Lab 4

Computer Networks Lab (CS302)

Team Members:

Alan Tony (191CS207)

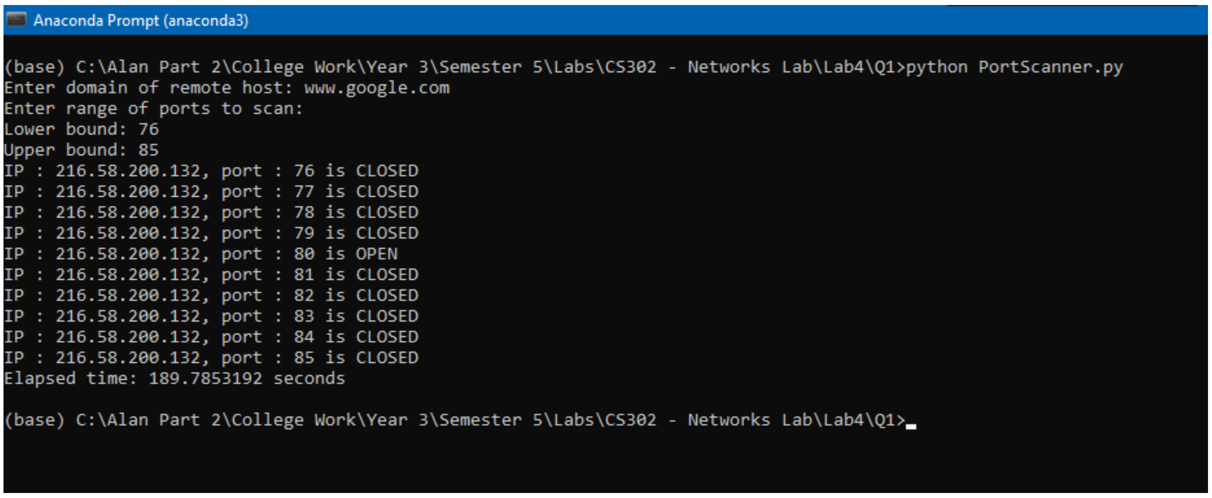
Aashish Prateek (191CS225)

Question 1

Code:

```python
import socket
from timeit import default_timer as timer

def port_scan(server_IP, port_num):

    #We need multiple scokets for different outgoing connections, otherwise we
get the error:
    #  "A connect request was made on an already connected socket"
    #Therefore each port_scan() call creates a new socket.
    # Instead of binding, we let the OS assign an ephemeral port
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    #Try to connect
    try:
        s.connect((server_IP, port_num))
        s.close()
        return True

    except:
        return False


if __name__ == '__main__':

    domain = input("Enter domain of remote host: ")
    if not len(domain):
        domain = 'www.google.com'
    # Get IP address of domain
    server_IP = socket.gethostbyname(domain)

    # input range of ports
    print("Enter range of ports to scan: ")
    lo = max(int(input("Lower bound: ")), 1)
    up = max(int(input("Upper bound: ")), lo)

    start_time = timer()
    for port_num in range(lo, up+1):

        if port_scan(server_IP, port_num):
            print(f"IP : {server_IP}, port : {port_num} is OPEN")
        else:
            print(f"IP : {server_IP}, port : {port_num} is CLOSED")
    end_time = timer()

    print(f"Elapsed time: {end_time - start_time} seconds")
```

Output:



Anaconda Prompt (anaconda3)

```
(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab4\Q1>python PortScanner.py
Enter domain of remote host: www.google.com
Enter range of ports to scan:
Lower bound: 76
Upper bound: 85
IP : 216.58.200.132, port : 76 is CLOSED
IP : 216.58.200.132, port : 77 is CLOSED
IP : 216.58.200.132, port : 78 is CLOSED
IP : 216.58.200.132, port : 79 is CLOSED
IP : 216.58.200.132, port : 80 is OPEN
IP : 216.58.200.132, port : 81 is CLOSED
IP : 216.58.200.132, port : 82 is CLOSED
IP : 216.58.200.132, port : 83 is CLOSED
IP : 216.58.200.132, port : 84 is CLOSED
IP : 216.58.200.132, port : 85 is CLOSED
Elapsed time: 189.7853192 seconds

(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab4\Q1>_
```

