Computer Networks Lab Report - Assignment 7

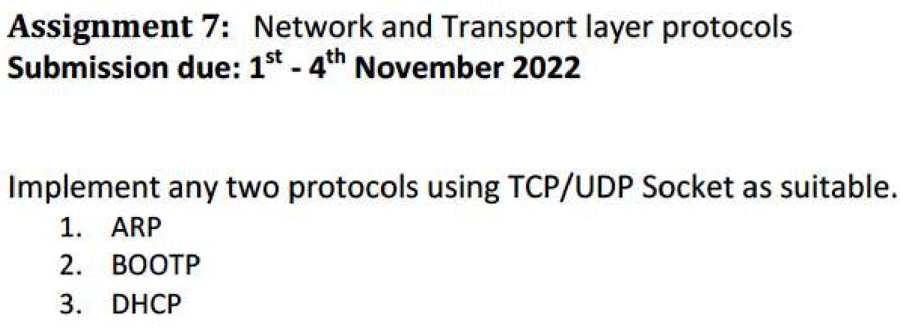
Name: Bikash Sah

Class: BCSE-3

Group: A1

Assignment Number: 7

Problem Statement:



Assignment 7: Network and Transport layer protocols
Submission due: 1st - 4th November 2022

Implement any two protocols using TCP/UDP Socket as suitable.

1. ARP
2. BOOTP
3. DHCP

Submission Date: **21 November, 2022**

Deadline: 4th November, 2022

Introduction

When a hosts send data to another host in a network, it generally gives destination ip address. Host to host delivery can only be acheived using MAC Addresses. Destination MAC addresses can be known using ARP Protocol which maps the mac address to its ip address.

DHCP is used to assign IP Addresses in a network based on availability of IP.

Design

ARP

Design of the Server:

The server is designed to be a simple ARP server. It will receive ARP packets from the clients and will send them to the destination client. The server will also send the ARP reply to the client who requested the ARP reply.

Design of the Client:

The client is designed to be a simple ARP client. It will send ARP packets to the server and will receive the ARP reply from the server. The client will also send the ARP reply to the destination client.

DHCP

Design of the Server:

The server will be designed in such a way that it will be able to handle multiple clients at the same time.

Initially, the server will be started and it will be waiting for the clients to connect to it. It will wait for the clients to send the request for the IP address.

The server will then check if the IP address is available or not. If the IP address is available, then the server will send the IP address to the client.

If the IP address is not available, then the server will send a message to the client that the IP address is not available.

The server will also be able to release the IP address if the client is not using it anymore.

Design of the Client:

The client will be designed in such a way that it will be able to send the request for the IP address to the server.

The client will also be able to release the IP address if it is not using it anymore.

The client will also be able to send the request for the IP address to the server if it has released the IP address.

Implementation

ARP:

Server

```
import socket
import select

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 9999
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("[ SERVER ] Starting Server...")
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((IP, PORT))
print("[ SERVER ] Server binded to IP: " + IP + " and Port: " + str(PORT))
# This makes server listen to new connections
server_socket.listen()
# List of sockets for select.select()
sockets_list = [server_socket]
# List of connected clients - socket as a key, user header and name as data
clients = {}
# for storing the ARP requesting clients
ARP_request = {}
print(f'[ SERVER ] Listening for Connections on IP : {IP} PORT : {PORT}...')


# Handles message receiving
def receive_message(client_socket):
    try:
        # Receive our "header" containing message length, it's size is defined and con
stant
        message_header = client_socket.recv(HEADER_LENGTH)
        if not len(message_header):
            return False
        # Convert header to int value
        message_length = int(message_header.decode('utf-8').strip())
        # Return an object of message header and message data
        return {'header': message_header, 'data': client_socket.recv(message_length)}

    except:
        return False


while True:

    read_sockets, _, exception_sockets = select.select(sockets_list, [], sockets_list)
    # Iterate over notified sockets
    for notified_socket in read_sockets:
        # If notified socket is a server socket - new connection, accept it
        if notified_socket == server_socket:
            # Accept new connection
            # That gives us new socket - client socket, connected to this given client
only, it's unique for that client
            # The other returned object is ip/port set
            client_socket, client_address = server_socket.accept()
            # Client should send his name right away, receive it
            user = receive_message(client_socket)

            # If False - client disconnected before he sent his name
            if user is False:
                continue
                # Add accepted socket to select.select() list
            sockets_list.append(client_socket)
            # Also save username and username header
            clients[client_socket] = user

            print('[ SERVER ] Accepted new Connection from username IP: {}'.format(use
r['data'].decode('utf-8')))

            # Else existing socket is sending a message
        else:
            # Receive message
            message = receive_message(notified_socket)

            # If False, client disconnected, cleanup
            if message is False:
                print('[ SERVER ] Closed connection from: {}'.format(clients[notified_
socket]['data'].decode('utf-8')))

                # Remove from list for socket.socket()
                sockets_list.remove(notified_socket)

                # Remove from our list of users
                del clients[notified_socket]

                continue
                # Get user by notified socket, so we will know who sent the message
            user = clients[notified_socket]

            print(f'[ SERVER ] Received message from IP {user["data"].decode("utf-
8")}:')
```

```
            # Iterate over connected clients and broadcast message
            # Splitting the incoming packet
            ARP_packet = message["data"].decode("utf-8").split()
            print("------------------SENDING PACKET ----------------------")
            print(f"[ SERVER ] SENDER IP: {ARP_packet[0]}")
            print(f"[ SERVER ] SENDER MAC: {ARP_packet[1]}")
            print("----------------------------------------------------------")
            print(f"[ SERVER ] RECEIVER IP: {ARP_packet[2]}")
            print(f"[ SERVER ] RECEIVER MAC: {ARP_packet[3]}")
            print("----------------------------------------------------------")

            # this is ARP request
            if ARP_packet[3] == "FF.FF.FF.FF.FF.FF":
                ARP_request[ARP_packet[0]] = notified_socket
                for client_socket in clients:

                    # But don't send it to sender
                    if client_socket != notified_socket:
                        # Send user and message (both with their headers) We are reusi
ng here message header sent by
                        # sender, and saved username header send by user when he conne
cted
                        client_socket.send(user['header'] + user['data'] + message['he
ader'] + message['data'])
            # else it is an ARP reply therefore it must be uni-cast.
            else:
                if ARP_packet[2] in ARP_request:
                    ARP_request[ARP_packet[2]].send(user['header'] + user['data'] + me
ssage['header'] + message['data'])

        # It's not really necessary to have this, but will handle some socket exceptio
ns just in case
    for notified_socket in exception_sockets:
        # Remove from list for socket.socket()
        sockets_list.remove(notified_socket)

        # Remove from our list of users
        del clients[notified_socket]
```

