HTTP APPLICATION

CN\_FINALL

Ariel University

Connection networking

Maor Saadon 318532421

Duvi Amiram 305677494

**Contents**

1.1 About the System

1.1.1 Project overview…………………………………………………  
1.1.2 DHCP……………………………………………………………………..  
1.1.3 DNS……………………………………………………………………..  
1.1.4 HTTP APP………………………………………………..  
1.1.5 RUDP  
1.1.6 Diagram of our system ………………………….  
1.1.7 How to run the program

1.2 DHCP

1.2.1 Code Description……………………………………………………..…………  
1.2.2 Flowchart  
1.2.4 Wireshark………………………………………………………

1.3 DNS

1.3.1 Code Description……………………………………………………..…………  
1.3.2 Flowchart……………………………………………………………………  
1.3.5 Wireshark…………………………………………………

1.4 HTTP APP

1.4.1 Code Description……………………………………………………..…………  
1.4.2 Flowchart……………………………………………………………..  
1.4.4 Wireshark……………………………………………………………………….

1.5 Client

1.5.1 Code Description……………………………………………………..…………  
1.5.2 Flowchart……………………………………………………………..  
1.5.4 Wireshark……………………………………………………………………….

1.6 Question

1.7 Bibliography………………………………………………………………………..……………..

**1.1 About the System**

1.1.1 Project overview

My project focuses on three servers and a client, with the servers being DNS, DHCP, and HTTP application servers. The DNS server will be responsible for translating domain names into IP addresses, while the DHCP server will be in charge of assigning IP addresses dynamically to the client devices. The HTTP application server will host a web application that the client can access through a web browser.

Additionally, the HTTP application server will have the role of performing redirects to another server and downloading files from there. The client device will be configured to interact with all three servers, ensuring seamless communication between them. The project aims to showcase how these servers can work together to provide efficient network services to client devices.

1.1.2 DHCP

What is that DHCP?

A Dynamic Host Configuration Protocol (DHCP) server is a network server that automatically assigns IP addresses and other network configuration parameters to devices on a network. DHCP simplifies network administration by allowing devices to obtain network settings automatically instead of requiring manual configuration.

When a device, such as a computer or a smartphone, connects to a network, it sends a request to the DHCP server for an IP address. The DHCP server responds by assigning a unique IP address to the device, along with other network configuration information such as subnet mask, default gateway, and DNS server addresses.

DHCP servers are commonly used in local area networks (LANs) and are particularly useful in large organizations where manual IP address assignment would be time-consuming and error-prone. DHCP servers can also be configured to assign specific IP addresses to specific devices, which is useful for devices that require a static IP address, such as servers or network printers.

1.1.3 DNS

The Domain Name System (DNS) is a hierarchical distributed naming system for computers, services, or other resources connected to the Internet or a private network. DNS translates human-readable domain names, such as "google.com", into IP addresses that computers use to identify each other on the network.

When a user types a domain name into a web browser, the browser sends a request to a DNS resolver, which is typically provided by the user's Internet service provider (ISP). The resolver then queries one or more DNS servers to obtain the IP address associated with the domain name. Once the IP address is obtained, the browser can then connect to the web server associated with that IP address to retrieve the requested web page.

DNS is critical to the functioning of the Internet as it enables users to access websites and other online services using memorable domain names instead of having to remember the numerical IP addresses associated with those services. DNS is also used for other purposes such as email routing, network address translation (NAT), and other network services.

1.1.4 HTTP APP

HTTP stands for Hypertext Transfer Protocol, which is the primary protocol used for transferring data over the World Wide Web. An HTTP application, also known as a web application, is a software application that uses HTTP as its primary communication protocol to interact with users and other software applications over the internet.

HTTP applications are typically accessed using a web browser, which sends HTTP requests to the web server hosting the application. The server then responds to the requests by sending back HTTP responses, which may include HTML, CSS, JavaScript, images, or other types of data.

Web applications can perform a wide range of tasks, from simple tasks such as displaying web pages or handling user authentication to complex tasks such as online shopping, social networking, and collaborative document editing. Popular examples of web applications include online email services, social media platforms, online banking, and e-commerce websites.

HTTP applications can be built using a variety of programming languages, frameworks, and tools, depending on the specific requirements of the application. Some popular web development frameworks include Ruby on Rails, Django, and Node.js, while popular web development tools include HTML, CSS, JavaScript, and jQuery.

In our case, when a client, such as a web browser, sends a request to a server using the HTTP protocol, the server can respond with an HTTP redirect response code (such as 301 or 302) to indicate that the requested resource has been moved permanently or temporarily to a different location.

In the case of a file download, this could mean that the file is no longer hosted on the original server, but has been moved to a different server or location. The redirect response will include a new URL for the client to request the resource from the new location.

Once the client receives the redirect response, it will typically send a new request to the new URL, and the server at that location will respond with the requested file for download.

This process of redirecting to a new server or location can be useful in a variety of scenarios, such as when a website is moved to a new domain or when large files are hosted on a separate file server for better performance.

1.1.4 RUDP

RUDP (Reliable User Datagram Protocol) is a transport layer protocol designed to provide a reliable, connection-oriented service over unreliable, connectionless transport protocols such as UDP (User Datagram Protocol).

The main purpose of RUDP is to provide a reliable data transfer service to applications that require guaranteed delivery of data. It does this by adding reliability and flow control mechanisms to UDP.