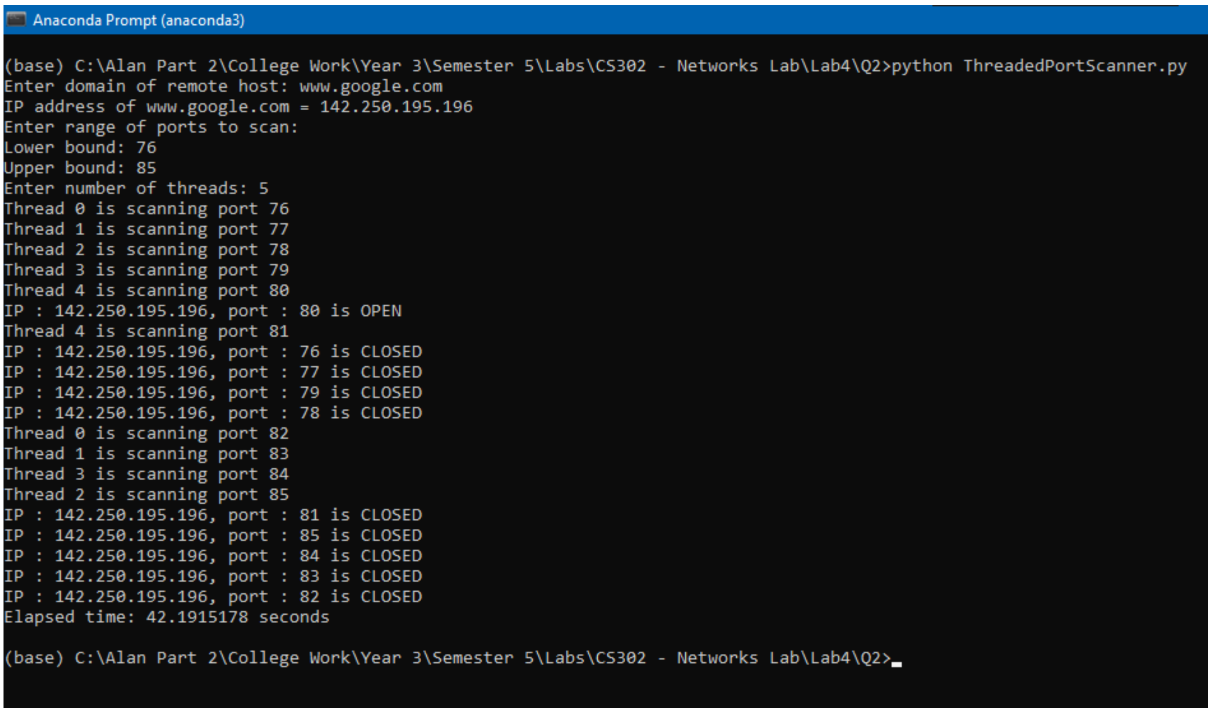
P.T.O

Question 2

Code:

```python
import threading
import socket
from timeit import default_timer as timer

lock = threading.Lock()
curr_port = hi = lo = 1

def port_scan(i):

    global curr_port, up, lo
    while True :

        #print(f"Entered thread {i}")

        # Get next port to scan
        lock.acquire()
        if curr_port > up:
            lock.release()
            break
        port_num = curr_port
        curr_port += 1
        lock.release()

        print(f"Thread {i} is scanning port {port_num}")

        #We need multiple scokets for different outgoing connections,
otherwise we get the error:
        #  "A connect request was made on an already connected socket"
        #Therefore each port_scan() call creates a new socket.
        # Instead of binding, we let the OS assign an ephemeral port
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        #Try to connect
        try:
            s.connect((server_IP, port_num))
            s.close()
            print(f"IP : {server_IP}, port : {port_num} is OPEN")

        except:
            print(f"IP : {server_IP}, port : {port_num} is CLOSED")


if __name__ == '__main__':

    domain = input("Enter domain of remote host: ")
    if not len(domain):
        domain = 'www.google.com'
    # Get IP address of domain
    server_IP = socket.gethostbyname(domain)
```