Client

```
import socket
import select
import errno
import sys

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 9999
my_IP = input("[ CLIENT ] Enter the IP for this Client: ")
my_Mac = input("[ CLIENT ] Enter the MAC for this Client: ")
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((IP, PORT))
```

```
# Set connection to non-blocking state, so .recv() call won;t block, just return some
 exception we'll handle
client_socket.setblocking(False)
username = my_IP.encode('utf-8')
username_header = f"{len(username):<{HEADER_LENGTH}}".encode('utf-8')
client_socket.send(username_header + username)
while True:

    ip_receiver = input("[ CLIENT ] Enter the IP of Receiver: ")


    # this is for ARP request

    if ip_receiver:
        message = my_IP + " " + my_Mac + " " + ip_receiver + " " + "FF.FF.FF.FF.FF.FF"
        # Encode message to bytes, prepare header and convert to bytes, like for usern
ame above, then send
        message = message.encode('utf-8')
        message_header = f"{len(message):<{HEADER_LENGTH}}".encode('utf-8')
        client_socket.send(message_header + message)
    try:
        # Now we want to loop over received messages (there might be more than one) an
d print them
        while True:
            # Receive our "header" containing username length, it's size is defined an
d constant
            username_header = client_socket.recv(HEADER_LENGTH)
            # If we received no data, server gracefully closed a connection, for examp
le using socket.close() or
            # socket.shutdown(socket.SHUT_RDWR)
            if not len(username_header):
                print("[ CLIENT ] Connection closed by the Server...")
                sys.exit()

            # Convert header to int value
            username_length = int(username_header.decode('utf-8').strip())

            # Receive and decode username
            username = client_socket.recv(username_length).decode('utf-8')
            # Now do the same for message (as we received username, we received whole
 message, there's no need to
            # check if it has any length)
            message_header = client_socket.recv(HEADER_LENGTH)
            message_length = int(message_header.decode('utf-8').strip())
            message = client_socket.recv(message_length).decode('utf-8')
            ARP_request = message.split()
            if ARP_request[2] == my_IP:
                ARP_reply = my_IP + " " + my_Mac + " " + ARP_request[0] + " " + ARP_re
quest[1]
                ARP_reply = ARP_reply.encode('utf-8')
                ARQ_header = f"{len(ARP_reply):<{HEADER_LENGTH}}".encode('utf-8')
                client_socket.send(ARQ_header + ARP_reply)
            else:
                print("----------------------------------------")
                print("[ CLIENT ] Discarded the Request...")
                print("----------------------------------------")
            # Print message
            print(f'{username} > {message}')
    except IOError as e:
        # This is normal on non blocking connections - when there are no incoming data
error is going to be raised
        # Some operating systems will indicate that using AGAIN, and some using WOULDB
LOCK error code
        # We are going to check for both - if one of them - that's expected, means no
 incoming data, continue as normal
        # If we got different error code - something happened
        if e.errno != errno.EAGAIN and e.errno != errno.EWOULDBLOCK:
            print('[ CLIENT ] Reading Error: {}'.format(str(e)))
            sys.exit()

        # We just did not receive anything
        continue

    except Exception as e:
        # Any other exception - something happened, exit
        print(' [ CLIENT ] Reading Error: '.format(str(e)))
        sys.exit()
```

DHCP:

Server

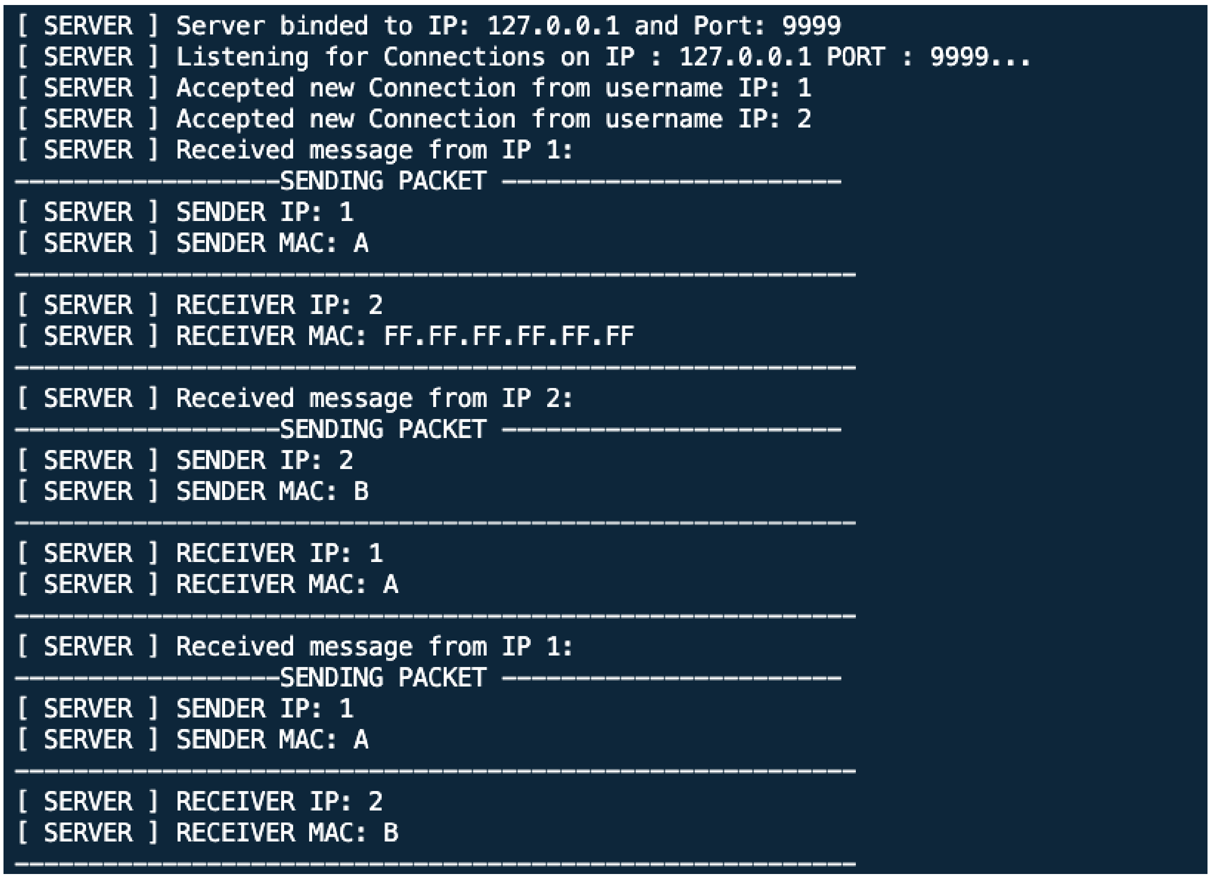
```
import socket
import select

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 1234
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("[ Server ] Starting Server...")
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((IP, PORT))
# This makes server listen to new connections
server_socket.listen()
# List of sockets for select.select()
sockets_list = [server_socket]
# List of connected clients - socket as a key, user header and name as data
clients = {}
# for storing the ARP requesting clients
print("[ Server ] Waiting for connections...")
subnet_mask = "255.255.255.0"
network = "168.173.10."
available_ip = [True] * 254
available_ip[0] = False


# Handles message receiving
def receive_message(client_socket):
    try:
        # Receive our "header" containing message length, it's size is defined and con
stant
        message_header = client_socket.recv(HEADER_LENGTH)
        if not len(message_header):
            return False
        # Convert header to int value
        message_length = int(message_header.decode('utf-8').strip())
        # Return an object of message header and message data
        return {'header': message_header, 'data': client_socket.recv(message_length)}

    except:
        return False


while True:

    read_sockets, _, exception_sockets = select.select(sockets_list, [], sockets_list)
    for notified_socket in read_sockets:
        if notified_socket == server_socket:
            client_socket, client_address = server_socket.accept()
            user = receive_message(client_socket)
            if user is False:
                continue
            sockets_list.append(client_socket)
            clients[client_socket] = user
            print('[ SERVER ] Accepted new connection from username IP: {}'.format(use
r['data'].decode('utf-8')))

            # Else existing socket is sending a message
        else:
            # Receive message
            message = receive_message(notified_socket)
            if message is False:
                print('[ SERVER ] Closed Connection from: {}'.format(clients[notified_
socket]['data'].decode('utf-8')))
                sockets_list.remove(notified_socket)
                del clients[notified_socket]
                continue
                # Get user by notified socket, so we will know who sent the message
            user = clients[notified_socket]

            print(f'[ SERVER ] Received DHCP request from Client {user["data"].decode
("utf-8")}:')
            ARP_packet = message["data"].decode("utf-8").split()
            if ARP_packet[0] == "release":
                print("[ SERVER ] Releasing...")
                index = ARP_packet[1][11:]
                index = int(index)
                available_ip[index] = True
                DHCP_reply = "0.0.0.0"
                DHCP_reply = DHCP_reply.encode('utf-8')
                DHCP_header = f"{len(DHCP_reply):<{HEADER_LENGTH}}".encode('utf-8')
                notified_socket.send(user['header'] + DHCP_header + DHCP_reply)
                print("[ SERVER ] DHCP Reply Sent...")

            if ARP_packet[0] == "request":
                print("[ SERVER ] Requesting...")
                for i in range(1, 254):
                    if available_ip[i]:
                        available_ip[i]=False
                        print("[ SERVER ] Available!")
                        DHCP_reply = network + str(i)
                        print(DHCP_reply)
                        DHCP_reply = DHCP_reply.encode('utf-8')
                        DHCP_header = f"{len(DHCP_reply):<{HEADER_LENGTH}}".encode('ut
f-8')
                        notified_socket.send(user['header'] + DHCP_header + DHCP_repl
y)
                        print("[ SERVER ] DHCP Reply Sent...")
                        break

        # It's not really necessary to have this, but will handle some socket exceptio
ns just in case
    for notified_socket in exception_sockets:
        # Remove from list for socket.socket()
        sockets_list.remove(notified_socket)

        # Remove from our list of users
        del clients[notified_socket]
```

