Communication Networks Project

HTTP Application

ID's:

211696521

328596978

Table of Contents:

Code Explanations

DHCPserver

This is DHCP server, which dynamically assigns IP addresses to network clients

The script first sets up a UDP socket to listen for DHCP messages on port 67. It then defines two IP addresses, which represent the start and end of the IP address range that the DHCP server can assign to clients.

Next, there are two functions defined:

is_ip_available:

takes an IP address as input and checks if it is available by sending an ICMP echo request to it. If there is a response, the IP address is not available. Otherwise, the IP address is available.

generate_dhcp_offer takes an IP address as input and generates a DHCP offer message that can be sent to a client to offer the IP address for use.

The script then enters an infinite loop to listen for incoming DHCP messages. When a message is received, it first checks if the message is long enough to contain the necessary DHCP fields. If not, it skips processing the message.

If the message is a DHCP discover message (type 1), the script tries to find an available IP address in the range specified earlier. If an available IP address is found, it generates a DHCP offer message with the available IP address and sends it to the client.

If the message is a DHCP request message (type 3), the script checks if the requested IP address is available. If it is not available, it sends a DHCP negative acknowledgement message to the client. Otherwise, it generates a DHCP acknowledgement message with the assigned IP address and sends it to the client.

If the message is not a DHCP discover or request message, the script simply skips processing the message.

For each processed message, the script prints a message to the console indicating the action taken

Client

This is a a DHCP and DNS client.
It uses a socket to send and receive DHCP messages to obtain an IP address from a DHCP server. Once the IP address is obtained, the script sends a DNS query to a DNS server using the obtained IP address as the source IP address.

The script works by sending a DHCP discover message to the broadcast address and waiting for a DHCP offer message from a DHCP server. If a DHCP offer is received, the script sends a DHCP request message to accept the offered IP address and waits for a DHCP acknowledge message from the server. Once the IP address is obtained, the script sends a DNS query to a DNS server using the obtained IP address as the source IP address.

The script has the following main components:

*A function to generate a random transaction ID for DHCP messages.
*Functions to check if a DHCP message is an offer or an acknowledgement and parse the message to extract relevant information.
*Functions to generate DHCP messages (discover, request).
*A main loop that sends DHCP messages and processes responses until an IP address is obtained.
*A DNS resolver that sends a DNS query to a DNS server using the obtained IP address as the source IP address.
The script assumes that a DHCP server is running on the same network as the client and that a DNS server is running on the local machine at IP address 127.0.0.1 on port 533. The DHCP client runs in an infinite loop, sending DHCP messages and waiting for responses until it obtains an IP address. Once an IP address is obtained, the script sends a DNS query to the DNS server using the obtained IP address as the source IP address.

DNSserver

This is a DNS server that listens for incoming DNS queries and updates, and responds to queries with IP addresses for known domain names. If a domain name is not in the cache, the server sends a query to a next-level DNS server to resolve it. The DHCP server updates the DNS cache with the IP address for a client when it assigns a new lease.

The code uses the socket and threading modules for networking and multi-threading, and the dns module for parsing and constructing DNS messages.

The DNS cache is implemented as a Python dictionary that maps domain names to IP addresses. The cache is initialized with two entries for google.com and yahoo.com.

The code defines several functions:

handle_dns_query: Handles an incoming DNS query message. If the domain name is in the DNS cache, it creates a DNS response message with the IP address for the domain name and sends it back to the client. If the domain name is not in the cache, it sends a DNS query to a next-level DNS server. If the domain name does not exist, it returns a DNS response message with the appropriate error code.

handle_dns_update: Handles an incoming DNS update message. It parses the message and updates the DNS cache with the domain name and IP address.

listen_for_dns_queries: Listens for incoming DNS queries on port 533 and passes them to handle_dns_query.

listen_for_dns_updates: Listens for incoming DNS updates on port 9898 and passes them to handle_dns_update.

update_dns_cache: Updates the DNS cache with a new IP address for a client's hostname.

handle_lease_update: Handles an incoming DHCP lease update message. It parses the message and updates the DNS cache with the client's hostname and IP address.

The code creates two threads for the DNS query and update listeners, and starts them with the start method. It then waits for both threads to complete with the join method.

TCPserver

This is a server-side TCP program that listens for incoming connections on a socket and then responds to incoming messages from clients.

The code begins by importing several libraries such as socket, requests, mimetypes, urlparse, and pathlib.

The mimetypes library is used to define additional file types and their corresponding MIME types. The urlparse library is used to parse the URL of the file that the client wants to download, and the pathlib library is used to determine whether the file already exists on the server.

The download_file function is defined to download a file from the specified URL and save it to the specified filename. It uses the requests library to download the file in chunks and save it to the specified filename.

The program then creates a socket object and binds it to a host and port. The listen function is called to start listening for incoming connections.

The program then enters an infinite loop to continuously listen for incoming connections. When a connection is received, the program accepts the connection and receives a message from the client containing the URL of the file to download and the desired filename to save it as.

