Name: Tanmoy Sarkar
Roll No: 002010501020
Class: BCSE III
Assignment No: 7
Subject: Computer Network
Group: A1

ARP PROTOCOL

Full form of ARP is Address Resolution Protocol.

In a network, to communicate between two hosts, we need the MAC to address the NIC of the source and destination host. In a network initially, the hosts got a static IP or dynamic IP allocated by DHCP servers.

With only source and destination IPs, hosts can't communicate between them. Here the ARP protocol helps to translate the IP address to MAC.

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 Implementation -

```
from time import sleep
import socket
import random
import string
from threading import Thread

# Frame format
# IPSender:MACSender:ipRequested:macRequested

class Client:
    def __init__(self, ip, mac):
        # Set ip and mac of self
        self.ip = ip
        self.mac = mac
        # Create socket
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)
        # Enable port reusage
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
        # Enable broadcasting mode
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        # Bind to Client Port
        self.sock.bind(('', 1148))
        # Arp Table
        self.arp_table = {}

        print("Enter `arp` to show arp table")
        print("Enter IP to send ARP request")


    def startProcess(self):
        send_thread = Thread(target=self.sendMsg)
        receive_thread = Thread(target=self.receiveMsg)
        send_thread.start()
        receive_thread.start()
        send_thread.join()
        receive_thread.join()
        self.sock.close()
```

```
    def sendMsg(self):
        while True:
            ip = input()
            if ip == "arp":
                self.printArpTable()
            elif ip in self.arp_table:
                print("Cached MAC : " + self.arp_table[ip])
            else:
                self.sock.sendto((f"{self.ip}:{self.mac}:{ip}:").encode(),
('<broadcast>', 1148))

    def receiveMsg(self):
        while True:
            data, addr = self.sock.recvfrom(1024)
            msg = data.decode()
            msg_splitted = msg.split(":")

            ipSender = msg_splitted[0]
            macSender = msg_splitted[1]

            ipRequested = msg_splitted[2]
            macRequested = msg_splitted[3]

            # Store the sender ip and mac in arp table if not already stored
            if ipSender != self.ip:
                self.arp_table[ipSender] = macSender

            if ipRequested == self.ip:
                # Check whether the mac address is already present in frame
                if macRequested == "":

self.sock.sendto(f"{ipSender}:{macSender}:{self.ip}:{self.mac}".encode(),
('<broadcast>', 1148))
            elif macRequested not in self.arp_table:
                if macRequested != "":
                    self.arp_table[ipRequested] = macRequested

    def printArpTable(self):
        print("\nIP\tMAC")
        for ip, mac in self.arp_table.items():
            print(ip + "\t" + mac)
        print()


if __name__ == "__main__":
    # Create client
```

```
    ip = input("Enter IP: ")
    mac = input("Enter MAC (enter -1 for random mac): ")
    if mac == "-1":
        mac = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in
range(12))
    client = Client(ip, mac)
    client.startProcess()
```

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.

DHCP PROTOCOL

Full form of DHCP is Dynamic Host Configuration Protocol. It is based on the Bootstrap Protocol (BOOTP). This adds the capability of automatic allocation of reusable network addresses.

How does a station in the network get an IP address from DHCP Server?
There are various steps to acquire the IP.
Initially, the station will have an IP of 0.0.0.0

1. *DHCPDISCOVER*: Any station that needs IP, neither knows about the existence of the DHCP Server nor the IP of the DHCP Server. So the station will broadcast DHCPDISCOVER packet to the network.
2. *DHCPOFFER*: The DHCP Server will catch the DHCPDISCOVER packet that was relayed by the station. The DHCP server will offer the station an IP to acquire and will broadcast in the network.
3. *DHCPREQUEST* : The station will receive the DHCPOFFER and if everything looks fine, will request DHCP Server to assign the IP.
4. *DHCPACK:* After receiving DHCPREQUEST, the DHCP server will check again the availability of the specified IP and if available will send DHCPACK mentioning the confirmed IP address and leased time.
5. *DHCPNAK*: If IP in DHCPREQUEST is not available, DHCP Server will send *DHCPNAK*

After receiving DHCPACK, the station has got an IP address for a specific leased time. After ending of the leased time station will request the DHCP server for reallotment of the IP by the DHCPREQUEST packet.