Client

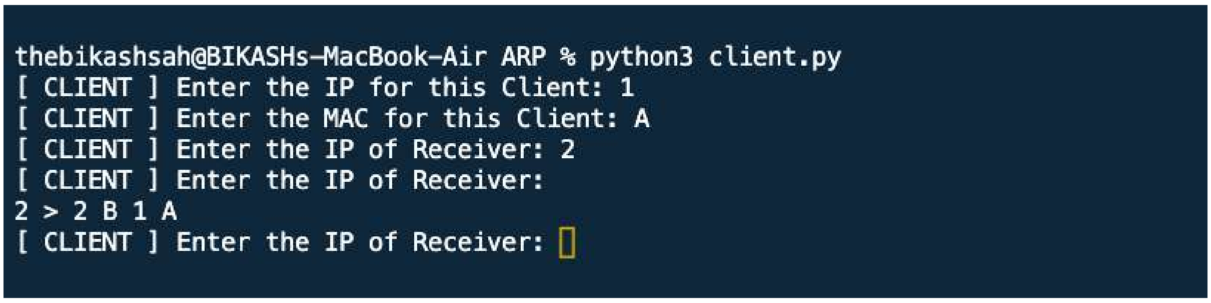
```
import socket
import select
import errno
import sys

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 1234
my_IP = "0.0.0.0"
username=input("[ CLIENT ] Enter Client's name : ")
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((IP, PORT))
# Set connection to non-blocking state, so .recv() call won;t block, just return some
 exception we'll handle
client_socket.setblocking(False)
username = username.encode('utf-8')
username_header = f"{len(username):<{HEADER_LENGTH}}".encode('utf-8')
client_socket.send(username_header + username)
while True:
    message = "ipconfig"
    while message == "ipconfig":
        message = input("[ CLIENT ] Enter message : ")
        print("________________________________")
        print(f"[ CLIENT ] IP :{my_IP}")
        print("________________________________")

    if message:
        message = message + " " + my_IP
        # Encode message to bytes, prepare header and convert to bytes, like for usern
ame above, then send
        message = message.encode('utf-8')
        message_header = f"{len(message):<{HEADER_LENGTH}}".encode('utf-8')
        client_socket.send(message_header + message)
    try:
        while True:
            username_header = client_socket.recv(HEADER_LENGTH)
            if not len(username_header):
                print('[ CLIENT ] : Connection closed by the Server...')
                sys.exit()
            message_header = client_socket.recv(HEADER_LENGTH)
            print(message_header.decode('utf-8'))
            message_length = int(message_header.decode('utf-8').strip())
            message = client_socket.recv(message_length).decode('utf-8')
            print(message)
            my_IP = message

    except IOError as e:
        # This is normal on non blocking connections - when there are no incoming data
error is going to be raised
        # Some operating systems will indicate that using AGAIN, and some using WOULDB
LOCK error code
        # We are going to check for both - if one of them - that's expected, means no
 incoming data, continue as normal
        # If we got different error code - something happened
        if e.errno != errno.EAGAIN and e.errno != errno.EWOULDBLOCK:
            print('[ CLIENT ] Reading error: {}'.format(str(e)))
            sys.exit()

        # We just did not receive anything
        continue

    except Exception as e:
        # Any other exception - something happened, exit
        print('[ CLIENT ] Reading error: '.format(str(e)))
        sys.exit()
```

