# DHCP Client-Server Project Report

**Jamison Stalter and Abraham Gomez**

## Overview

This project involves developing a console-based client-server application implementing a simplified DHCP (Dynamic Host Configuration Protocol) protocol using socket programming in the Python language. The DHCP protocol facilitates the automatic assignment of IP addresses to network devices.

## Project Components

### Server Implementation (`server.py`)

The server program listens to client requests using the UDP transport layer. Key functionalities include:

- **IP Address Management:** The server manages a pool of IP addresses (192.168.45.1 to 192.168.45.14), tracks assigned addresses, and maintains records with client information.
- **DHCP Protocol Steps:** The server follows the DHCP protocol steps: DISCOVER, OFFER, REQUEST, and ACKNOWLEDGE.
- **List Step:**
    - Implemented a for loop to iterate through the records array and display all elements stored in the record.
- **DISCOVER Step:**
    - Introduced a variable to check the availability of an IP address.
    - Determined a free IP address and sent it to the client.
- **REQUEST Step:**
    - The server either declined the request or acknowledged it.

    - If acknowledged, the server sent the MAC address, IP address, and a new timestamp.
- **RELEASE Step:**
    - Sent out a released IP address associated with a specific MAC address.
    - No action was taken if there was no corresponding record.
- **RENEW Step:**
    - Checked the timestamp from the older record and renewed it for the client.
- Client Communication: The server responds to client requests, handles renewals and releases, and maintains proper recordkeeping.

## Client Implementation (`client1.py`, `client2.py`)

The client program initiates DHCP requests and interacts with the server. Notable features include:

- **Automatic MAC Address Retrieval:** The client retrieves its MAC address automatically.
- **DHCP Protocol Messages:** The client sends DISCOVER messages and handles server responses (OFFER, ACKNOWLEDGE, DECLINE, etc.).
- **User Interface:** The client provides a simple menu for user interactions, including renewing, releasing, and quitting.
- **Error Handling:** Implement robust error handling for unexpected server responses or network issues. Display meaningful error messages to the user.
- **Logging:** Integrate logging to keep a record of important events and actions. Helpful for debugging and understanding the program's flow.
- **Modularization:** Break down the client code into functions or classes for readability and maintainability. Easier to extend or modify in the future.
- **Configurability:** Allow users to configure server IP, port, or other settings easily. Implement a configuration file or command-line arguments for customization.
- **Graceful Exit:** Implement a graceful exit mechanism to ensure resources are released appropriately. Provide a clear message to the user when quitting.
- **Input Validation:** Validate user input to prevent unexpected errors or misuse of the program. Ensure the user enters valid choices and handle incorrect inputs gracefully.

- **Display More Information:** Display additional information about the client's status, such as the current IP address, expiration time, etc. Keep the user informed about the ongoing DHCP process.
- **Enhance Renewal and Release:** Improve renewal and release processes to handle various scenarios. Check if the client has an active lease before attempting to renew or release.
- **Interactive Prompts:** Make prompts more interactive by providing context-specific options based on the current state of the client.

## Admin Client Implementation (`admin.py`)

An admin client is provided to query the server for a list of current records:

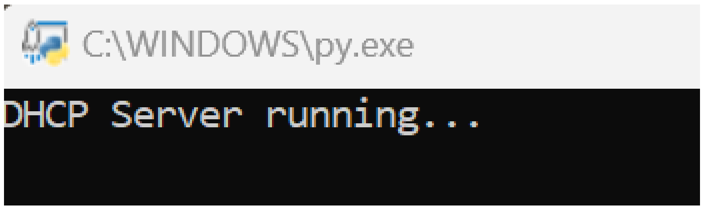
- **List Request:** The admin client sends a "LIST" message to the server.
- **Server Response:** The admin client displays the list of current records received from the server.
- **User-Friendly Menu:**
    - Clear Prompts: Design clear and concise prompts that guide the administrator through available options.
    - Menu System: Implement a structured menu system that organizes different functionalities.
    - Option Selection: Allow the administrator to easily select options from the menu.
    - Informative Output: Ensure that the output, especially the list of records, is presented in an easily understandable format.
- **Interactive Interface:**
    - Create an interactive interface that responds to administrator input promptly.
    - Provide feedback messages to acknowledge successful actions or notify about potential issues.