RUDP provides reliable delivery of data by implementing mechanisms such as retransmission of lost or damaged packets, packet sequencing, and acknowledgment of received packets. RUDP also provides flow control to prevent the receiver from being overwhelmed by the sender's data.

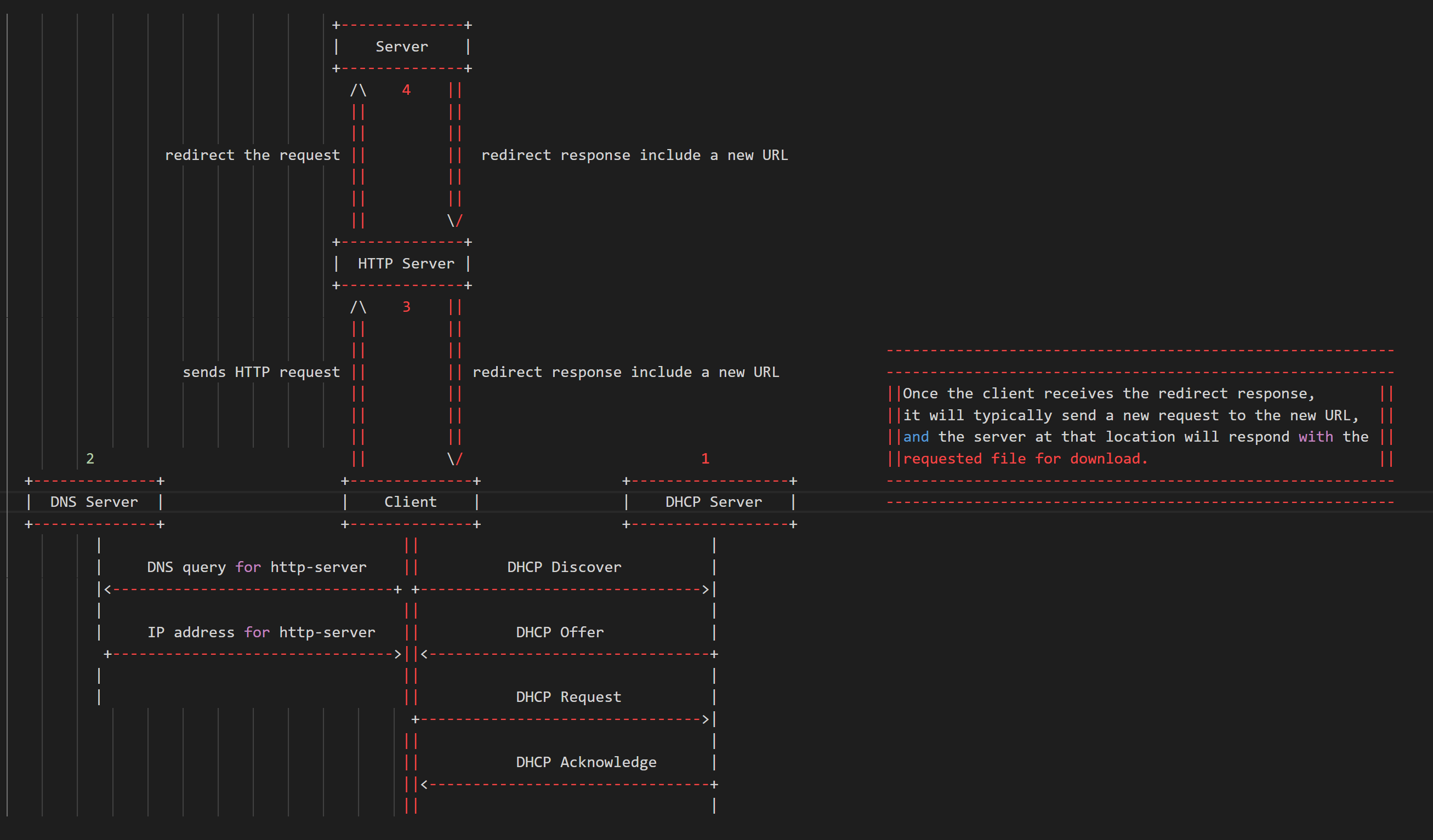
In addition to reliability and flow control, RUDP also supports congestion control. Congestion control mechanisms are used to avoid network congestion by controlling the rate at which data is transmitted. This helps to prevent network congestion, which can result in dropped packets and degraded performance.

RUDP has some advantages over other reliable transport protocols such as TCP (Transmission Control Protocol). One advantage is that RUDP is less complex than TCP, which makes it more suitable for use in embedded systems and other environments with limited resources. RUDP also has lower latency and overhead than TCP, which can be beneficial in applications that require real-time or low-latency data transfer.

However, RUDP also has some disadvantages compared to TCP. For example, RUDP does not provide congestion control as effectively as TCP, which can lead to network congestion and performance issues. Additionally, RUDP is not widely used and may not be supported by all network devices and applications.

Overall, RUDP is a reliable, connection-oriented protocol that can be a good alternative to TCP in certain situations. It provides mechanisms for reliable data transfer, flow control, and congestion control, while also being less complex and having lower latency than TCP. However, its effectiveness in preventing network congestion and its limited support in some environments should be taken into a count when considering its use.