Test Cases and Results

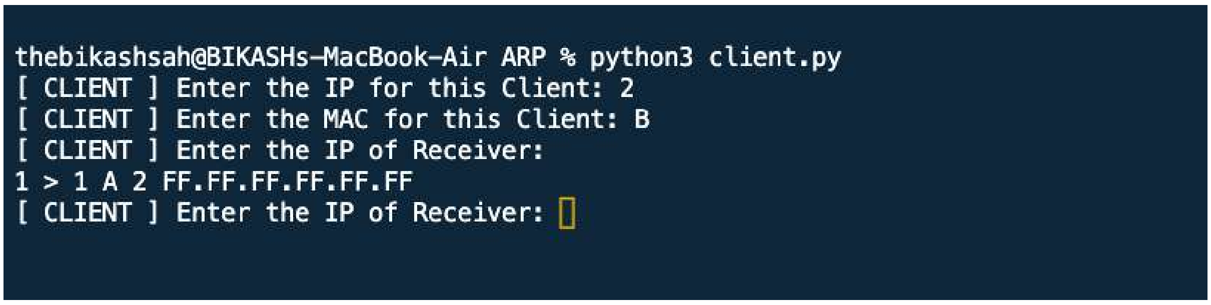
ARP:



```
[ SERVER ] Server binded to IP: 127.0.0.1 and Port: 9999
[ SERVER ] Listening for Connections on IP : 127.0.0.1 PORT : 9999...
[ SERVER ] Accepted new Connection from username IP: 1
[ SERVER ] Accepted new Connection from username IP: 2
[ SERVER ] Received message from IP 1:
------------------SENDING PACKET ------------------------
[ SERVER ] SENDER IP: 1
[ SERVER ] SENDER MAC: A
---------------------------------------------------------
[ SERVER ] RECEIVER IP: 2
[ SERVER ] RECEIVER MAC: FF.FF.FF.FF.FF.FF
---------------------------------------------------------
[ SERVER ] Received message from IP 2:
------------------SENDING PACKET ------------------------
[ SERVER ] SENDER IP: 2
[ SERVER ] SENDER MAC: B
---------------------------------------------------------
[ SERVER ] RECEIVER IP: 1
[ SERVER ] RECEIVER MAC: A
---------------------------------------------------------
[ SERVER ] Received message from IP 1:
------------------SENDING PACKET ------------------------
[ SERVER ] SENDER IP: 1
[ SERVER ] SENDER MAC: A
---------------------------------------------------------
[ SERVER ] RECEIVER IP: 2
[ SERVER ] RECEIVER MAC: B
---------------------------------------------------------
```

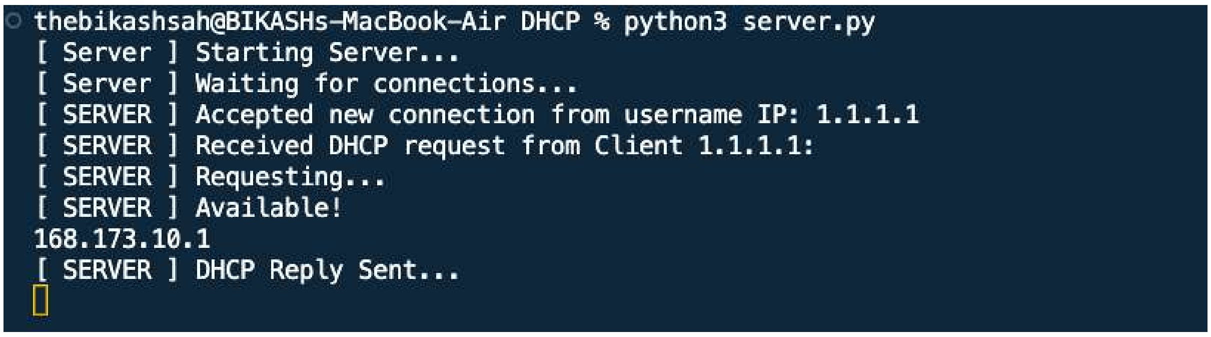


```
thebikashsah@BIKASHs-MacBook-Air ARP % python3 client.py
[ CLIENT ] Enter the IP for this Client: 1
[ CLIENT ] Enter the MAC for this Client: A
[ CLIENT ] Enter the IP of Receiver: 2
[ CLIENT ] Enter the IP of Receiver:
2 > 2 B 1 A
[ CLIENT ] Enter the IP of Receiver: 
```

