**Computer Networks Lab Report**

**Assignment-** 7

**Class:** BCSE-III

**Semester:** 1st

**Group:**  A3

**Group Members:-**

* PARTHIV SARKAR - 002110501134
* ARKAJYOTI NASKAR - 002110501144
* SOHAM CHOWDHURY - 002110501145
* ARIYAN BHAUMIK - 002110501149

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

**1. ARP**

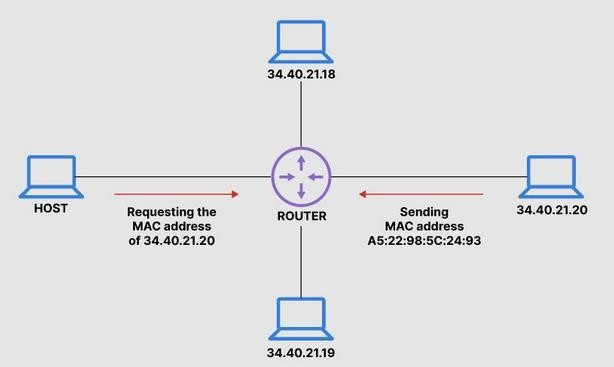
**2. BOOTP**

**3. DHCP**

**BOOTP DESIGN**

***BOOP (Address Resolution Protocol)***

BOOTP has been implemented using UDP sockets and socket. SOCK\_DGRAM has been used accordingly. The server has a static BOOTP table, which maps MAC addresses to IP addresses, or physical addresses to logical addresses. A client sends its MAC address to the server, which in turns searches for it in the BOOTP table. Accordingly, the server sends a response to the client.



**IMPLEMENTATION**

A graph has been implemented where messages are passed using UDP. A message will be broadcasted and when it reaches a relay station, the relay station sends it to the BOOTP sever.

The BOOTP server consults its internal table and sends the reply to the relay station which is routed to the querying node.

**CODE**

import socket as socket

import random

import pickle

import time

from multiprocessing import Process, Manager

def randomMac():

return "02:00:00:%02x:%02x:%02x" % (random.randint(0, 255),

random.randint(0, 255),

random.randint(0, 255))

#frame format [physical,TTL]

def relayStationRoutine(i,serverInd,visited):

host = f"127.0.0.{i}"

port = 8080

host1 = f"127.0.0.{i}"

port1 = 8081

serverHost = f"127.0.0.{serverInd}"

serverPort = 8080

start = time.time()

UDPsock = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)

UDPsock.settimeout(15)

UDPsock.bind((host,port))

UDPsock1 = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

UDPsock1.settimeout(15)

UDPsock1.bind((host1, port1))

while time.time()-start<=15:

try:

msg,addr = UDPsock.recvfrom(1024)

except Exception:

break

msg = pickle.loads(msg)

if msg[1]==0:

continue

msg[1]-=1

print(f"Message reached relay station {i}")

print()

time.sleep(1)

UDPsock.sendto(pickle.dumps(msg),(serverHost,serverPort))

# time.sleep(0.5)

try:

msg,addr = UDPsock1.recvfrom(1024)

except Exception:

break

while addr[0]!=serverHost:

try:

time.sleep(0.5)

msg,addr = UDPsock.recvfrom(1024)

except Exception:

break

msg = pickle.loads(msg)

try:

UDPsock1.sendto(pickle.dumps(msg),(msg[0],port))

except Exception:

pass

UDPsock.close()

UDPsock1.close()

def bootpServerRoutine(i,table):

host = f"127.0.0.{i}"

port = 8080

start = time.time()

UDPsock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

UDPsock.settimeout(15)

UDPsock.bind((host, port))

cnt = 0

msg=[0,0]

addr = [0,0]

while time.time()-start<=15:

if cnt==0:

try:

msg,addr = UDPsock.recvfrom(1024)

except Exception:

break

msg = pickle.loads(msg)

if cnt==0:

print(f"Message reached bootp server")

print()

time.sleep(1)

cnt+=1

if cnt==1:

physicalAddr = msg[0]

ipAddr = table[physicalAddr]

msg = [ipAddr]

UDPsock.sendto(pickle.dumps(msg),(addr[0],port+1))