The program then checks if the URL is valid and if the file already exists on the server. If the URL is invalid, the program returns an error message to the client. If the file already exists on the server, the program returns a message to the client indicating that the file already exists. If the file does not exist on the server, the program downloads the file and saves it to the specified filename.

Finally, the program sends a response message to the client containing the status of the download operation, whether successful or not, and closes the client socket.

Overall, this program is a simple server-side program that listens for incoming connections and downloads files from a specified URL, while handling errors and providing feedback to the client.

Client

This client creates a graphical user interface (GUI) for a simple HTTP downloader application that can use either TCP or RUDP (Reliable UDP) protocol to download files from a given URL.

The program uses the tkinter library to create the GUI and the socket library to establish a socket connection with the server. The GUI includes several input fields and a "Download" button that initiates the file download process.

The connect() function is responsible for initiating the connection with the server using the selected protocol and downloading the specified file. It gets the URL, protocol, and filename from the input fields and validates them before establishing the connection with the server using the selected protocol. If the connection is successful, the function sends a request message to the server containing the URL and filename, and then waits for a response. If the response is received successfully, the function updates the GUI with the response message.

The program starts by creating the GUI and adding several input fields and buttons. The root window is created with the title "HTTP Downloader". The input fields include a URL input field, a protocol selection field with options for TCP and RUDP, a filename input field, and a "Download" button. The response from the server is displayed in a label below the "Download" button.

The program then enters the main event loop using the root.mainloop() method, which waits for user input and responds to events such as button clicks. When the user clicks the "Download" button, the connect() function is called to initiate the file download process.

RUDPserver

This is a server-side RUDP program that listens for incoming connections on a socket and then responds to incoming messages from clients.

The server binds to a specified host and port using the socket module, and listens for incoming UDP packets. When a packet is received, it is decoded and split into two parts: the URL of the file to be downloaded, and the name of the file that it should be saved as.

The program then checks if the file already exists on the server, and if it does, it sends a message back to the client informing them that the file already exists. If the file does not exist on the server, it attempts to download the file from the given URL using the requests library. The content type of the response is used to guess the file extension, and the file is saved to the server with the given name and extension.

The server sends a response back to the client, informing them whether the download was successful or if an error occurred during the download.

For DNS & DHCP & client

How to run

To run the DNS server, DHCP server, and client, you will need to open three separate terminal windows or tabs. In each terminal, navigate to the directory where the scripts are saved.
In the first terminal, run the DNS server by executing the command:
"python3 DNS_Server.py".
In the second terminal, run the DHCP server by executing the command:
"python3 DHCP_Server.py".
In the third terminal, run the client by executing the command:
"python3 client.py".
The DNS and DHCP servers need to be running in order for the client to successfully obtain an IP address and connect to the internet.
Once the client is running, it will automatically request an IP address from the DHCP server and update the DNS server with its hostname and IP address.
The client will then be able to access the internet through the DNS server.