## Testing Steps

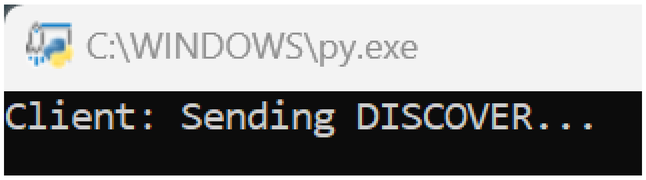
The project provides a testing plan with specific steps to ensure proper functionality. Key test scenarios include:

Client Obtains IP Address
Prevents Duplicate IP Assignment
Renewing IP Address Lease
Releasing IP Address
Admin Client Lists Current Records
Handling DoS Attack (Extra Credit)

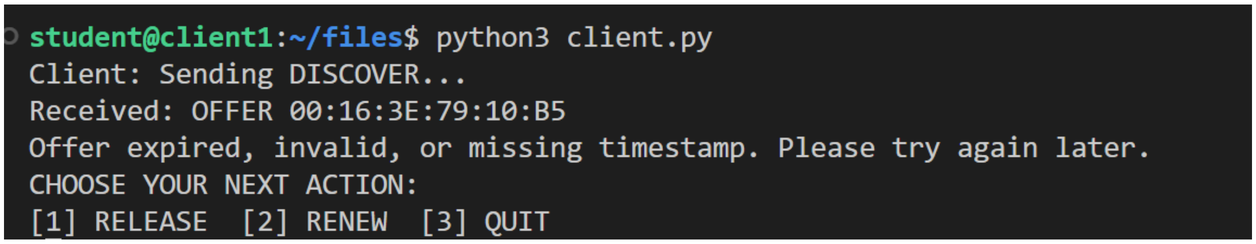
Screenshots :



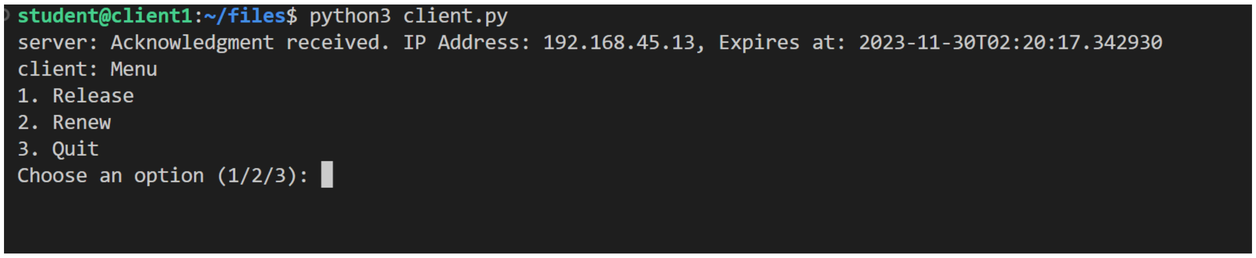
```
C:\WINDOWS\py.exe
DHCP Server running...
```



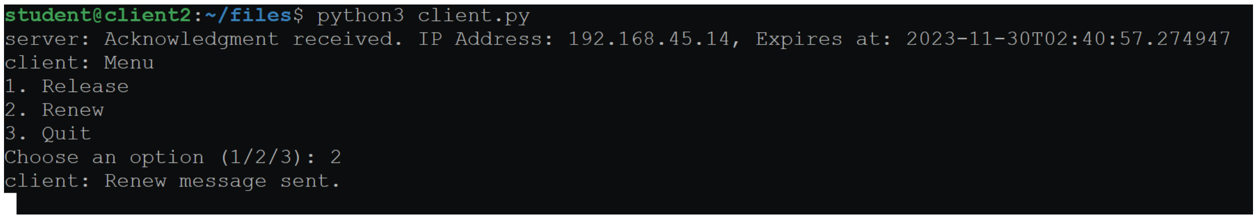
```
C:\WINDOWS\py.exe
Client: Sending DISCOVER...
```