1.1.6 Diagram of our system



The flow of the system is as follows:

1. The client sends a DHCP request to the DHCP server to obtain an IP address.

2. The DHCP server assigns an IP address to the client.

3. The client sends a DNS query to the DNS server to resolve the hostname of the HTTP server.

4. The DNS server responds to the client with the IP address of the HTTP server.

5. The client sends an HTTP request to the HTTP server to download a file.

6. The HTTP server responds to the client with the file.

7. The client receives the file.

**1.2 DHCP**

1.2.1 Code Description

Before we will jump into the code, we want to explain about D.O.R.A.

DORA is an acronym for the four-step process used by DHCP to assign IP addresses to devices on a network.   
Here are the steps involved:

**1. Discover:** The client device sends out a broadcast message on the network, requesting an IP address. The broadcast message includes the client's MAC address and other information about the client.

**2. Offer:** The DHCP server receives the broadcast message and responds with an IP address offer. The server sends a broadcast message back to the client, including an available IP address and other configuration information, such as the subnet mask, DNS servers, and default gateway.

**3. Request:** The client receives the IP address offer from the DHCP server and decides whether to accept it. If the client accepts the offer, it sends a broadcast message to the server, indicating that it wants to use the offered IP address.

**4. Acknowledge:** The DHCP server receives the broadcast message from the client and sends a final message to the client, acknowledging that the IP address has been assigned to the client. This message includes the lease time for the IP address, which indicates how long the client can use the address before it needs to renew the lease.

Overall, the DORA process ensures that IP addresses are assigned in a coordinated and efficient manner, minimizing the risk of IP address conflicts, and making it easier for network administrators to manage network resources.

DHCP server:

**1.** **Import necessary libraries:**

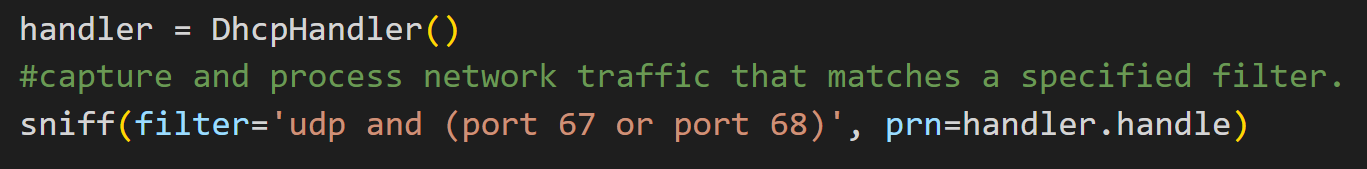
The first step of the code is to import the necessary libraries, namely **scapy.all**, **scapy.layers.dhcp**, **scapy.layers.inet**, and **scapy.layers.l2**. These libraries are used throughout the code to handle and manipulate network packets.

Text

Description automatically generated

**2. Instantiate a DhcpHandler object & Use scapy to sniff network traffic:**

The next step is to create an instance of the DhcpHandler class. And then we will use scapy's **sniff** function to capture and process network traffic that matches a specified filter. The **sniff** function takes two arguments: a filter expression and a callback function (handler.handle in this case). The filter expression is used to specify the types of packets that should be captured and processed by the handle method. In this case, the filter expression specifies that UDP packets with a source or destination port of 67 or 68 should be captured and processed.



**3. Initialize instance variables:**

The **DhcpHandler** class has an **\_\_init\_\_** method that initializes two instance variables. The **ip\_pool** variable is a list of IP addresses that can be assigned to clients. The **ip\_assignments** variable is a dictionary that maps assigned IP addresses to the MAC addresses of the clients that have been assigned those addresses.

Text

Description automatically generated

**4. Define a method to get the next available IP address:**

The **DhcpHandler** class has a **get\_next\_available\_ip** method that returns the next available IP address from the **ip\_pool** list. If there are no more available IP addresses, the method returns None.

Text

Description automatically generated

**5. Define a method to handle DHCP packets:**

The **DhcpHandler** class has a **handle** method that is responsible for handling DHCP packets. The method takes a packet object as an argument and checks whether the packet is a DHCP Discover or a DHCP Request.

If the packet is a DHCP Discover, the method extracts the client's MAC address from the packet, assigns the next available IP address to the client, constructs a DHCP offer packet, and sends the offer packet to the client.

Text

Description automatically generated

If the packet is a DHCP Request, the method extracts the client's MAC address and requested IP address from the packet, assigns the requested IP address to the client, constructs a DHCP ACK packet, and sends the ACK packet to the client.

Text

Description automatically generated

1.2.2 Flowchart

Here's a simple flowchart that outlines the basic steps involved in a DHCP lookup process between a DHCP client and server:

**1.** Get MAC address of client interface

**2.** Send DHCP Discover packet to broadcast address

**3.** Wait for DHCP Offer packet from server

**4.** If Offer received:

a. Send DHCP Request packet with offered IP address and server ID

b. Wait for DHCP Acknowledgement packet from server

**5.** If Acknowledgement received:

a. Configure network interface with assigned IP address and lease time

b. End

Here's a simple diagram that illustrate the communication between a DHCP server and a DHCP client:

A picture containing text

Description automatically generated

In this diagram, the DHCP client sends a DHCP Discover message to the broadcast address. The DHCP server receives the message and responds with a DHCP Offer message. The client then sends a DHCP Request message requesting the offered IP address, and the server sends a DHCP Acknowledge message to confirm that the IP address has been assigned to the client.