UDPsock.close()

def nodeRoutine(i,adj,visited):

host = f"127.0.0.{i}"

port = 8080

UDPsock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

UDPsock.settimeout(15)

UDPsock.bind((host, port))

start = time.time()

msg=[0,0]

addr = [0,0]

while time.time()-start<=15:

try:

msg,addr = UDPsock.recvfrom(1024)

msg = pickle.loads(msg)

except Exception:

break

print(f"Message received at {i}")

time.sleep(1)

if msg[1]==0:

continue

msg[1]-=1

for ip in adj[host]:

if visited[ip]:

continue

UDPsock.sendto(pickle.dumps(msg),(ip,port))

visited[ip]=True

UDPsock.close()

def queryRoutine(i,physical,adj,visited):

host = f"127.0.0.{i}"

port = 8080

visited[host]=True

UDPsock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

UDPsock.settimeout(15)

UDPsock.bind((host, port))

start = time.time()

msg = [0, 0]

addr = [0, 0]

print(f"Initializing query node {i}")

print()

time.sleep(1)

msg = [physical,12]

for ip in adj[host]:

if not visited[ip]:

UDPsock.sendto(pickle.dumps(msg),(ip,port))

visited[ip]=True

try:

msg,addr = UDPsock.recvfrom(1024)

msg = pickle.loads(msg)

print(f"Ip address of node is {msg[0]}")

except Exception:

print(f"Ip not found")

finally:

print()

UDPsock.close()

if \_\_name\_\_=='\_\_main\_\_':

n = int(input("Enter number of nodes "))

m = int(input("Enter number of edges "))

manager = Manager()

ind2mac = {i: randomMac() for i in range(1, n+1)}

ind2ip = {i:f"127.0.0.{i}" for i in range(1,n+1)}

adj = {ind2ip[i]:[] for i in range(1,n+1)}

table = {ind2mac[i]:ind2ip[i] for i in range(1,n+1)}

visited = manager.dict()

for ip in ind2ip.values():

visited[ip]=False

print("Enter the edges")

for i in range(m):

st = input()

e1 = int(st.split(" ")[0])

e2 = int(st.split(" ")[1])

adj[ind2ip[e1]].append(ind2ip[e2])

adj[ind2ip[e2]].append(ind2ip[e1])

x = int(input("Continue ? "))

while x==1:

start = int(input("Enter the query node: "))

relays = list(map(lambda x:int(x),input("Enter the relay indices: ").split(" ")))

server = n

nodes = [i for i in range(1,n) if i!=start and i not in relays]

print(start,server,relays,nodes)

startProcess = Process(target=queryRoutine,args=(start,ind2mac[start],adj,visited))

serverProcess = Process(target=bootpServerRoutine,args=(n,table))

relayProcesses = [Process(target=relayStationRoutine,args=(i,n,visited)) for i in relays]

nodeProcesses = [Process(target=nodeRoutine,args=(i,adj,visited)) for i in nodes]

for r in relayProcesses:

r.start()

for n1 in nodeProcesses:

n1.start()

serverProcess.start()

startProcess.start()

startProcess.join()

serverProcess.join()

for r in relayProcesses:

r.join()

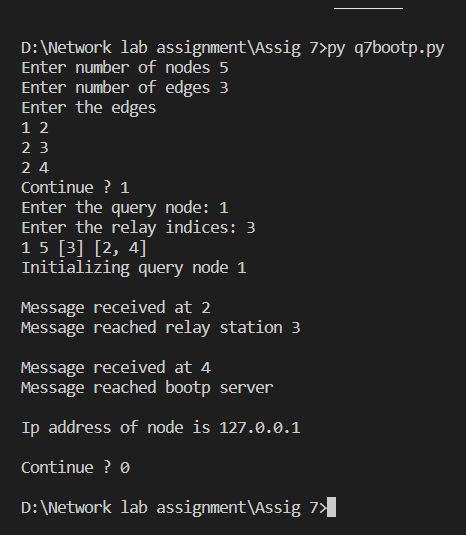
for n1 in nodeProcesses:

n1.join()

for ip in ind2ip.values():

visited[ip] = False

x = int(input("Continue ? "))



**ARP DESIGN**

ARP (Address Resolution Protocol) is a protocol for finding the mac address of a node using the IP address. The ARP request is broadcast on the network and the required node unicasts its reply to the querying node. The implementation is socket. SOCK\_DGRAM, where message are sent from querying node to all the broadcast nodes.