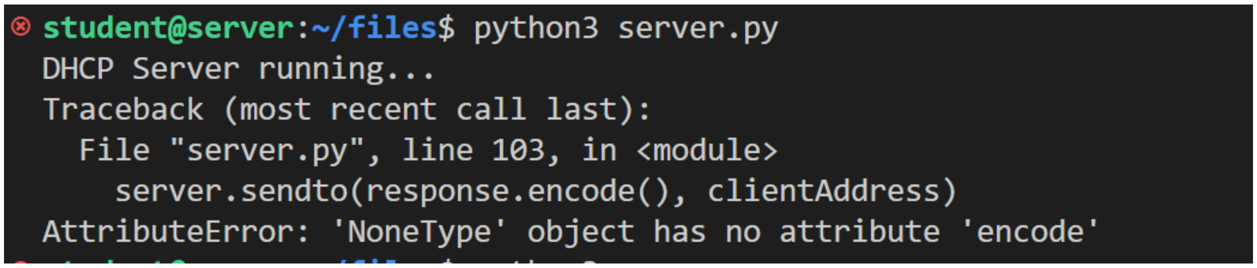
```
student@client1:~/files$ python3 client.py
Client: Sending DISCOVER...
Received: OFFER 00:16:3E:79:10:B5
Offer expired, invalid, or missing timestamp. Please try again later.
CHOOSE YOUR NEXT ACTION:
[1] RELEASE  [2] RENEW  [3] QUIT
```



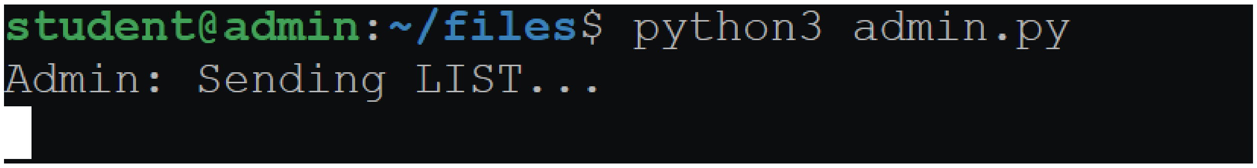
```
student@client1:~/files$ python3 client.py
server: Acknowledgment received. IP Address: 192.168.45.13, Expires at: 2023-11-30T02:20:17.342930
client: Menu
1. Release
2. Renew
3. Quit
Choose an option (1/2/3):
```



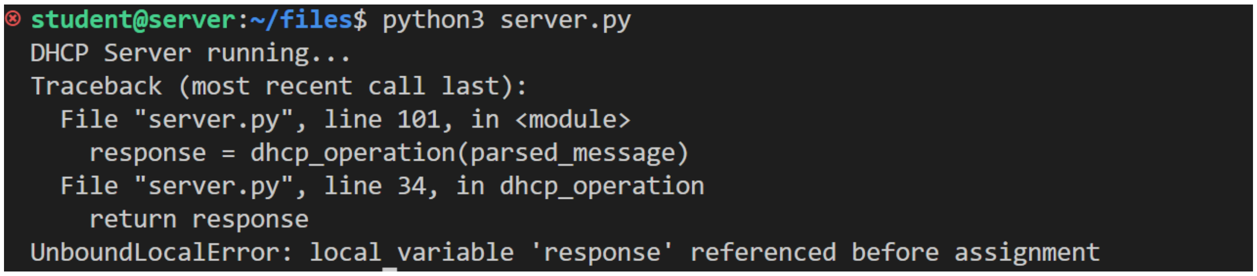
```
student@client2:~/files$ python3 client.py
server: Acknowledgment received. IP Address: 192.168.45.14, Expires at: 2023-11-30T02:40:57.274947
client: Menu
1. Release
2. Renew
3. Quit
Choose an option (1/2/3): 2
client: Renew message sent.
```



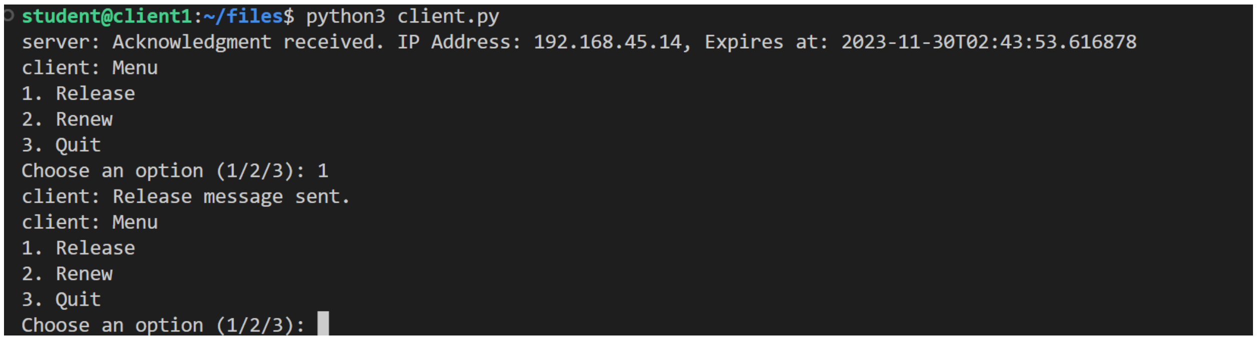
```
student@server:~/files$ python3 server.py
DHCP Server running...
Traceback (most recent call last):
  File "server.py", line 103, in <module>
    server.sendto(response.encode(), clientAddress)
AttributeError: 'NoneType' object has no attribute 'encode'
```