Output:



```
(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab4\Q2>python ThreadedPortScanner.py
Enter domain of remote host: www.google.com
IP address of www.google.com = 142.250.195.196
Enter range of ports to scan:
Lower bound: 76
Upper bound: 85
Enter number of threads: 5
Thread 0 is scanning port 76
Thread 1 is scanning port 77
Thread 2 is scanning port 78
Thread 3 is scanning port 79
Thread 4 is scanning port 80
IP : 142.250.195.196, port : 80 is OPEN
Thread 4 is scanning port 81
IP : 142.250.195.196, port : 76 is CLOSED
IP : 142.250.195.196, port : 77 is CLOSED
IP : 142.250.195.196, port : 79 is CLOSED
IP : 142.250.195.196, port : 78 is CLOSED
Thread 0 is scanning port 82
Thread 1 is scanning port 83
Thread 3 is scanning port 84
Thread 2 is scanning port 85
IP : 142.250.195.196, port : 81 is CLOSED
IP : 142.250.195.196, port : 85 is CLOSED
IP : 142.250.195.196, port : 84 is CLOSED
IP : 142.250.195.196, port : 83 is CLOSED
IP : 142.250.195.196, port : 82 is CLOSED
Elapsed time: 42.1915178 seconds

(base) C:\Alan Part 2\College Work\Year 3\Semester 5\Labs\CS302 - Networks Lab\Lab4\Q2>
```

The non-threaded solution (in Q1) took 109s whereas the threaded solution took only 42s for the same task. This shows the improved performance in scanning for multiple ports using multi-threading.

Question 3

To capture DNS traffic, the following steps were followed:

1. Start a Wireshark capture.
2. Commands entered on the Command Prompt:
   - ***i.*** ***ipconfig /renew***
     (Renews DHCP assigned IP address)
   - ***ii.*** ***ipconfig /flushdns***
     (Clears DNS cache)
   - ***iii.*** ***nslookup 8.8.8.8***
     (Looks up the hostname for the specified IP address)
3. Stop the Wireshark capture.

Part a) Analyze DHCP traffic

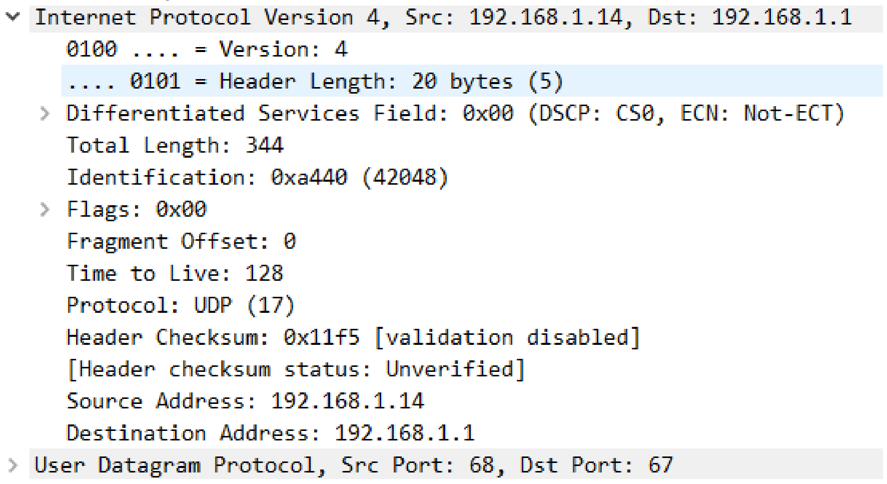
To analyze the DHCP traffic produced in the steps followed for DNS capture, we applied a filter **udp.port == 68.** We can use port 67 also, because port 68 is a destination port whereas port 67 is a source port. We will see 2 packets filtered out which are of types **DHCP Request** and **DHCP ACK** respectively.