### How does it work?

-> Suppose,

|  |  |  |
| --- | --- | --- |
|  | **IP** | **MAC** |
| Host A | 192.168.1.1 | 1f:39:1b:71:4f:c9 |
| Host B | 192.168.1.2 | d1:69:b7:c5:18:53 |

**Situation:** Host A wants to send a message to Host B, but Host A has not mac of Host B

**Step A:** Host A will create a packet, consists of

* IP of Host A
* MAC of Host A
* IP of Host B
* No MAC for Host B

and will broadcast the packet on the network.

**Step B:** All hosts will receive the packet, and Host B will detect that someone has a query for its MAC. Host B will form a packet, consisting of

* IP of Host B
* MAC of Host B
* IP of Host A
* MAC of Host A

and will broadcast the packet on the network.

**Step C:** All hosts in the network will receive the data, if the mac of that specific IP is not in the cache, the hosts will store the mac IP relation.

**Step D:** Host A now has the MAC of Host B and they can communicate between them.

**CODE:**

from multiprocessing import Process

def query(ip\_list):

host = "127.0.0.255"

port = 8080

physical = "02:00:00:%02x:%02x:%02x" % (random.randint(0, 255),

random.randint(0, 255),

random.randint(0, 255))

UDPsock = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

UDPsock.bind((host, port))

cnt=0

while True:

ip = random.choice(ip\_list)

print(f"Wanting physical address of {ip}\n")

arp\_frame = [host, physical, ip, -1]

if cnt == 4:

arp\_frame[0] = -1

for i in ip\_list:

tup = (i, port)

UDPsock.sendto(pickle.dumps(arp\_frame), tup)

msg, addr = UDPsock.recvfrom(1024)

msg = pickle.loads(msg)

time.sleep(4)

print(f"Received physical address is {msg[3]}\n")

time.sleep(2)

cnt+=1

if cnt==5:

break

def answer(i):

host = f"127.0.0.{i}"

port = 8080

physical = "02:00:00:%02x:%02x:%02x" % (random.randint(0, 255),

random.randint(0, 255),

random.randint(0, 255))

print(f"Physical address of host {host} is {physical}\n")

while True:

UDPsock = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

UDPsock.bind((host, port))

msg, addr = UDPsock.recvfrom(1024)

msg = pickle.loads(msg)

if msg[2] == host:

msg[3] = physical

time.sleep(4)

UDPsock.sendto(pickle.dumps(msg), addr)

else:

print(f"Ip {host} dropped message")

time.sleep(4)

if msg[0] == -1:

break

if \_\_name\_\_ == '\_\_main\_\_':

n = int(input("Enter number of processes "))

print()

ip\_list = [f"127.0.0.{i}" for i in range(1,n+1)]

cq = Process(target=query,args=(ip\_list,))

ans = [Process(target=answer,args=(i,)) for i in range(1,n+1)]

for a in ans:

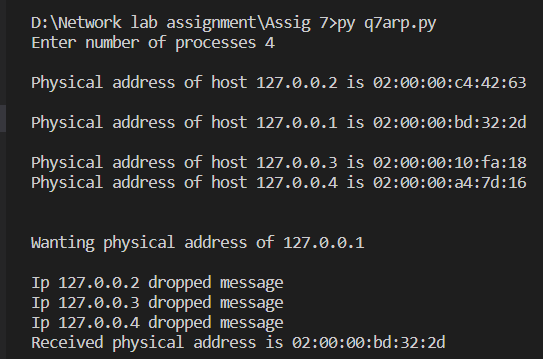
a.start()

cq.start()

for a in ans:

a.join()

cq.join()



**Conclusion:**

The implementation of ARP Protocol bind itself to all network interfaces available and request MAC by directly multicast the requests to the network and caching all incoming responses and all stations maintain their ARP Table.