COMPUTER NETWORKS LAB

NAME:	ARITRA DATTA
ROLL NO:	002010501054
CLASS:	BCSE – III
GROUP:	A2
ASSIGNMENT:	7
DEADLINE:	4th November 2022

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

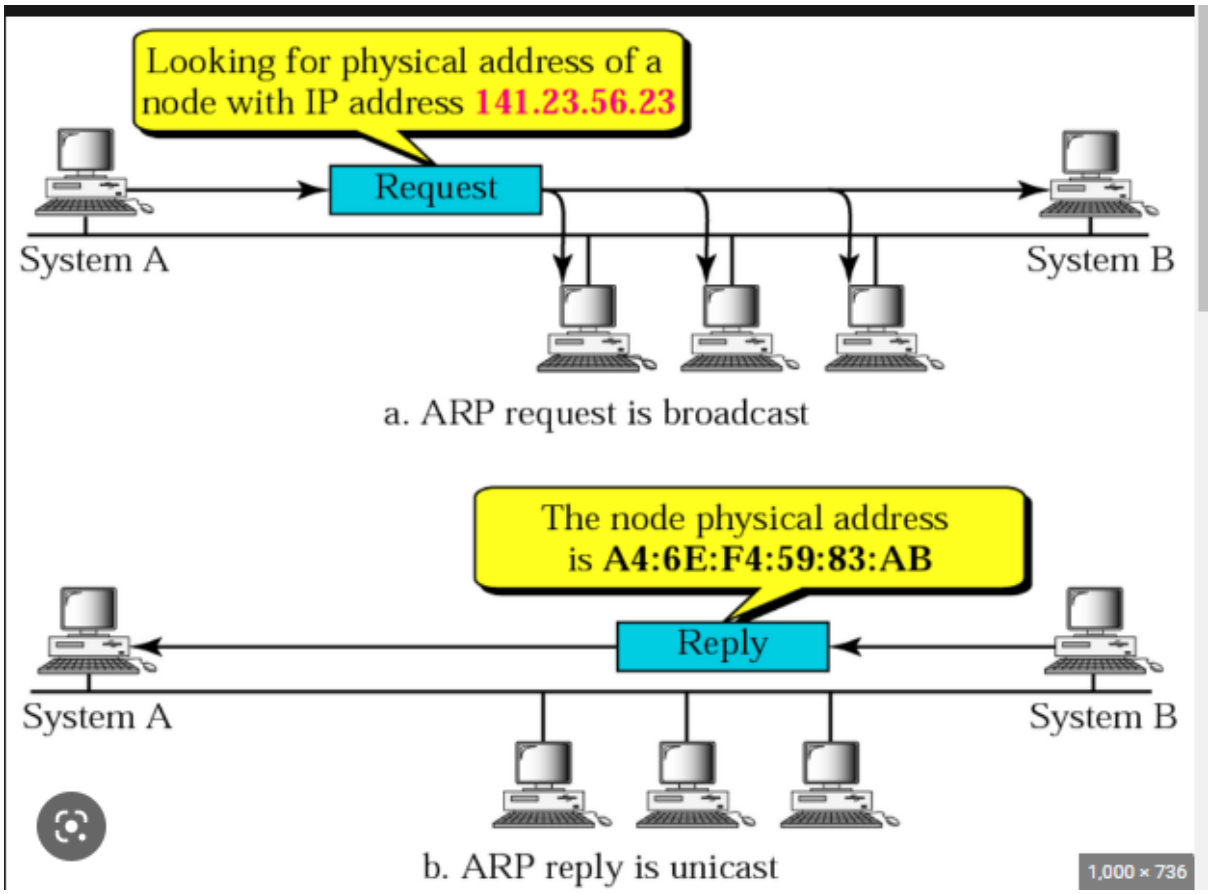
1. ARP
2. BOOTP
3. DHCP

Date of Submission: 7th November, 2022

DESIGN

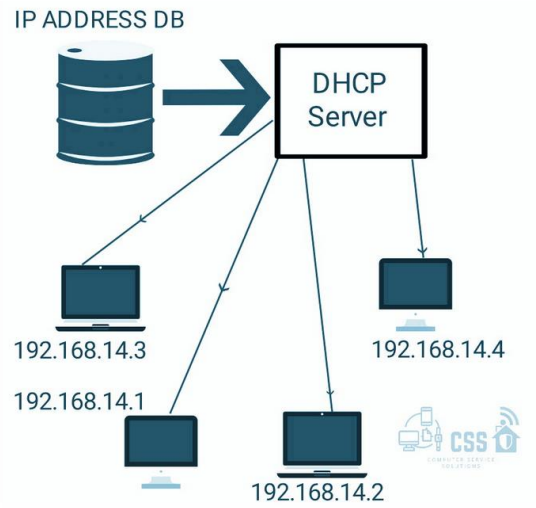
ARP (Address Resolution Protocol)

ARP has been implemented using TCP sockets and **socket.SOCK_STREAM** has been used accordingly. Here every client sends a Broadcast message with the destination IP whose MAC it wants to all clients and only the client that matches the IP sends a Unicast message with it's IP and it's Mac to the client from which the message came and thus the original client can get the MAC from the src Mac portion of the received unicast packet



DHCP (Dynamic Host Control Protocol)

DHCP is implemented using UDP sockets here and **socket.SOCK_DGRAM** has been used accordingly. Whenever a client connects to the DHCP Server, it requests a temporary IP from DHCP. The DHCP maintains a set of available IPs, from which it pops an IP, and allocates that for the requesting client. Now, whenever the client disconnects, it sends its temporary IP as a message to the DHCP Server, which then removes it from current clients, and adds the IP back to the set of available IPs, making it available for allocation for some different clients.