Observations:

a. In the **Ethernet II layer,** MAC address of the destination in the **DHCP request packet** matches the one obtained from the *Physical Address* field after entering the command *ipconfig /all.*

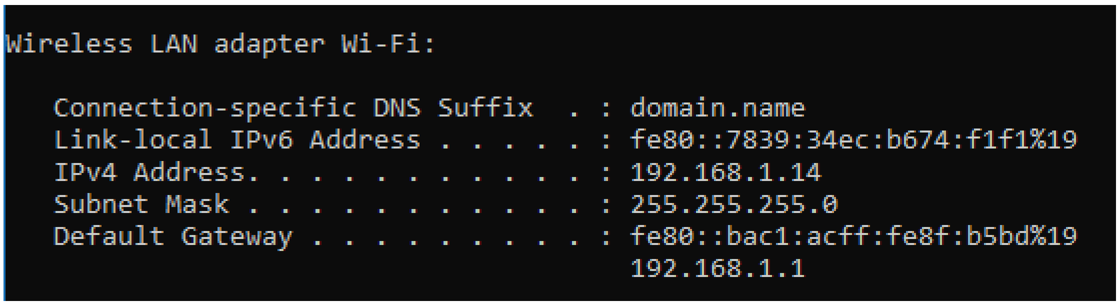
b. The next layer is **Internet Protocol (IPV4)**. From here we can obtain the IP address of the client and server.

In DHCP Request Packet:



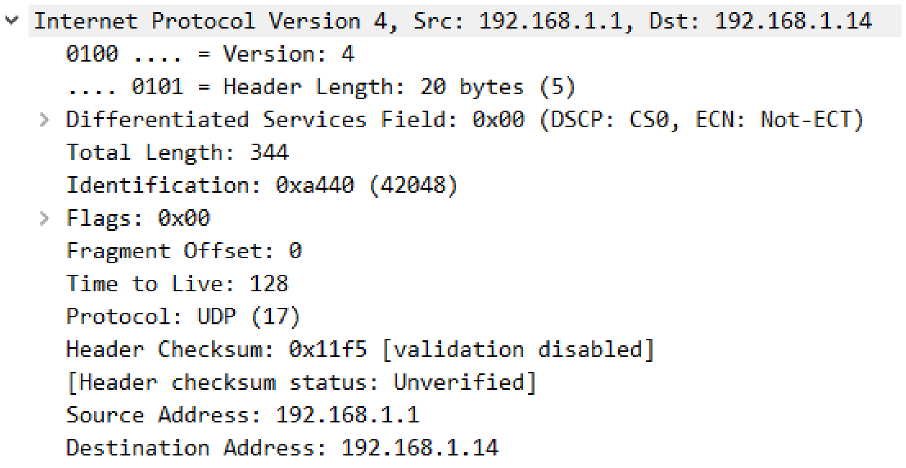
```
v Internet Protocol Version 4, Src: 192.168.1.14, Dst: 192.168.1.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 344
    Identification: 0xa440 (42048)
  > Flags: 0x00
    Fragment Offset: 0
    Time to Live: 128
    Protocol: UDP (17)
    Header Checksum: 0x11f5 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.14
    Destination Address: 192.168.1.1
> User Datagram Protocol, Src Port: 68, Dst Port: 67
```

Our IP can be confirmed with the *ipconfig* command:



```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : domain.name
   Link-local IPv6 Address . . . . . : fe80::7839:34ec:b674:f1f1%19
   IPv4 Address. . . . . . . . . . . : 192.168.1.14
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::bac1:acff:fe8f:b5bd%19
                                       192.168.1.1
```