Station and DHCP server decide the type of packet by a byte in packet, `message type`

Packet Name	Message Type
DHCPDISCOVER	1
DHCPOFFER	2
DHCPREQUEST	3

DHCPACK	5
DHCPNAK	6

Code Implementation -

Helper class [Frame] to encode and decode frame

```
# Frame format
# Request Type [1-> request, 2-> reply] | Message Type | Transaction ID | Client MAC
Address | Client IP Address | Lease Time | Subnet Mask | Gateway IP | DNS IP | Host
Name

class Frame:
    def __init__(self):
        self.requestType = ""
        self.messageType = ""
        self.transactionID = ""
        self.clientMAC = ""
        self.clientIP = ""
        self.leaseTime = ""
        self.subnetMask = ""
        self.gatewayIP = ""
        self.dnsIP = ""
        self.hostName = ""

    @staticmethod
    def decode(frameString):
        frame = Frame()
        frameSplitted = frameString.split("|")
        frame.requestType = frameSplitted[0]
        frame.messageType = frameSplitted[1]
        frame.transactionID = frameSplitted[2]
        frame.clientMAC = frameSplitted[3]
        frame.clientIP = frameSplitted[4]
        frame.leaseTime = frameSplitted[5]
        frame.subnetMask = frameSplitted[6]
        frame.gatewayIP = frameSplitted[7]
        frame.dnsIP = frameSplitted[8]
        frame.hostName = frameSplitted[9]

        return frame

    def encode(self):
        return str(self.requestType) + "|" + str(self.messageType) + "|" +
str(self.transactionID) + "|" + str(self.clientMAC) + "|" + str(self.clientIP) + "|" +
```

```
str(self.leaseTime) + "|" + str(self.subnetMask) + "|" + str(self.gatewayIP) + "|" +
str(self.dnsIP) + "|" + str(self.hostName)


if __name__ == "__main__":
    frame = Frame()
    s = frame.encode()
    print(s)
    x = s.split("|")
    print(x)
    print(len(x))
```