CODE

ARP_Server.py

```
import socket
import threading
import time
import random

IP = socket.gethostbyname(socket.gethostname())
PORT = 4453
ADDR = (IP, PORT)
FORMAT = "utf-8"
HEADERSIZE = 10
IPMASK = "192.168.1."
BROADCAST = "FF:FF:FF:FF:FF:FF"
MAC_DATABASE = ["00-04-3A-5F-66-3A","01-34-47-55-3B-3A","00-24-5A-6C-56-3A","00-05-3D-2A-33-5A"]
ClientMap = {}
clients = []

def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode(FORMAT).strip())

        data = client_socket.recv(msg_len).decode(FORMAT)
        return data

    except:
        return False

def createFrame(message):
    return  f"{len(message):<{HEADERSIZE}}" + message

def broadcast(message,node):
    for client in clients:
        if client == node:
            continue
        client.send(message)

def handle_client(client):
    while True:
        try:
            message = receive_message(client)
            packet = message.split('$')
            if packet[-1] == BROADCAST:
                broadcast(createFrame(message).encode(FORMAT),client)
            else:
                unicast_ip = packet[-1]
```

```
        except:
            index = clients.index(client)
            clients.remove(client)
            client.close()
            break

def StartNetwork():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(ADDR)
    server.listen()
    num = 0
    print('Network is ready to Start ...')
    while True:
        client, address = server.accept()
        num += 1
        IP = IPMASK + str(addr)
        MAC = MAC_DATABASE[addr-1]
        print(f'Connection Established with Client {num}, IP : {IP}')
        clients.append(client)
        ClientMap[IP] = client
        client.send(createFrame(f'IP : {IP} ,  MAC : {MAC} is Connected to  the Network\n').encode(FORMAT))
        thread = threading.Thread(target=handle_client, args=(client,))
        thread.start()


if __name__ == "__main__":
    StartNetwork()
```

ARP_Client.py

```
import threading
import socket

IP = socket.gethostbyname(socket.gethostname())
PORT = 4453
ADDR = (IP, PORT)
FORMAT = "utf-8"
HEADERSIZE = 10
IP = ""
MAC = ""
BROADCAST = "FF:FF:FF:FF:FF:FF"


client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)

def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode(FORMAT).strip())

        data = client_socket.recv(msg_len).decode(FORMAT)
        return data

    except:
        return False

def getip(msg):
    i = msg.index(',')
    return msg[5:i].strip()

def getmac(msg):
    i = msg.index('M')
    j = msg.index('is')
    return msg[i+5:j].strip()

def createFrame(message):
    return  f"{len(message):<{HEADERSIZE}}" + message
```

OUTPUT

SERVER

```
E:\NETWORK\L3 Protocols\ARP>python server.py
Network is ready to Start ...
Connection Established with Client 1, IP : 192.168.1.1
Connection Established with Client 2, IP : 192.168.1.2
Connection Established with Client 3, IP : 192.168.1.3
```

Client 1

```
E:\NETWORK\L3 Protocols\ARP>python client.py

IP : 192.168.1.1 ,  MAC : 00-04-3A-5F-66-3A is Connected to  the Network


Packet Rejected
192.168.1.3
BROADCAST Message sent
MAC : 00-24-5A-6C-56-3A received for IP : 192.168.1.3

UNICAST Message sent
```

Client 2

```
E:\NETWORK\L3 Protocols\ARP>python client.py

IP : 192.168.1.3 ,  MAC : 00-24-5A-6C-56-3A is Connected to  the Network

192.168.1.2
BROADCAST Message sent
MAC : 01-34-47-55-3B-3A received for IP : 192.168.1.2

UNICAST Message sent


Packet Rejected
```

Client 3

```
E:\NETWORK\L3 Protocols\ARP>python client.py

IP : 192.168.1.2 ,  MAC : 01-34-47-55-3B-3A is Connected to  the Network

UNICAST Message sent


Packet Rejected
192.168.1.1
BROADCAST Message sent
MAC : 00-04-3A-5F-66-3A received for IP : 192.168.1.1

192.168.127.128
BROADCAST Message sent
```