FOR TCPserver & RUDPserver & Client

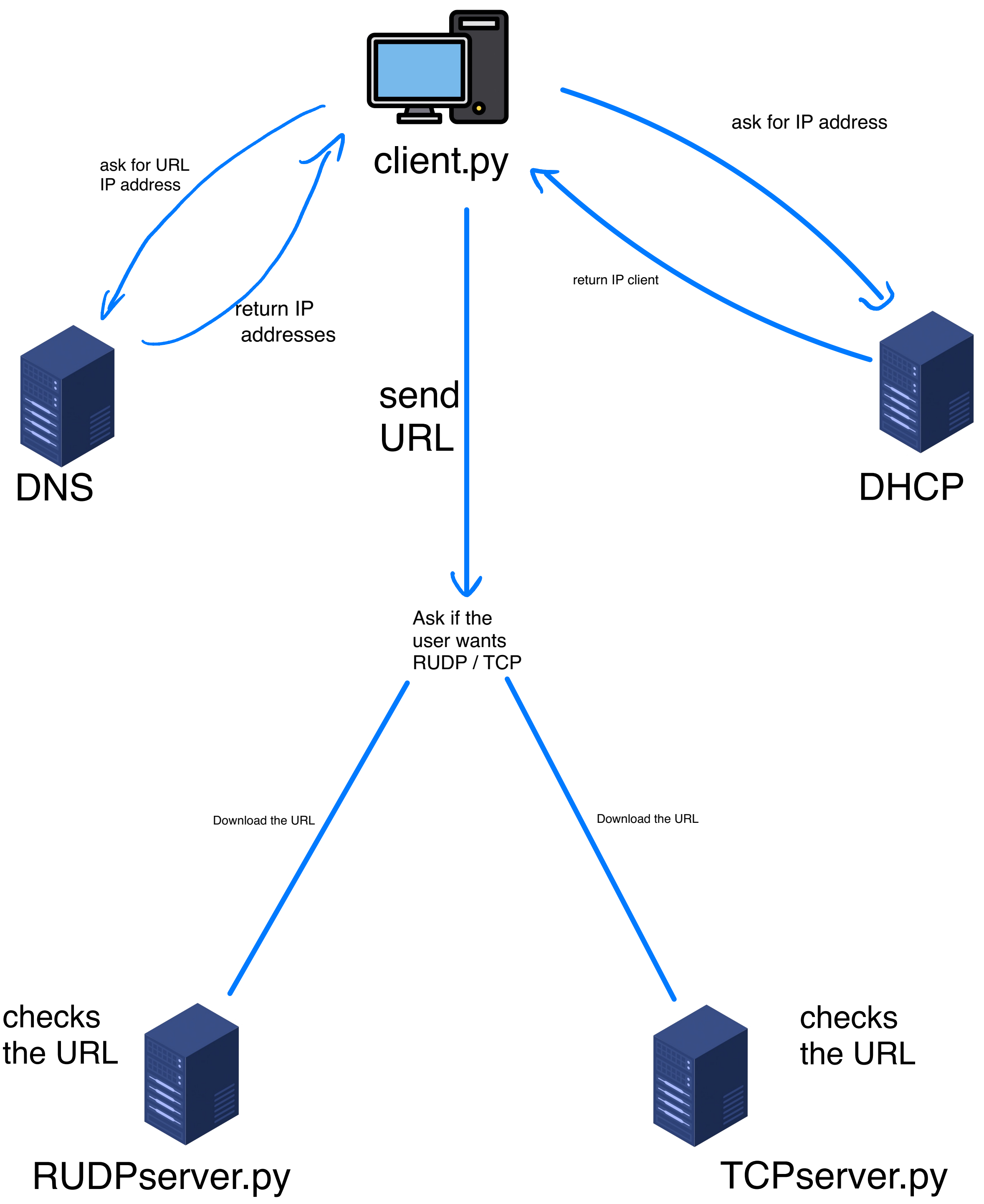
How to run

To run the HTTP program, you will need to open three separate terminals.
In each terminal, navigate to the directory where the scripts are saved.
In the first terminal, run the TCP server by executing the command:
"python3 TCPserver.py".
In the second terminal, run the RUDP server by executing the command:
"python3 RUDPserver.py".
In the third terminal, run the client by executing the command:
"python3 Client.py".
The RUDP and TCP servers need to be running befor the client.
The GUI for the client will open. Enter the URL of the file you want to download, select the protocol you want to use (TCP or RUDP), and enter the desired filename.

Click the "Download" button to initiate the download.

The response label in the GUI will display the status of the download (success or error message).
The downloaded file will be saved in the same folder as the Python files.

State Diagram:



Important NOTE:
We draw the ideal idea of our project , our project works without DNS and DHCP server, when the client sends a URL to one of two servers(RUDPserver, TCPserver) server recognizes the IP of the client(prints the ip to the console) and checks URL if it correct.
We do not use DNS and DHCP because have trouble in their connection to the client and servers.

Q1: Write at least 4 main differences between TCP and QUIC protocols.

Answer:

1) **Connection Setup-** TCP requires 3 way handshake to establish a connection ,while QUIC requires 1 way handshake.

2) **Compatibility-** TCP is more often used protocol in most operating systems ,while QUIC is newer protocol so it not widely supported by all operating systems.

3) **Transport Layer-** QUIC operates at the application layer and transport layer ,while TCP use only the transport layer.

4) **Latency-** QUIC is designed to reduce latency by reducing the number of round trips required to establish a connection.

5) **Error Correction-** TCP has a detailed error correction mechanism ,including retransmission of lost packets and congestion control ,while QUIC has simpler error correction mechanism that is built on top of UDP.

Q2: Write at least 2 main differences between Cubic and Vegas.

Answer:

1) **Performance-** on high speed networks Cubic is better suited, because it is designed to handle larger bandwidth delay products than Vegas.

2) **Window Size Calculation-** Vegas use a linear increase in window size ,while Cubic uses a cubic increase.

Q3: Explain BGP protocol, how is it different from OSPF and does it work according to short paths?

Answer:

BGP- Border Gateway Protocol.
BGP is a routing protocol used exchange routing information between autonomous systems.
It is designed to allow routes in different autonomous systems to communicate and exchange information about the best path for forwarding traffic.

Difference- OSPF is a link state protocol that used to distribute routing information within a single autonomous system. It designed to find the shortest path between nodes within a network ,while BGP is designed to find the best path between networks.

Does BGP work according to the shortest path? - NO, it selects the best path based on a combination of path attributes.

Q4: Fill the table.

How the table data will change if will be a NAT between a client to the servers ?

Will you use QUIC protocol ?

Answer:

Application:
Port Src:
Port Des:
IP Src:
IP Des:
Mac Src:
Mac Des:

Q5: Explain the differences between ARP and DNS protocols.

Answer:

DNS = Domain Name System.
ARP = Address Resolution Protocol.

ARP is used to resolve a network layer address to a link layer address within a single network segment ,while DNS is used to resolve domain names to IP addresses across multiple networks.
ARP operates at the data link layer ,while DNS operates at application layer.

Conclusion:

RUDPserver and TCPserver codes that demonstrate different approaches to downloading file from the internet and communicating between them and the client.
They also address common issues such as packet loss and latency by using reliable protocols like TCP and incorporating error checking and handling.