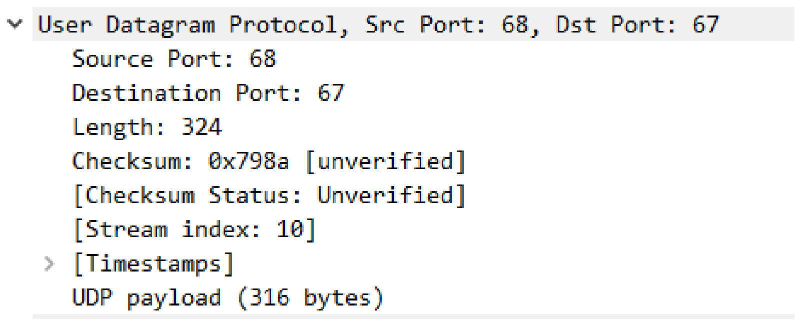
In DHCP ACK Packet:



```
v Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.14
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 344
    Identification: 0xa440 (42048)
  > Flags: 0x00
    Fragment Offset: 0
    Time to Live: 128
    Protocol: UDP (17)
    Header Checksum: 0x11f5 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.1
    Destination Address: 192.168.1.14
```

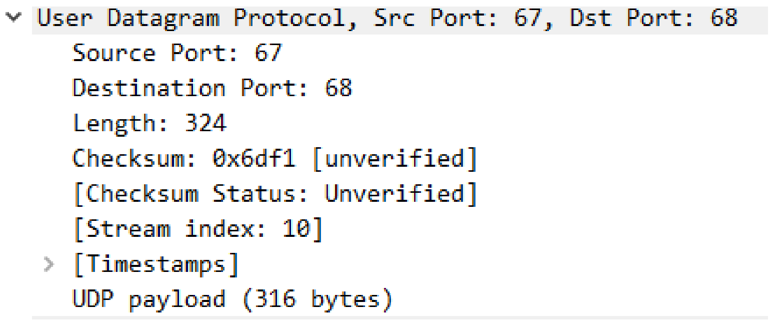
c. The next layer is **UDP.**

In DHCP Request Packet:



```
v User Datagram Protocol, Src Port: 68, Dst Port: 67
    Source Port: 68
    Destination Port: 67
    Length: 324
    Checksum: 0x798a [unverified]
    [Checksum Status: Unverified]
    [Stream index: 10]
  > [Timestamps]
    UDP payload (316 bytes)
```

In DHCP ACK Packet:



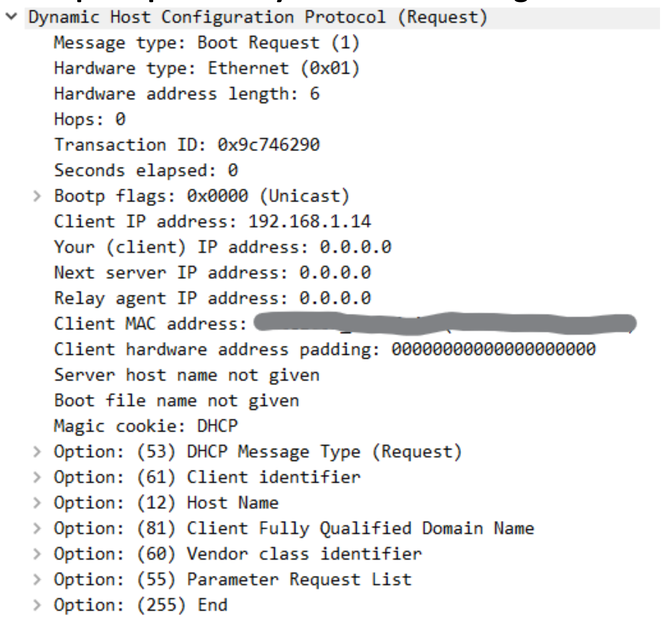
```
v User Datagram Protocol, Src Port: 67, Dst Port: 68
    Source Port: 67
    Destination Port: 68
    Length: 324
    Checksum: 0x6df1 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 10]
  > [Timestamps]
    UDP payload (316 bytes)
```

As we can see, in the DHCP Request, Source is 68 (Source port) as client sends this packet and Destination is port 67 (Destination port) and vice versa for DHCP ACK packet (Response form server).