DHCP_Server.py

```python
import socket
import threading
import time
import random

IP = socket.gethostbyname(socket.gethostname())
PORT = 4452
ADDR = (IP, PORT)
FORMAT = "utf-8"
HEADERSIZE = 10
IPMASK = "192.168.1."
IP_POOL = []
free = {}
allocated = {}
clients = []
N = 0

def generateIp():
    for i in range(0,256,5):
        ip = IPMASK + str(random.randint(i,min(255,i+5)))
        global IP_POOL
        IP_POOL.append(ip)
        global free
        free[ip] = True

def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode(FORMAT).strip())

        data = client_socket.recv(msg_len).decode(FORMAT)
        return data

    except:
        return False

def createFrame(message):
    return  f"{len(message):<{HEADERSIZE}}" + message
```

```python
def handle_client(client,lock):
    while True:
        message = receive_message(client)
        global N
        if message == "0"  or  not message:
            lock.acquire()
            N = N + 1
            exit = False
            free[allocated[client]] = True
            print(f"IP : {allocated[client]} got leased out and is now free\n")
            allocated[client] = None
            if not message :
                index = clients.index(client)
                clients.remove(client)
                client.close()
                exit = True
            lock.release()
            if exit:
                break
        elif N > 0 :
            ip = ""
            lock.acquire()
            N -= 1
            for ipp in IP_POOL:
                if free[ipp]:
                    ip = ipp
                    break
            lock.release()
            client.send(createFrame(ip).encode(FORMAT))
            lock.acquire()
            allocated[client] = ip
            free[ip] = False
            print(f"IP : {ip} has been Allocated to a Client\n")
            lock.release()
        else:
            client.send(createFrame("0").encode(FORMAT))
```

```
def main():
    generateIp()
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(ADDR)
    server.listen()
    global N
    N = int(input('Enter Capacity of DHCP Server : '))
    lock = threading.Lock()
    while True:
        client, address = server.accept()
        if N > 0:
            clients.append(client)
            ip = ""
            N -= 1
            for ipp in IP_POOL:
                if free[ipp]:
                    ip = ipp
                    break
            client.send(createFrame(ip).encode(FORMAT))
            allocated[client] = ip
            free[ip] = False
            print(f"IP : {ip} has been Allocated to a Client\n")
            thread = threading.Thread(target=handle_client, args=(client,lock))
            thread.start()
        else:
            client.send(createFrame("0").encode(FORMAT))


if __name__ == "__main__":
    main()
```

DHCP_Client.py

```
IP = socket.gethostbyname(socket.gethostname())
PORT = 4452
ADDR = (IP, PORT)
FORMAT = "utf-8"
HEADERSIZE = 10
LEASE_TIME = 10


def receive_message(client_socket):
    try:
        msg_header = client_socket.recv(HEADERSIZE)

        if not len(msg_header):
            return False

        msg_len = int(msg_header.decode(FORMAT).strip())

        data = client_socket.recv(msg_len).decode(FORMAT)
        return data

    except:
        return False

def createFrame(message):
    return  f"{len(message):<{HEADERSIZE}}" + message

def clientStart(client):
    while True:
        message = receive_message(client)
        if not message  or  message == "0":
            print("Can't Connect to the DHCP Server\n")
            client.close()
            break
        else:
            print(f"Client Online IP : {message}")
            time.sleep(LEASE_TIME)
            client.send(createFrame("0").encode(FORMAT))
            print("Client Offline\n")
            time.sleep(2)
            print("IP Renewal Request Sent")
            client.send(createFrame("1").encode(FORMAT))


def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(ADDR)
    clientStart(client)

if __name__ == "__main__":
    main()
```

OUTPUT

```
E:\NETWORK\L3 Protocols\DHCP>python server.py
Enter Capacity of DHCP Server : 2
IP : 192.168.1.3 has been Allocated to a Client

IP : 192.168.1.10 has been Allocated to a Client

IP : 192.168.1.3 got leased out and is now free

IP : 192.168.1.10 got leased out and is now free

IP : 192.168.1.3 has been Allocated to a Client

IP : 192.168.1.10 has been Allocated to a Client

IP : None got leased out and is now free

IP : 192.168.1.3 got leased out and is now free

IP : 192.168.1.10 got leased out and is now free

IP : 192.168.1.3 has been Allocated to a Client

IP : 192.168.1.10 has been Allocated to a Client

IP : 192.168.1.10 got leased out and is now free

IP : 192.168.1.3 got leased out and is now free
```

DHCP SERVER

```
E:\NETWORK\L3 Protocols\DHCP>python client.py
Client Online IP : 192.168.1.10
Client Offline

IP Renewal Request Sent
Client Online IP : 192.168.1.10
```

Client 1

```
E:\NETWORK\L3 Protocols\DHCP>python client.py
Client Online IP : 192.168.1.10
Client Offline

IP Renewal Request Sent
Can't Connect to the DHCP Server


E:\NETWORK\L3 Protocols\DHCP>
```

Client 2

```
Client Online IP : 192.168.1.3
Client Offline

IP Renewal Request Sent
Client Online IP : 192.168.1.3
Client Offline

IP Renewal Request Sent
Client Online IP : 192.168.1.3
Client Offline
```

Client 3