Code for DHCP Client [Station in network]

```
from time import sleep
import socket
import random
import string
from frame import Frame
from threading import Thread

class DhcpClient:
    def __init__(self, mac, host_name, dhcp_client_port=1168, dhcp_server_port=1167):
        self.mac = mac
        self.ip = '0.0.0.0'
        self.subnet_mask = '0.0.0.0'
        self.dns_ip = '0.0.0.0'
        self.gateway_ip = '0.0.0.0'
        self.lease_time = 0
        self.host_name = host_name
        self.dhcp_client_port = dhcp_client_port
        self.dhcp_server_port = dhcp_server_port

        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)
        # Enable port reusage
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
        # Enable broadcasting mode
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

        # Bind to Client Port
        self.sock.bind(('', self.dhcp_client_port))

        # Transaction ID
        self.transacionID = "-1"
```

```
    def broadcast(self, message):
        self.sock.sendto(message.encode(), ("<broadcast>", self.dhcp_server_port))

    def receiveMsg(self):
        data, addr = self.sock.recvfrom(2048)
        return data.decode()

    def startProcess(self):
        # Generate transaction id
        self.transacionID = ''.join(random.choices(string.ascii_lowercase +
string.digits, k=8))
        receive_thread = Thread(target=self.receive_data)
        receive_thread.start()
        sleep(0.01)
        self.send_dhcp_discover()
        receive_thread.join()



    def send_dhcp_discover(self):
        # Send DHCP Discover
        frame = Frame()
        frame.requestType = 1
        frame.messageType = 1
        frame.transactionID = self.transacionID
        frame.clientMAC = self.mac
        frame.clientIP = '0.0.0.0'
        frame.hostName = self.host_name
        self.broadcast(frame.encode())

    def send_dhcp_request(self):
        frame = Frame()
        frame.requestType = 1
        frame.messageType = 3
        frame.transactionID = self.transacionID
        frame.clientMAC = self.mac
        frame.clientIP = self.ip
        frame.hostName = self.host_name
        self.broadcast(frame.encode())

    def receive_data(self):
        while True:
            receieved_frame_raw = self.receiveMsg()
            receieved_frame = Frame.decode(receieved_frame_raw)
            # DHCP OFFER
            if receieved_frame.requestType == "2" and receieved_frame.transactionID ==
```

```
self.transacionID and receieved_frame.messageType == "2" and receieved_frame.clientMAC
== self.mac:
                self.ip = receieved_frame.clientIP
                # Send DHCP Request
                self.send_dhcp_request()

            # DHCP ACK
            if receieved_frame.requestType == "2" and receieved_frame.transactionID ==
self.transacionID  and receieved_frame.clientMAC == self.mac:
                if receieved_frame.messageType == "5":
                    self.ip = receieved_frame.clientIP
                    self.subnet_mask = receieved_frame.subnetMask
                    self.gateway_ip = receieved_frame.gatewayIP
                    self.dns_ip = receieved_frame.dnsIP
                    self.lease_time = int(receieved_frame.leaseTime)
                    self.printDetails()
                    lease_thread = Thread(target=self.handleLeaseTime)
                    lease_thread.start()
                elif receieved_frame.messageType == "6":
                    # NACK
                    print("DHCP NAK received from DHCP server. Listening again ...")


    def handleLeaseTime(self):
        sleep(self.lease_time+0.1)
        print("Lease time expired. Listening again ...")
        self.send_dhcp_request()

    def printDetails(self):
        print("-----------------------------------------------")
        print("Host        : ", self.host_name)
        print("IP          : ", self.ip)
        print("MAC         : ", self.mac)
        print("Subnet mask : ", self.subnet_mask)
        print("Gateway IP  : ", self.gateway_ip)
        print("DNS         : ", self.dns_ip)
        print("Lease time  : ", self.lease_time)
        print("-----------------------------------------------")


    def close(self):
        self.sock.close()


# Steps
# 1. DHCP Discover
```

```
# 2. DHCP Request

if __name__ == "__main__":
   mac_id = "02:00:00:%02x:%02x:%02x" % (random.randint(0, 255),random.randint(0,
255), random.randint(0, 255))
   print("MAC ID: ", mac_id)
   client_name = "client-" + mac_id
   client = DhcpClient(mac_id, client_name)
   client.startProcess()
   client.close()
```