1.2.3 Wireshark

**A picture containing text

Description automatically generated**

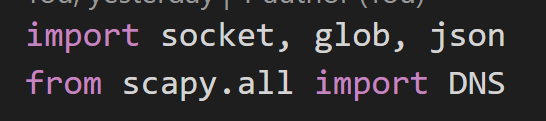
**1.3 DNS**

1.3.1 Code Description

The DNS server receives DNS requests from clients, looks up the requested domain name in its DNS zones, and returns the corresponding IP address to the client. The code is divided into several functions, each of which performs a specific task.

Here's a step-by-step explanation of the code:

**1.** The first line of the code imports the required modules - socket, glob, and json - for socket programming, file handling, and JSON parsing, respectively. It also imports the DNS class from the Scapy module.



**2.** The next function, **get\_bit()**, is a utility function that takes an integer value and a bit index and returns the value of the bit at that index in the binary representation of the integer value.

Text

Description automatically generated

**3.** **The get\_flags\_by\_bit()** function is used to decode the DNS response flags from a two-byte value. The function takes a two-byte value containing the response flags as input and returns the decoded response flags as a two-byte value.

Text

Description automatically generated

4. **The get\_flags()** function is used to set the flags of a DNS response message. The function takes a byte string containing the input flags and returns a byte string containing the response flags.

Text

Description automatically generated

**5. The get\_question\_domain()** function takes a binary string representing a DNS query message as input and extracts the domain name and question type from it. The function returns a tuple containing the domain name (as a list of strings) and the question type (as a binary string).

Text

Description automatically generated

Text

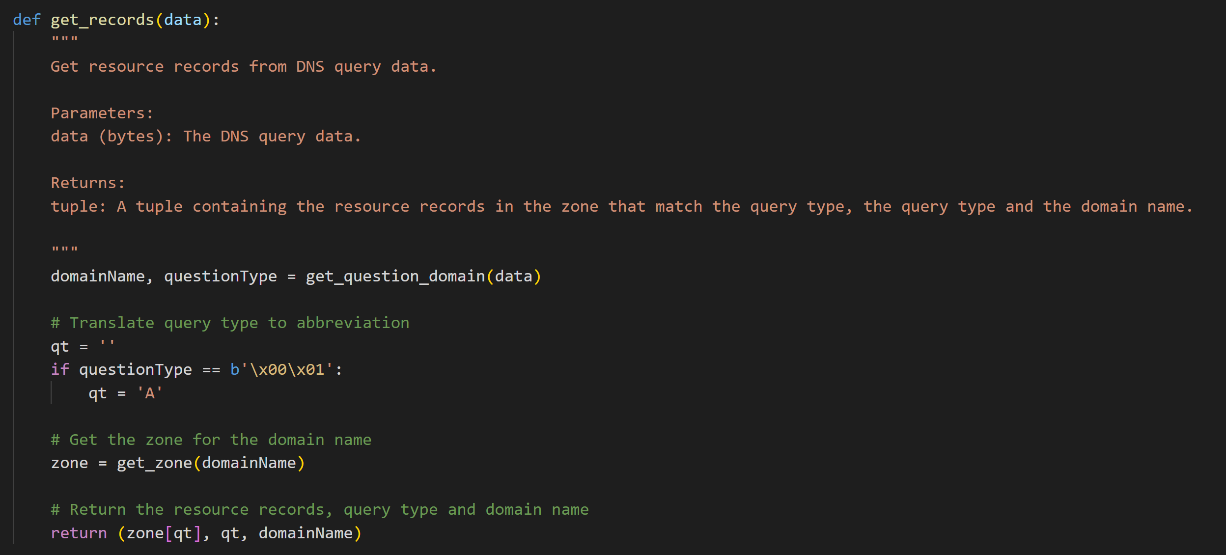
Description automatically generated**6. The load\_zone()** function loads DNS zones from JSON files and returns a dictionary with zone names as keys and zone data as values. The function iterates over all zone files in the zones directory and loads the data from each file into a dictionary. The "$origin" key in each zone file specifies the zone name.

**7. The get\_zone()** function retrieves the zone data for a given domain name by concatenating the domain name and querying the global zoneData dictionary.

