Computer Networks Lab Report- Assignment 7

TITLE:

Name: Debarghya Maitra
Class: BCSE 3 rd Year
Group: A3
Submission Date: 4/11/2022

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

1. ARP
2. BOOTP
3. DHCP

DESIGN:

From the above mentioned 3 protocols I have chosen to implement BOOTP and DHCP using UDP sockets.

1) BOOTP: The Bootstrap Protocol(BOOTP) is a computer networking protocol used in Internet Protocol networks to automatically assign an IP address to network devices from a configuration server. The server assigns the following to a newly added network device in a particular network:
i) The client's IP address, subnet mask, and default gateway address; ii)The IP address and host name of the BOOTP server; iii)The IP address of the server that has the boot image, which the client needs to load its operating system.

In this assignment, the above mentioned functionality is implemented with the following methods:

i) server: The server listens at any mentioned interface(0.0.0.0) for all network interfaces for requests of IP, subnet mask, etc on a UDP socket. After a valid request, the server responds with all the requirements of the device.

ii) client: The client multicasts requests for its IP address, subnet mask etc to any mentioned interface (255.255.255.255) for broadcasting to all interfaces; but currently it supports multicasting to only one interface(with maximum 254 nodes) at a time via UDP sockets. The client also sends the previously allocated IP address for it, if the server finds that IP address isn't yet taken, the client gets the requested IP address, else it gets declined and the client makes a request for another IP address.

iii) calculateSubnetMask: Calculates subnet mask for mentioned number of devices(max supported for now is 254)

2) DHCP: Dynamic Host Configuration Protocol is just an extension of the BOOTP and it provides automatic configuration of the network interface of the requesting device, alongwith some of the important parameters like DNS server, timeout period(how long the IP address is valid for the device). So for demonstration purposes, DHCP is implemented quite similar to BOOTP with added points like DNS server address(kept constant as google DNS server i.e 8.8.8.8) and time out time(also kept standard as 30 mins).

IMPLEMENTATION:

1) server, client for BOOTP:

```
def calculateSubnetMaskv4(n: int) -> str:
```

```
    x = nextPowerOf2(n)
    mask = "1"*(32-x) + "0"*x
    subnet = [str(int(mask[i:i+8], 2)) for i in range(0, 32, 8)]
    return ".".join(subnet)
def server(interface, port, ndevices):
    taken_ips = []
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((interface, port))
    #print(f"Current subnet mask is (for {ndevices}):", calculateSubnetMaskv4(ndevices))
    print(f"Listening for requests at {sock.getsockname()}")
    while True:
        data, address = sock.recvfrom(BUFSIZE)
        text = data.decode('ascii')
        print(f"The client at ('0.0.0.0', {address[1]}) is: {text}")
        s = text.split(":")[1]
        if s in taken_ips:
            sock.sendto(b"Declined Request", address)
            print("Declined Request")
        else:
            taken_ips.append(s)
                    sock.sendto(f"{s},{calculateSubnetMaskv4(ndevices)},{interface}".encode("ascii"),
address)
def client(network, port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    addr = ("192.168.101.6", 40400)
    sock.bind(addr)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 0)
    text = f"Requesting for IP, subnet mask, default gateway, previous IP:{addr[0]}"
    s = network.split(".")
    netw_list = [".".join(s[:3])+"."+str(int(i)) for i in range(256)]
    print("Requested for private IP, subnet-mask, gateway")
    for netw in netw_list:
        try:
            sock.sendto(text.encode("ascii"), (netw, port))
        except Exception:
```

```
            pass
    prev = addr[0]
    while True:
        random.seed(time())
        data, address = sock.recvfrom(BUFSIZE)
        text = data.decode('ascii')
        if "Declined" in text:
            print(text)
            print("Requesting again...")
            l = prev.split(".")
            new = ".".join(l[:3]) + "." + str(random.randint(1, 254))
            prev = new
            sock.sendto(f"Requesting for IP:{new}".encode('ascii'), address)
        else:
            print("Reply from BOOTP server:")
            addrs = text.split(",")
            print(f"Assigned IP: {addrs[0]}")
            print(f"Subnet Mask of network: {addrs[1]}")
            print(f"Default gateway: {address[0]}")
            sock.close()
            break
```