Code for DHCP Server

```
from re import S
import socket
import random
import string
from threading import Thread
from time import sleep
from turtle import pen
from frame import Frame

class DhcpServer:
    def __init__(self, mac, starting_ip="192.168.0.2", subnet_mask="255.255.255.0",
dhcp_client_port=1168, dhcp_server_port=1167):
        self.mac = mac
        self.dhcp_client_port = dhcp_client_port
        self.dhcp_server_port = dhcp_server_port

        self.dns_ip = "8.8.8.8"
        self.subnet_mask = subnet_mask
        self.gateway_ip = "192.168.0.1"

        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)
        # Enable port reusage
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
        # Enable broadcasting mode
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

        # Bind to Client Port
        self.sock.bind(('', self.dhcp_server_port))

        self.starting_ip = [int(x) for x in starting_ip.split(".")]
        self.subnet_mask = [int(x) for x in subnet_mask.split(".")]

        # Transaction ID for DHCP
        self.transaction_id_offerred_ip = {}
        self.available_ips = set()

        # Lease time table
        self.lease_time_table = {}


    def broadcast(self, message):
        self.sock.sendto(message.encode(), ("<broadcast>", self.dhcp_client_port))

    def receiveMsg(self):
```

```
        data, addr = self.sock.recvfrom(1024)
        return data.decode()

    def startProcess(self):
        expiry_thread = Thread(target=self.timeoutExpiry)
        expiry_thread.start()
        while True:
            request_frame = Frame.decode(self.receiveMsg())
            if request_frame.requestType == "1":
                if request_frame.messageType == "1":
                    # Wait for DHCP Discover request
                    print("DHCP Discover request received from ",
request_frame.clientMAC)
                    # Generate IP
                    ip = self.generateNextIP()
                    self.transaction_id_offerred_ip[request_frame.transactionID] = ip

                    # DHCP OFFER
                    offer = Frame()
                    offer.requestType = "2"
                    offer.messageType = "2"
                    offer.clientMAC = request_frame.clientMAC
                    offer.clientIP = ip
                    offer.transactionID = request_frame.transactionID
                    offer.hostName = request_frame.hostName
                    # sleep(2)
                    print("Sending DHCP Offer to ", request_frame.clientMAC)
                    self.broadcast(offer.encode())

                    # Wait for DHCP Request
                    request = Frame.decode(self.receiveMsg())
                    if request.requestType == "1" and request.messageType == "3":
                        print("DHCP Request received from ", request.clientMAC)
                        # sleep(2)
                        # Check if transaction id is same
                        if request.transactionID in self.transaction_id_offerred_ip:
                            # Check if IP is same
                            if self.transaction_id_offerred_ip[request.transactionID]
== request.clientIP and (request.clientIP in self.available_ips or request.clientIP
not in self.lease_time_table):
                                # DHCP ACK
                                ack = Frame()
                                ack.requestType = "2"
                                ack.messageType = "5"
                                ack.clientMAC = request.clientMAC
                                ack.clientIP = request.clientIP
```

```
                                ack.hostName = request.hostName
                                ack.transactionID = request.transactionID
                                ack.dnsIP = self.dns_ip
                                ack.subnetMask = '.'.join([str(x) for x in
self.subnet_mask])
                                ack.gatewayIP = self.gateway_ip
                                ack.leaseTime = "10"
                                self.broadcast(ack.encode())
                                # Delete from available ips
                                if request.clientIP in self.available_ips:
                                    self.available_ips.remove(request.clientIP)
                                self.lease_time_table[request.clientIP] = 10
                                print("DHCP ACK sent to ", request.clientMAC)
                            else:
                                print("IP address not same as offerred")
                        else:
                            print("Transaction ID not same as offerred")
                elif request_frame.messageType == "3":
                    # DHCP ACK
                    ack = Frame()
                    ack.requestType = "2"
                    ack.messageType = "5"
                    ack.clientMAC = request.clientMAC
                    ack.clientIP = request.clientIP
                    ack.hostName = request.hostName
                    ack.transactionID = request.transactionID
                    ack.dnsIP = self.dns_ip
                    ack.subnetMask = '.'.join([str(x) for x in self.subnet_mask])
                    ack.gatewayIP = self.gateway_ip
                    ack.leaseTime = "10"
                    self.broadcast(ack.encode())
                    # Delete from available ips
                    if request.clientIP in self.available_ips:
                        self.available_ips.remove(request.clientIP)
                    self.lease_time_table[request.clientIP] = 10

    def timeoutExpiry(self):
        while True:
            ips_need_to_be_removed = []
            for ip, time in self.lease_time_table.items():
                if time <= 0:
                    ips_need_to_be_removed.append(ip)
                    print("IP address released ", ip)
                else:
                    self.lease_time_table[ip] -= 1
```

```
            for ip in ips_need_to_be_removed:
                del self.lease_time_table[ip]
                self.available_ips.add(ip)
            sleep(1)

    def generateIP(self):
        return ".".join([str(x) for x in self.starting_ip])

    def generateNextIP(self):
        # TODO do according to subnet mask
        self.starting_ip[3] += 1
        return ".".join([str(x) for x in self.starting_ip])

    def close(self):
        self.sock.close()


if __name__ == "__main__":
    server = DhcpServer("02:02:01:02:02:00")
    server.startProcess()
    server.close()
```

Conclusion:
Basic features of DHCP protocol has been implemented and it can provide stations in the network an IP address. Some advanced packet format like DHCPREALEASE, DHCPINFORM, and DHCPDECLINE is not implemented for this scope.