We can also see the checksum field, which is used to check if packets are received correctly.

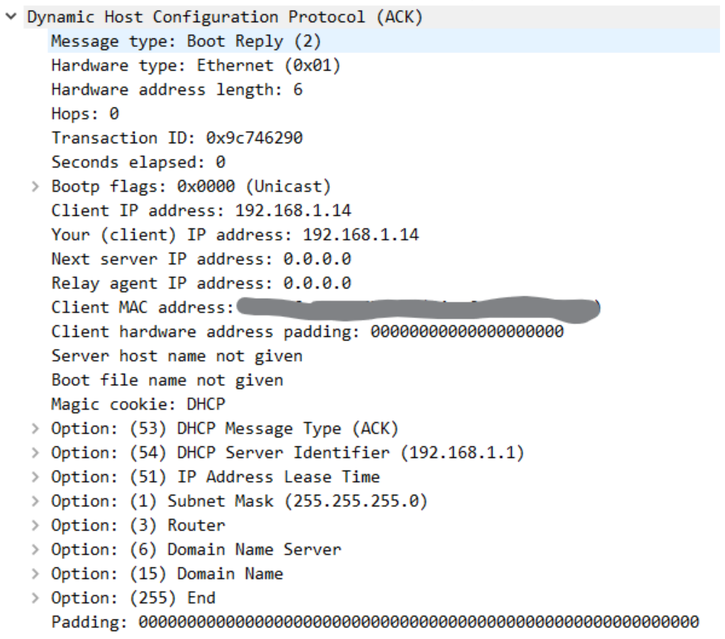
d. The next layer is different for DHCP Request and DHCP ACK packets:

In request packet: Dynamic Host Configuration Protocol (DHCP)



```
v Dynamic Host Configuration Protocol (Request)
    Message type: Boot Request (1)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x9c746290
    Seconds elapsed: 0
  > Bootp flags: 0x0000 (Unicast)
    Client IP address: 192.168.1.14
    Your (client) IP address: 0.0.0.0
    Next server IP address: 0.0.0.0
    Relay agent IP address: 0.0.0.0
    Client MAC address: [redacted]
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  > Option: (53) DHCP Message Type (Request)
  > Option: (61) Client identifier
  > Option: (12) Host Name
  > Option: (81) Client Fully Qualified Domain Name
  > Option: (60) Vendor class identifier
  > Option: (55) Parameter Request List
  > Option: (255) End
```

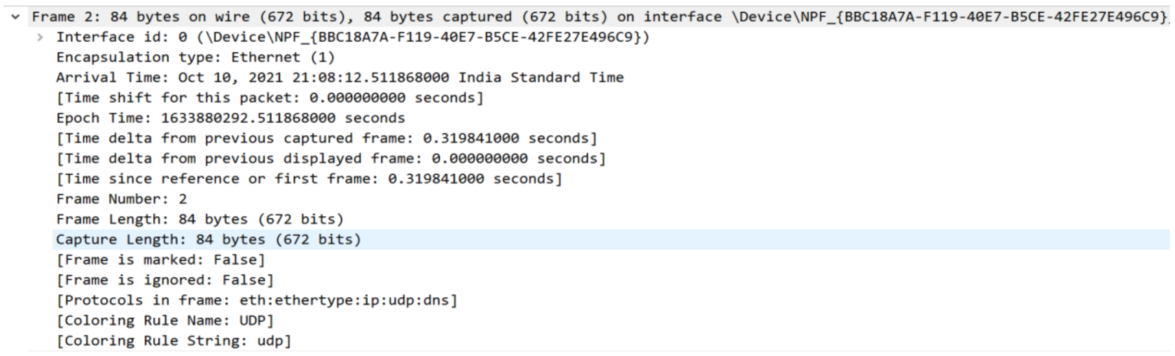
In ACK Packet: Dynamic Host Configuration Protocol (ACK)



```
⌄ Dynamic Host Configuration Protocol (ACK)
    Message type: Boot Reply (2)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0x9c746290
    Seconds elapsed: 0
  › Bootp flags: 0x0000 (Unicast)
    Client IP address: 192.168.1.14
    Your (client) IP address: 192.168.1.14
    Next server IP address: 0.0.0.0
    Relay agent IP address: 0.0.0.0
    Client MAC address: [redacted]
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  › Option: (53) DHCP Message Type (ACK)
  › Option: (54) DHCP Server Identifier (192.168.1.1)
  › Option: (51) IP Address Lease Time
  › Option: (1) Subnet Mask (255.255.255.0)
  › Option: (3) Router
  › Option: (6) Domain Name Server
  › Option: (15) Domain Name
  › Option: (255) End
    Padding: 0000000000000000000000000000000000000000000000000000000000
```