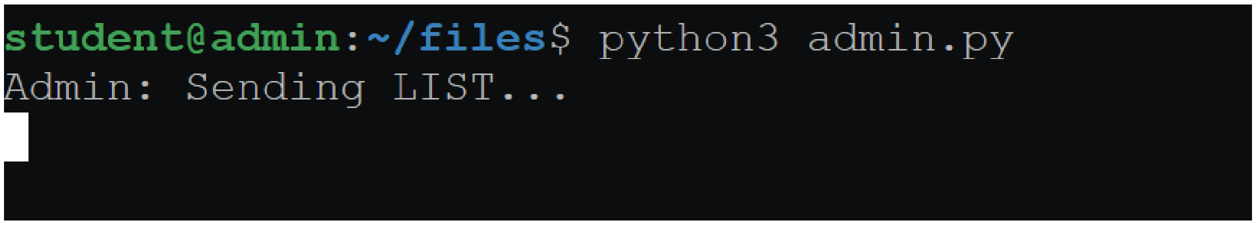
```
student@admin:~/files$ python3 admin.py
Admin: Sending LIST...
```



```
student@server:~/files$ python3 server.py
DHCP Server running...
Traceback (most recent call last):
  File "server.py", line 101, in <module>
    response = dhcp_operation(parsed_message)
  File "server.py", line 34, in dhcp_operation
    return response
UnboundLocalError: local variable 'response' referenced before assignment
```



```
student@client1:~/files$ python3 client.py
server: Acknowledgment received. IP Address: 192.168.45.14, Expires at: 2023-11-30T02:43:53.616878
client: Menu
1. Release
2. Renew
3. Quit
Choose an option (1/2/3): 1
client: Release message sent.
client: Menu
1. Release
2. Renew
3. Quit
Choose an option (1/2/3):
```


```
student@admin:~/files$ python3 admin.py
Admin: Sending LIST...
```

## Challenges and Solutions

During the development process, several challenges were encountered:

- Concurrency Issues: Ensured the server avoids assigning the same IP address to multiple clients simultaneously.
- Error Handling: Implemented proper error handling to manage unexpected scenarios and ensure robustness.
- One notable challenge in the project was establishing a reliable connection between the client and server, ensuring seamless communication. Overcoming this hurdle involved meticulous troubleshooting and synchronization of socket configurations to facilitate accurate data exchange between the two components.
- When attempting to run the admin client, we encountered an issue where sending a "LIST" request did not generate the expected list of records. This limitation hindered our ability to capture screenshots displaying the client information in the admin list. Additionally, the server's tendency to crash intermittently posed a challenge, requiring us to restart the testing process multiple times. These issues contributed to disruptions in the testing workflow and impacted the comprehensive evaluation of the client-server application.

## Conclusion

The DHCP client-server project effectively shows the use of Python socket programming to construct a simpler DHCP protocol. The server manages IP address assignments efficiently, and clients interact with the server to obtain, renew, release, and query IP addresses.

The testing scenarios demonstrate the DHCP protocol's dependability and accuracy. Additional features, such as an admin client and resistance against DoS assaults, round out the project's functionality.

Overall, the project demonstrates good client-server communication, adherence to the DHCP protocol, and robust IP address management.

## References

- GeeksforGeeks. (n.d.). Socket Programming in Python. Retrieved from https://www.geeksforgeeks.org/socket-programming-python/
- Real Python. (n.d.). Python Socket Programming. Retrieved from https://realpython.com/python-sockets/
- Corey Schafer. (Year, Month Day of Video Publication). Socket Programming in Python. [Video file]. Retrieved from https://www.youtube.com/watch?v=3QiPPX-KeSc