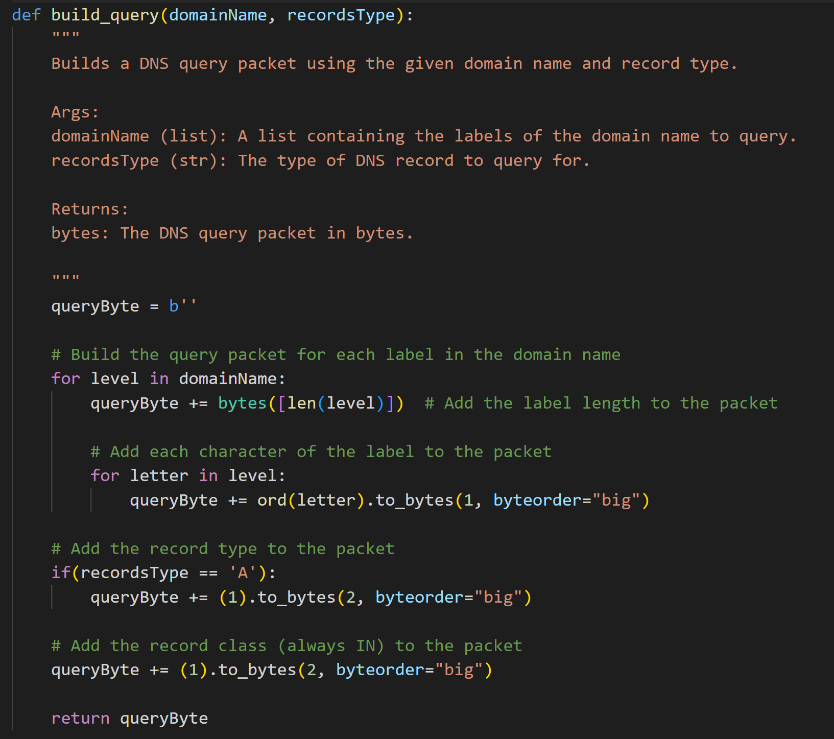
Text

Description automatically generated

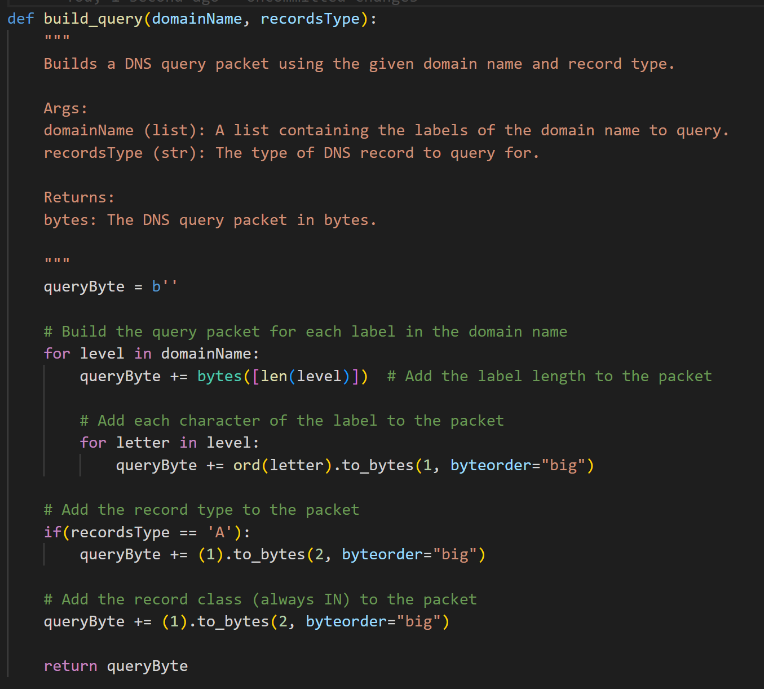
**8. The get\_records()** function takes DNS query data as input and returns resource records (RRs) for the queried domain name. The function extracts the domain name and question type from the query data using the get\_question\_domain() function and retrieves the zone data for the domain name using the get\_zone() function. The function then checks the type of the query (A or AAAA) and returns the corresponding RRs.



**9. build\_query()** : builds a DNS query packet using the given domain name and record type. The function takes two arguments: a list containing the labels of the domain name to query (domainName) and the type of DNS record to query for (recordsType). The function starts by initializing an empty byte string called queryByte, which will be used to build the DNS query packet. It then iterates over each label in the domainName list and adds the label length to the packet followed by each character of the label. After all labels have been added to the packet, the function adds the record type to the packet (if it is an 'A' record) and the record class (which is always 'IN') to the packet. And then return the DNS query packet as a bytes object.