Part b) Analyze DNS Traffic

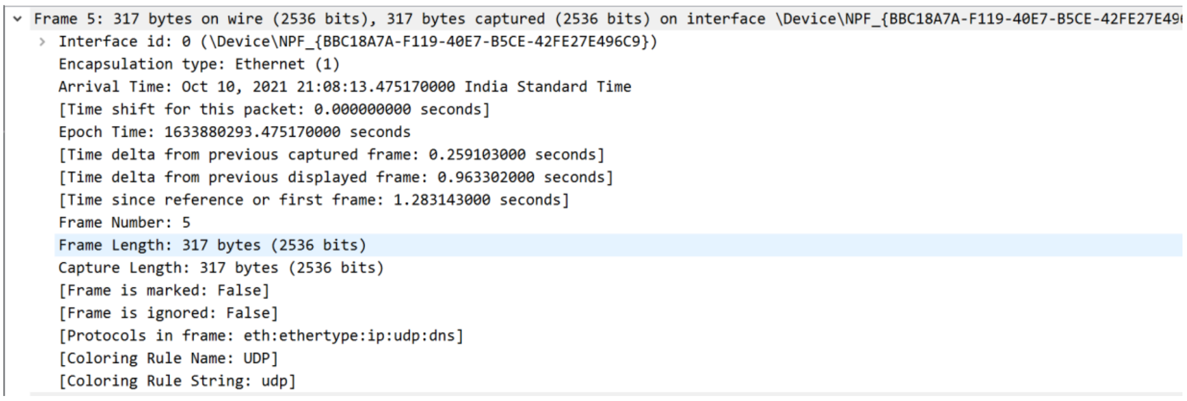
i.



```
⌄ Frame 2: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface \Device\NPF_{BBC18A7A-F119-40E7-B5CE-42FE27E496C9},
  › Interface id: 0 (\Device\NPF_{BBC18A7A-F119-40E7-B5CE-42FE27E496C9})
    Encapsulation type: Ethernet (1)
    Arrival Time: Oct 10, 2021 21:08:12.511868000 India Standard Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1633880292.511868000 seconds
    [Time delta from previous captured frame: 0.319841000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.319841000 seconds]
    Frame Number: 2
    Frame Length: 84 bytes (672 bits)
    Capture Length: 84 bytes (672 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:udp:dns]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
```

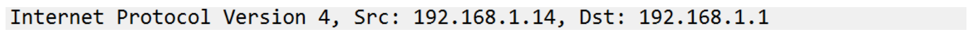
We observe all the details of the packet along with its frame number=2, frame length=84 bytes, time shift of the packet in seconds and also time since first reference and many such details.

