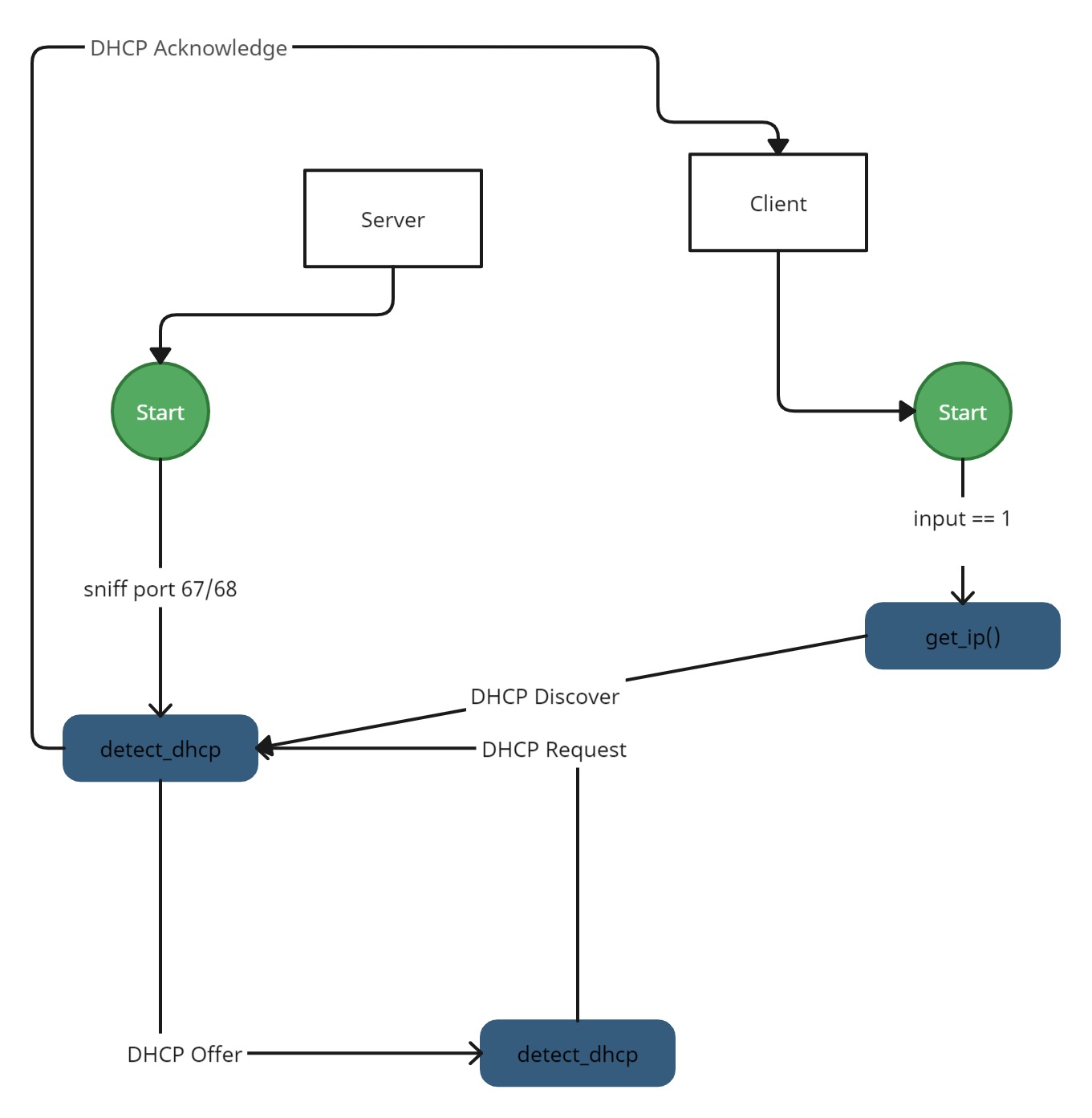
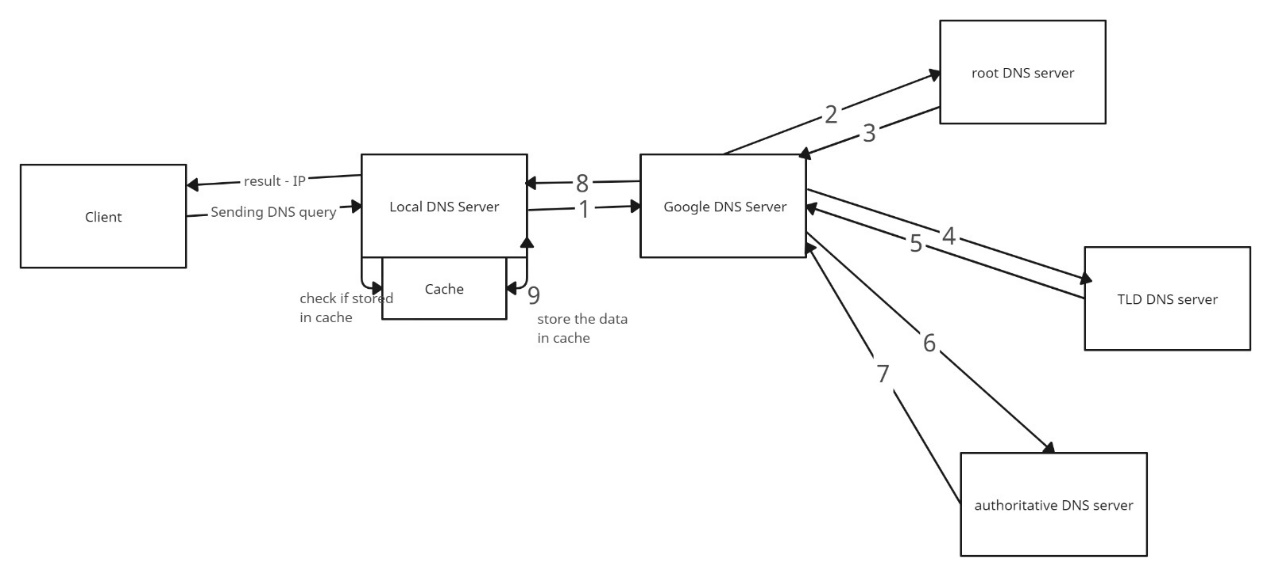