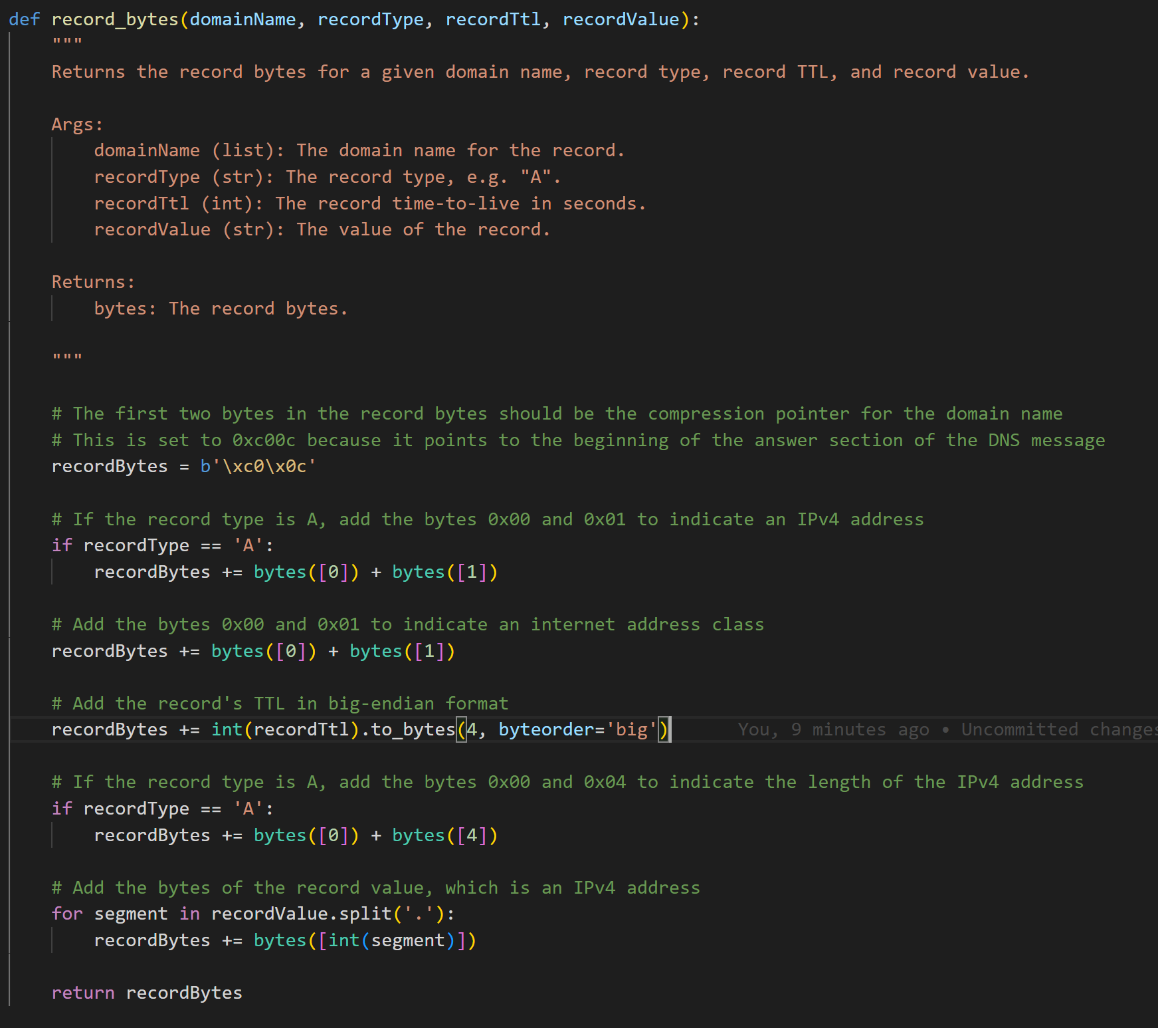
**10. build\_query()** builds a DNS query packet using the given domain name and record type. It takes in two arguments, domainName and recordsType, where domainName is a list containing the labels of the domain name to query and recordsType is the type of DNS record to query for. The function creates an empty byte string called queryByte, and then proceeds to build the query packet for each label in the domainName list. For each label, it adds the length of the label as a single byte, and then adds each character of the label as a single byte. Next, the function checks if the recordsType is 'A', and if it is, it adds the record type to the packet as a two-byte sequence with a value of 1. Finally, the function adds the record class (always IN) to the packet as a two-byte sequence with a value of 1. The resulting DNS query packet is returned as a bytes object.



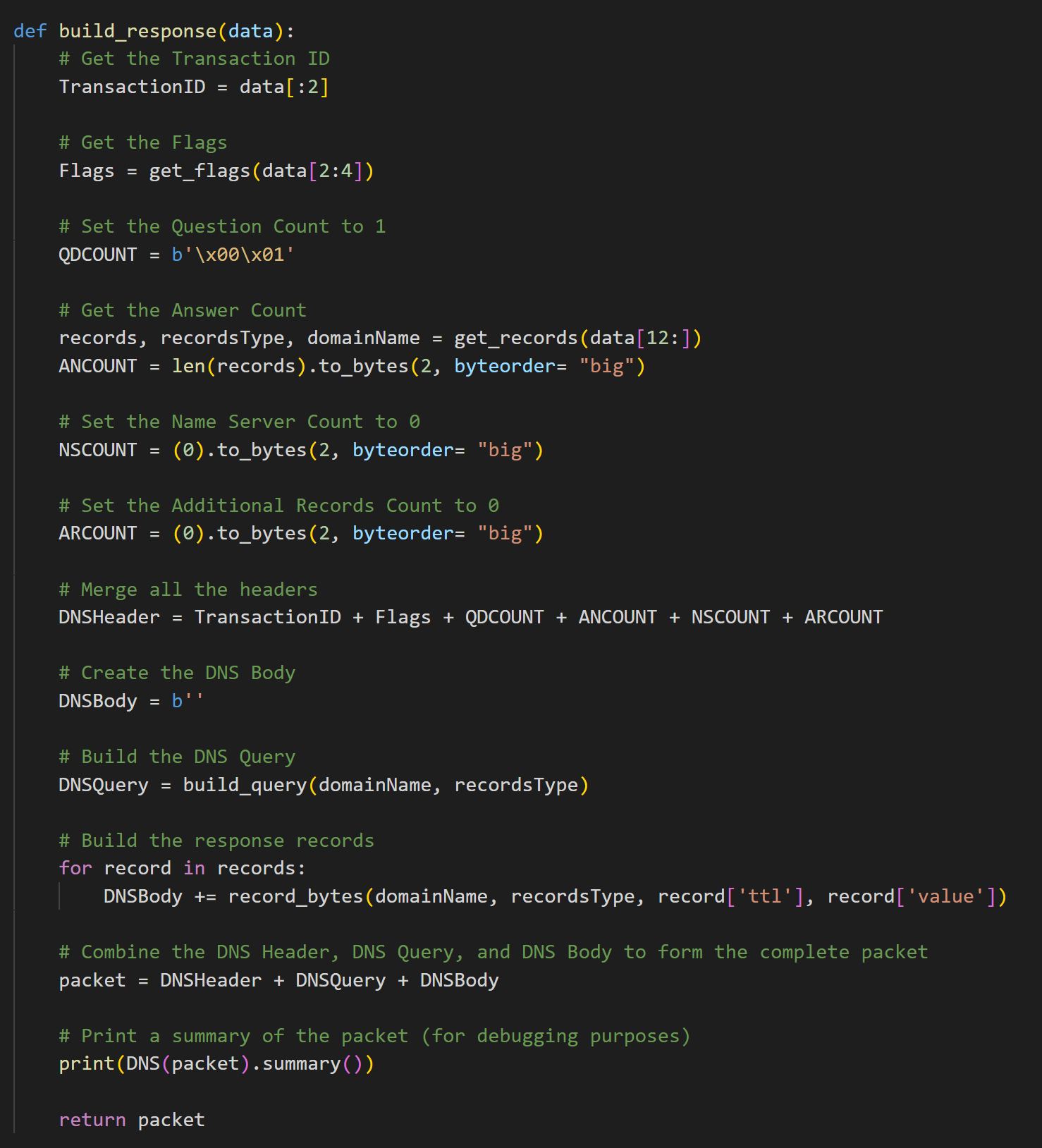
**11.** The **record\_bytes()** function takes in a domain name, record type, record TTL, and record value, and returns the record bytes for that record. Here's how it works:

* The function starts by setting the compression pointer to 0xc00c, which points to the beginning of the answer section of the DNS message.
* If the record type is 'A', it adds the bytes 0x00 and 0x01 to indicate an IPv4 address.
* The function then adds the bytes 0x00 and 0x01 to indicate an internet address class.
* Next, it adds the record's TTL in big-endian format, which is a 4-byte integer.
* If the record type is 'A', it adds the bytes 0x00 and 0x04 to indicate the length of the IPv4 address.
* Finally, the function adds the bytes of the record value, which is an IPv4 address, by iterating over each segment of the IP address and adding the byte representation of that segment.

The function returns the resulting record bytes.



**12.** The **build\_response()** function takes in a DNS query packet data and constructs a DNS response packet.   
It does this by first extracting relevant fields from the query packet such as the Transaction ID, Flags, Question Count, Answer Count, and Additional Records Count. It then sets the Question Count to 1 and the Name Server Count and Additional Records Count to 0.  
Next, it constructs the DNS header by merging all the fields together.  
The DNS body is then built by first constructing the DNS query using the build\_query function and then constructing the response records using the record\_bytes function.  
Finally, the DNS header, DNS query, and DNS body are merged together to form the complete DNS response packet which is returned.  
The function also prints a summary of the packet for debugging purposes.



The main part of the code creates a UDP socket and binds it to a port (53) to listen for incoming DNS requests from clients. The code enters an infinite loop where it waits for incoming DNS requests, processes them using the get\_records() function, and sends the response back to the client. The code also handles exceptions and errors that may occur during socket communication.

Text

Description automatically generated

1.3.2 Flowchart

Here's a simple flowchart that outlines the basic steps involved in a DNS lookup process between a DNS client and server:

**1.** DNS client sends a query for a domain name to the local DNS resolver.

**2.** If the local resolver has the IP address for the domain name in its cache, it returns the IP address to the client.

**3.** If the local resolver does not have the IP address in its cache, it sends the query to the root DNS server.

**4.** The root server responds with the IP address of the top-level domain server (such as .com or .org).

**5.** The local resolver sends the query to the top-level domain server.

**6.** The top-level domain server responds with the IP address of the authoritative DNS server for the domain name (such as ns1.example.com).

**7.** The local resolver sends the query to the authoritative DNS server.

**8.** The authoritative DNS server responds with the IP address for the domain name.

**9.** The local resolver caches the IP address and returns it to the client.

Here's a simple diagram that illustrate the communication between a DNS server and a DNS client:

A picture containing text

Description automatically generated

In this diagram, the DNS client sends a query to the DNS server using User Datagram Protocol (UDP) on port 53. The DNS server receives the query and processes it, then sends a response back to the client on UDP port 53. The client receives the response and uses the information provided to complete its request.

Note that in some cases, a DNS server may also communicate with other DNS servers to resolve a query. This communication typically occurs over Transmission Control Protocol (TCP) rather than UDP and may involve multiple rounds of querying and response exchanges. However, for the purposes of this diagram, we have simplified the communication to focus on the basic interaction between a DNS client and a DNS server.

1.3.3 Wireshark

**1.4 HTTP APP**

1.4.1 Code Description

1.4.2 Flowchart

1.4.3 Wireshark

**1.5 Client**

1.5.1 Code Description

1.5.2 Flowchart

1.5.3 Wireshark

**1.6 Question**

**1.** List at least four major differences between the TCP and QUIC protocols

TCP (Transmission Control Protocol) and QUIC (Quick UDP Internet Connections) are two transport layer protocols used to transfer data over the internet. Here are four major differences between these two protocols:

**1.** Reliability: TCP is a reliable protocol, which means it guarantees the delivery of data and detects lost packets by retransmitting them. QUIC is also reliable, but it uses a different mechanism to achieve reliability. QUIC uses a combination of forward error correction and retransmissions to ensure the delivery of data.

**2.** Connection setup: TCP requires a three-way handshake to establish a connection between the client and server. This process involves a series of messages between the client and server before data transfer can begin. QUIC, on the other hand, uses a single round-trip to establish a connection. This makes QUIC faster than TCP for establishing connections.

**3.** Congestion control: TCP uses a congestion control algorithm that adjusts the flow of data based on the network conditions. QUIC uses a similar congestion control algorithm, but it is more aggressive and can adapt to changing network conditions more quickly than TCP.

**4.** Packet loss recovery: When a packet is lost during transmission, TCP uses a retransmission mechanism to resend the lost packet. QUIC, on the other hand, uses a more efficient mechanism that allows it to recover from multiple lost packets at once, reducing latency and improving performance. This mechanism is called "retransmission with loss detection."