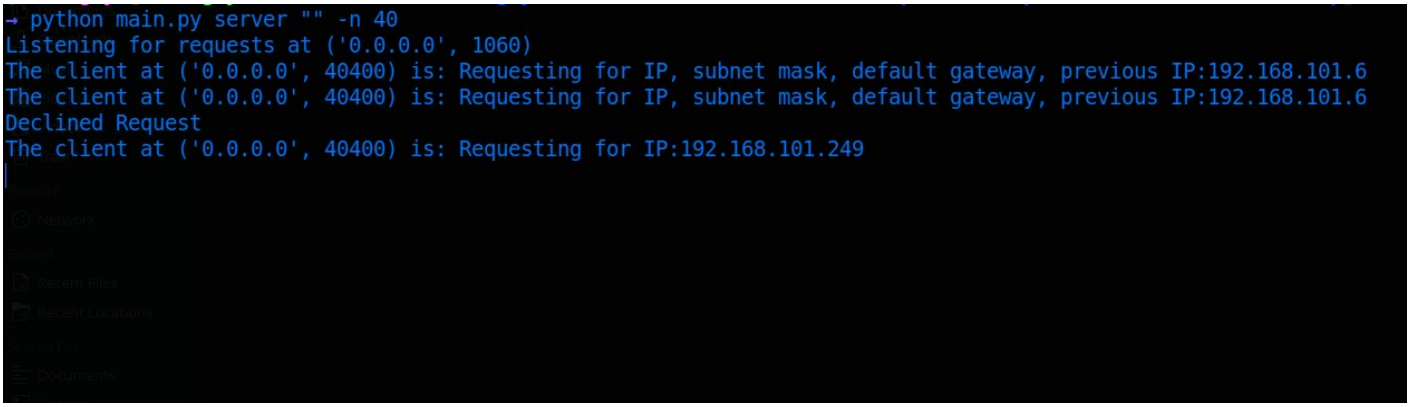
2) server, client for DHCP:

```
def server(interface, port, ndevices):
    taken_ips = []
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((interface, port))
    #print(f"Current subnet mask is (for {ndevices}):", calculateSubnetMaskv4(ndevices))
    print(f"Listening for requests at {sock.getsockname()}")
    while True:
        data, address = sock.recvfrom(BUFSIZE)
        text = data.decode('ascii')
        print(f"The client at ('255.255.255.255', {address[1]}) is: {text}")
        s = text.split(":")[1]
        if s in taken_ips:
            sock.sendto(b"Declined Request", address)
            print("Declined Request")
        else:
            taken_ips.append(s)
                sock.sendto(f"{s},{calculateSubnetMaskv4(ndevices)},8.8.8.8, 30mins".encode("ascii"),
address)
```