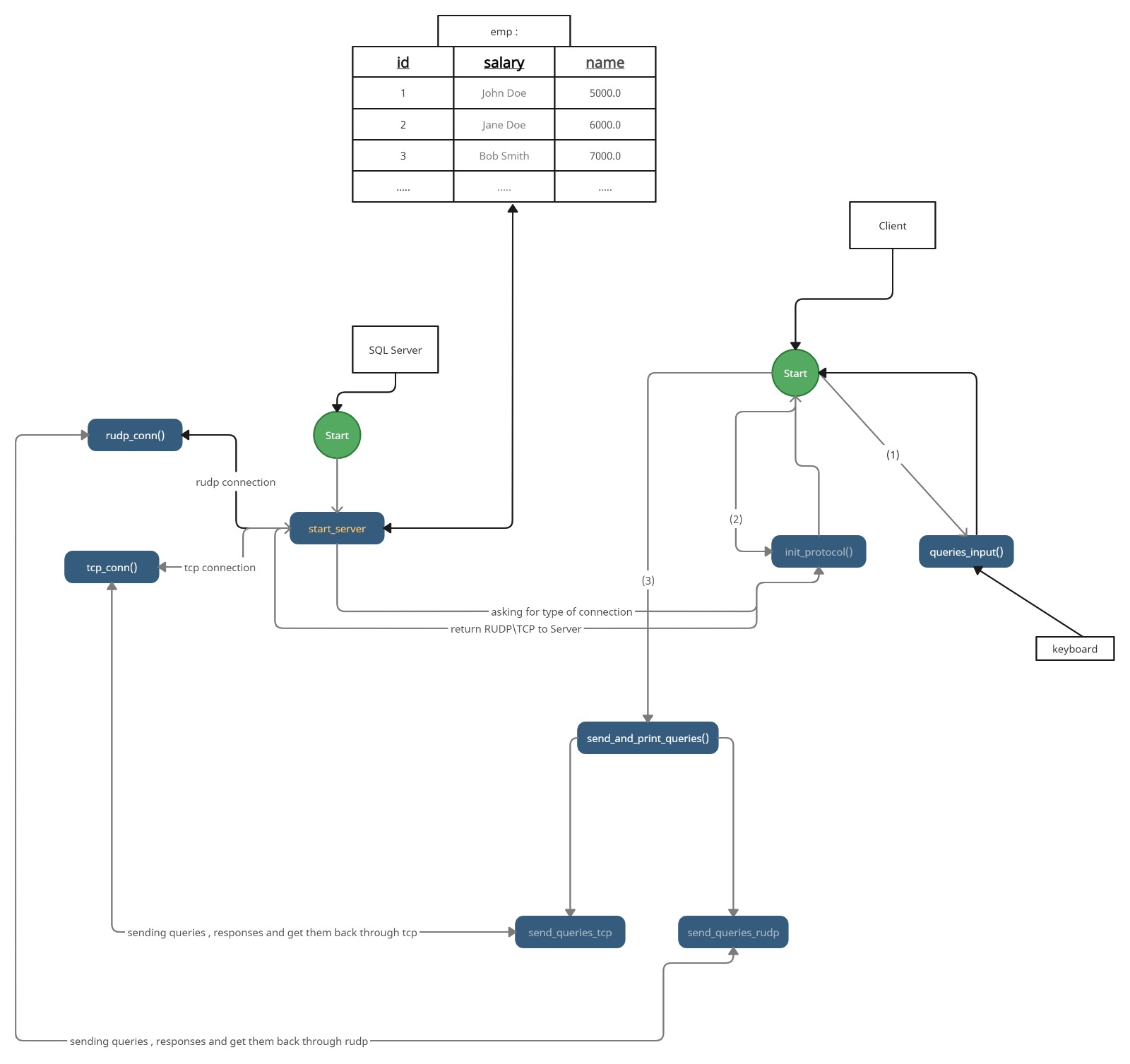
**3**

**Timeout**

**Timeout**

# *4. State Diagrams-*

**



*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)