**2.** List at least two main differences between Cubic and Vegas

Cubic and Vegas are two different congestion control algorithms used in TCP. Here are two main differences between Cubic and Vegas:

**1.** Approach: Cubic and Vegas use different approaches to control congestion. Cubic uses a window-based approach that increases the congestion window size slowly at the beginning and then ramps up quickly to utilize available bandwidth. Vegas, on the other hand, uses a delay-based approach that measures the round-trip time (RTT) of packets and adjusts the sending rate based on the observed delay.

**2.** Responsiveness: Cubic and Vegas differ in their responsiveness to changes in the network conditions. Cubic is less responsive than Vegas and takes longer to react to changes in the available bandwidth. This makes Cubic more suitable for long-lived flows with steady-state traffic. Vegas, on the other hand, is more responsive and can quickly adapt to changes in network conditions, making it more suitable for short-lived flows and bursty traffic.

3. Explain what the BGP protocol is, how it differs from OSPF and does it work according to short routes.

BGP (Border Gateway Protocol) is a routing protocol used to exchange routing information between different networks on the Internet. It is primarily used by Internet Service Providers (ISPs) to route traffic between different autonomous systems (AS).

Unlike OSPF (Open Shortest Path First), which is an Interior Gateway Protocol (IGP) used within a single autonomous system (AS), BGP is an Exterior Gateway Protocol (EGP) used to exchange routing information between different autonomous systems.

One of the key differences between BGP and OSPF is the way they calculate routes. OSPF uses the shortest path algorithm to calculate the shortest path between two routers within the same AS, whereas BGP uses a policy-based routing approach to determine the best path for traffic to take between different ASes. BGP allows network administrators to define policies that influence the path selection process, based on factors such as AS path length, network performance, and cost.

BGP does not necessarily always work according to the shortest routes. The selection of the best path in BGP is based on various factors, including AS path length, local preference, MED (Multi-Exit Discriminator), and other policy-based attributes. Therefore, BGP may choose a longer path if it meets certain policy criteria, such as avoiding a particular network or preferring a certain type of connection. The primary goal of BGP is to select the best path according to the policies defined by the network administrator.

4. Given the code you developed in this project, please add the data to this table based on your project's message process. Explain how the messages will change if there is a NAT between the user and the servers and whether you will use the QUIC protocol.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Application** | **Port Src** | **Port Des** | **Ip Src** | **Ip Des** | **Mac Src** | **Mac Des** |
| HTTP |  |  |  |  |  |  |

Assuming the table you refer to is related to a network communication system, the message process may involve several steps, such as establishing a connection, exchanging data, and terminating the connection. Each step may involve different types of messages, such as SYN, SYN-ACK, ACK, data packets, FIN, etc.

If there is a NAT (Network Address Translation) between the user and the servers, the messages will need to be translated to accommodate the NAT. For example, the source IP address in the packets sent by the user will be replaced by the public IP address of the NAT device, and the port numbers may be modified to avoid conflicts with other connections. The NAT device will maintain a mapping table to keep track of the translations.

Whether to use the QUIC protocol will depend on several factors, such as the network conditions, the type of data being transferred, and the security requirements. QUIC is designed to provide faster and more secure transport than TCP, especially in high-latency and lossy networks. However, if there are strict firewall policies or other constraints that limit the use of QUIC, TCP may be a better option.

5. Explain the differences between the ARP protocol and DNS

ARP (Address Resolution Protocol) and DNS (Domain Name System) are two protocols used in computer networks for different purposes. Here are the main differences between ARP and DNS:

**1.** Purpose: ARP is used to map a physical (MAC) address to an IP address, while DNS is used to map a domain name to an IP address.

**2.** Scope: ARP operates at the data link layer, which means it is used to resolve addresses within a local network segment. DNS operates at the application layer, which means it can be used to resolve addresses across different networks and the internet.

**3.** Functionality: ARP is a simple protocol that sends a broadcast message to the network to request the MAC address of a specific IP address. Once the MAC address is obtained, it is cached in the system's ARP cache for future use. DNS, on the other hand, is a more complex protocol that involves a series of recursive queries to different DNS servers to resolve a domain name to an IP address.

**4.** Implementation: ARP is implemented as a part of the network interface driver in most operating systems and requires no additional software. DNS requires a DNS resolver to be installed on the system, which communicates with one or more DNS servers to resolve domain names.

In summary, ARP is used to map physical addresses to IP addresses within a local network segment, while DNS is used to map domain names to IP addresses across different networks and the internet. ARP operates at the data link layer, while DNS operates at the application layer, and DNS is a more complex protocol that involves recursive queries to multiple servers, while ARP is a simple protocol that sends a broadcast message to obtain the MAC address of a specific IP address.

**1.7 Bibliography**

DNS server :

<https://www.ietf.org/rfc/rfc1035.txt>

<https://www.youtube.com/@howCode>

DHCP server:

<https://www.thepythoncode.com/article/dhcp-listener-using-scapy-in-python>

<https://www.youtube.com/watch?v=e6-TaH5bkjo>

<https://he.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol>

<https://wiki.wireshark.org/DHCP>

HTTP server:

<https://requests.readthedocs.io/en/latest/>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections>

<https://stackoverflow.com/questions/27241804/sending-a-file-over-tcp-sockets-in-python>