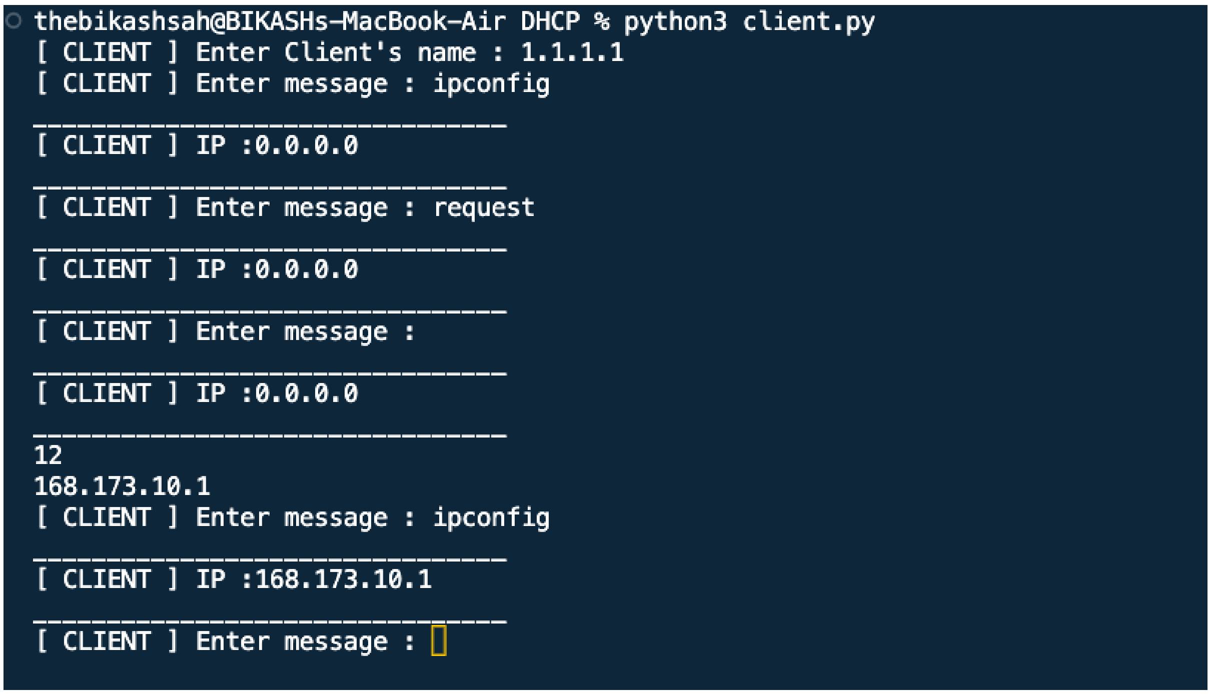


```
thebikashsah@BIKASHs-MacBook-Air ARP % python3 client.py
[ CLIENT ] Enter the IP for this Client: 2
[ CLIENT ] Enter the MAC for this Client: B
[ CLIENT ] Enter the IP of Receiver:
1 > 1 A 2 FF.FF.FF.FF.FF.FF
[ CLIENT ] Enter the IP of Receiver: 
```

DHCP:



```
thebikashsah@BIKASHs-MacBook-Air DHCP % python3 server.py
[ Server ] Starting Server...
[ Server ] Waiting for connections...
[ SERVER ] Accepted new connection from username IP: 1.1.1.1
[ SERVER ] Received DHCP request from Client 1.1.1.1:
[ SERVER ] Requesting...
[ SERVER ] Available!
168.173.10.1
[ SERVER ] DHCP Reply Sent...
```



```
thebikashsah@BIKASHs-MacBook-Air DHCP % python3 client.py
[ CLIENT ] Enter Client's name : 1.1.1.1
[ CLIENT ] Enter message : ipconfig
_________________________________
[ CLIENT ] IP :0.0.0.0
_________________________________
[ CLIENT ] Enter message : request
_________________________________
[ CLIENT ] IP :0.0.0.0
_________________________________
[ CLIENT ] Enter message :
_________________________________
[ CLIENT ] IP :0.0.0.0
_________________________________
12
168.173.10.1
[ CLIENT ] Enter message : ipconfig
_________________________________
[ CLIENT ] IP :168.173.10.1
_________________________________
[ CLIENT ] Enter message :
```

Comments

> This assignment was like a real life project, it had a problem statement and I had to come up with a solution, I have learnt a lot, I learnt a lot of python in this assignment and also learnt how to implement big problems by dividing it into subproblems.