The response for the query yields the following packet results:



```
⌄ Frame 5: 317 bytes on wire (2536 bits), 317 bytes captured (2536 bits) on interface \Device\NPF_{BBC18A7A-F119-40E7-B5CE-42FE27E49
  › Interface id: 0 (\Device\NPF_{BBC18A7A-F119-40E7-B5CE-42FE27E496C9})
    Encapsulation type: Ethernet (1)
    Arrival Time: Oct 10, 2021 21:08:13.475170000 India Standard Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1633880293.475170000 seconds
    [Time delta from previous captured frame: 0.259103000 seconds]
    [Time delta from previous displayed frame: 0.963302000 seconds]
    [Time since reference or first frame: 1.283143000 seconds]
    Frame Number: 5
    Frame Length: 317 bytes (2536 bits)
    Capture Length: 317 bytes (2536 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:udp:dns]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
```

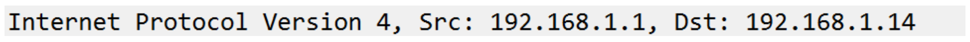
ii.



```
Internet Protocol Version 4, Src: 192.168.1.14, Dst: 192.168.1.1
```

Here we observe that the source IP address for the query made by the client is 192.168.1.14 and its destination address(server) is 192.168.1.1

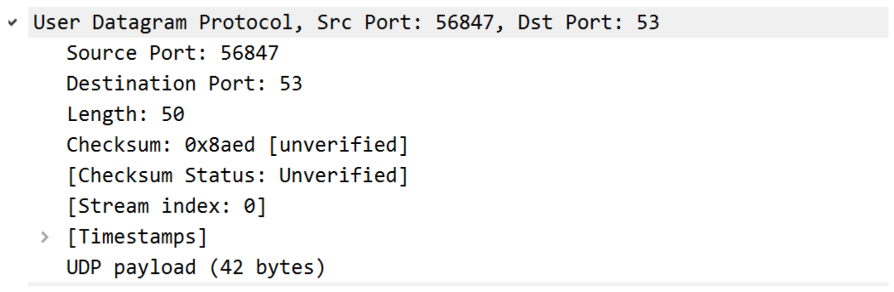
The response to the query has the following fields for source and destination IP addresses:



```
Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.14
```

Here we observe that the source and destination IP addresses compared to the query are basically swapped.

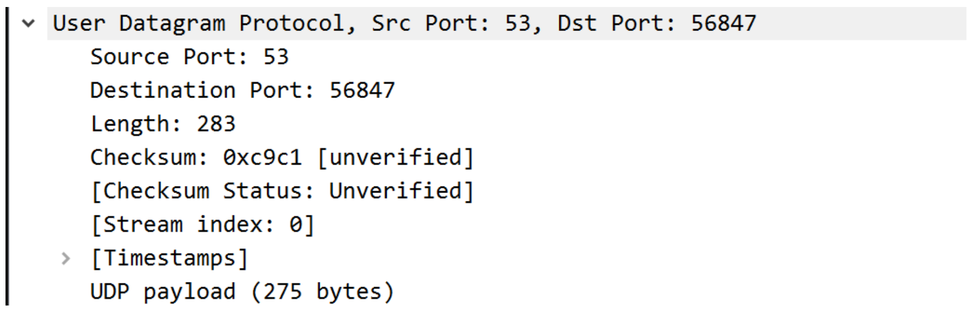
iii.



```
⌄ User Datagram Protocol, Src Port: 56847, Dst Port: 53
     Source Port: 56847
     Destination Port: 53
     Length: 50
     Checksum: 0x8aed [unverified]
     [Checksum Status: Unverified]
     [Stream index: 0]
   › [Timestamps]
     UDP payload (42 bytes)
```

Here we observe that the query made by the client uses source port 56847 and the destination port used is port 53.

The response to the query and its corresponding source and destination ports are:



```
⌄ User Datagram Protocol, Src Port: 53, Dst Port: 56847
     Source Port: 53
     Destination Port: 56847
     Length: 283
     Checksum: 0xc9c1 [unverified]
     [Checksum Status: Unverified]
     [Stream index: 0]
   › [Timestamps]
     UDP payload (275 bytes)
```

*****