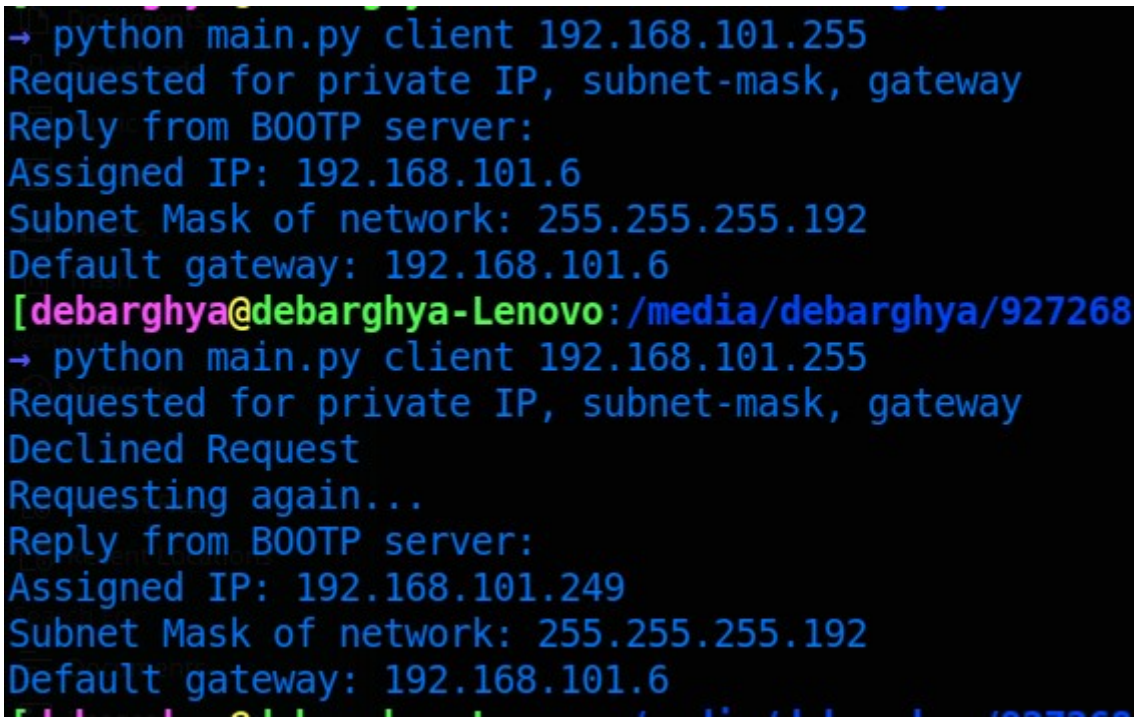
```
def client(network, port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    addr = ("192.168.101.6", 40400)
    sock.bind(addr)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 0)
      text = f"Requesting for IP, subnet mask, default gateway, DNS server, Timeout, previous IP: {addr[0]}"
    s = network.split(".")
    netw_list = [".".join(s[:3])+"."+str(int(i)) for i in range(256)]
    print("Broadcasted for private IP, subnet-mask, gateway(Discover)")
    for netw in netw_list:
        try:
            sock.sendto(text.encode("ascii"), (netw, port))
        except Exception:
            pass
    prev = addr[0]
    while True:
        random.seed(time())
        data, address = sock.recvfrom(BUFSIZE)
        text = data.decode('ascii')
        if "Declined" in text:
            print(text)
            print("Requesting again...")
            l = prev.split(".")
            new = ".".join(l[:3]) + "." + str(random.randint(1, 254))
            prev = new
            sock.sendto(f"Requesting for IP:{new}".encode('ascii'), address)
        else:
            print(f"Offer from DHCP server({address}):")
            addrs = text.split(",")
            print(f"Assigned IP: {addrs[0]}")
            print(f"Subnet Mask of network: {addrs[1]}")
            print(f"Default gateway: {address[0]}")
            print(f"DNS server: {addrs[2]}")
            print(f"Time out for the offer: {addrs[3]}")
            sock.close()
            break
```

RESULTS:

1) BOOTP:

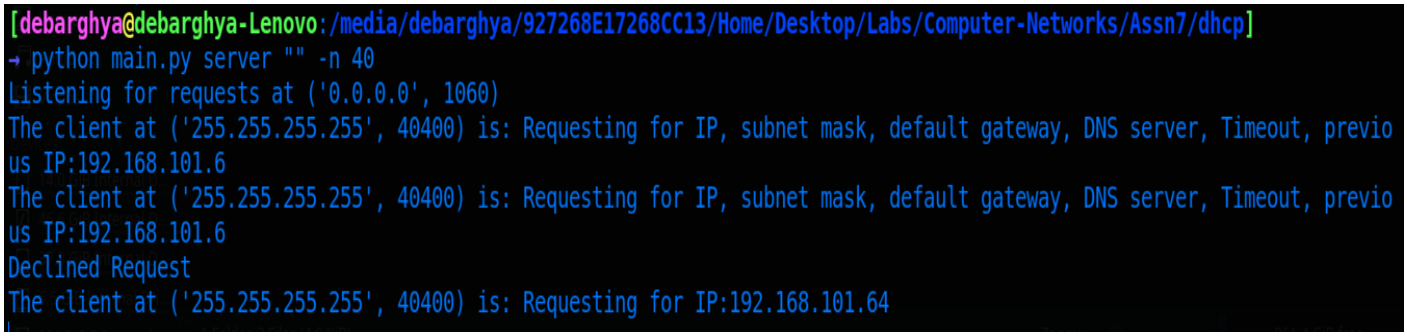


server logs for 40 connected devices

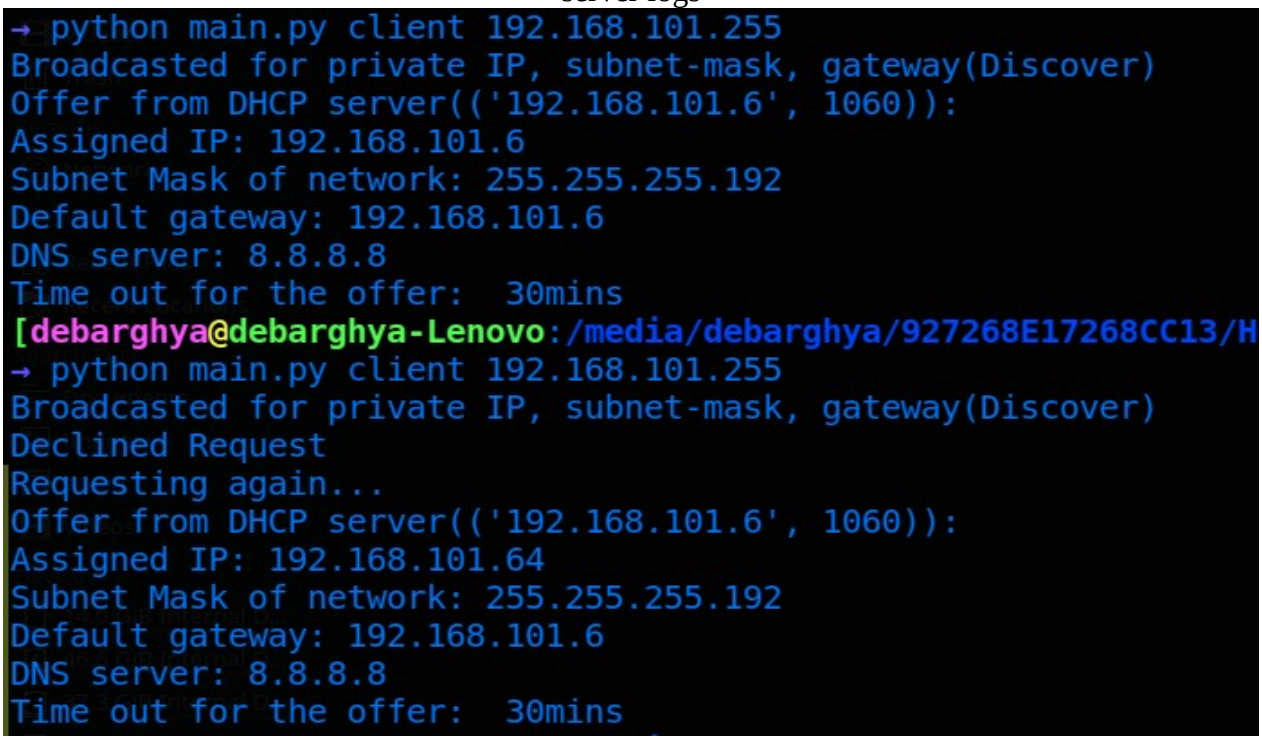


client request logs for 2 different devices

2)DHCP:



server logs



ANALYSIS:

In actual use cases, both BOOTP and DHCP works as follows:
i)Client broadcasts its request to all nodes connected to the network and the server accepts the request and unicasts the requested things to the client. Here for demonstration the client multicasts only on the particular interface.
ii)Also the default gateway may or may not be the same as the server, here its implemented as same.
iii)All the requests and responses are transferred with the help of packets of particular format in actual situations, here its done directly by sending raw data to the sockets.
iv)The DHCP client loses its identity after the timeout time and requests the server again for an IP, here that functionality isn't implemented yet.
v)Though the above results show the demonstration on a single system, the implementation has been tested on both docker container environment and real systems connected to same network, where all the results are quite similar.
vi)If a requested IP address is not available at the moment, the server must check its available IP list and assign the client accordingly, but here the client again requests again with a randomly generated IP which is within range, this can be improved.

COMMENTS

This assignment really helped me understand how BOOTP and DHCP works and how new devices get their identities in the network they are connecting to. I also learnt how BOOTP slowly became deprecated and DHCP evolved and currently all devices use DHCP for network configuration.
Finally, I'd mention though I hadn't implemented ARP, but learnt a lot about the protocol while trying to simulate it, I couldn't figure out why should we implement a network layer protocol with TCP/UDP sockets, it doesn't make sense.