Data Communication and Computer Network Laboratory

ASSIGNMENT

MCA 1st year 2nd Sem

Name	Roll-no
Dwip Shekhar Mondal	002210503037

Assignment - IV

Basic Packet Capturing and Analysis →

In this exercise you will be using the python scapy library for packet capturing and network analysis.

• Sniff Packets: Use scapy's sniff() function to capture packets from the network. The sniff() function allows you to specify the number of packets to capture or the duration for which to capture packets. You can also filter the packets based on specific criteria like source/destination IP addresses or protocols. You can also use scapy to sniff packets offline from pcap files.

• Analyse Packets: Once the packets are captured, you can access various attributes of each packet to perform your analysis. Some commonly used attributes include:

– packet.summary(): Provides a summary of the packet, including source and destination IP addresses, protocol, and packet size.

– packet.show(): Displays the detailed information of the packet, including the layers and their corresponding values.

– packet[TCP].payload: Accesses the payload data of a TCP packet.

– packet[UDP].payload: Accesses the payload data of a UDP packet.

Using scapy do the following:

1. Capture 10000 packets (either online/offline)
2. Count the number of distinct host IP addresses and display them.
3. For each distinct pair of source/destination host IP addresses determine the number of TCP/UDP segments exchanged and also the average payload length.
4. For each distinct quadruple of source/destination host IP addresses and source/destination port numbers determine the number of TCP/UDP segments exchanged and also the average payload length.

Question 1:

Problem statement →

Capture 10000 packets (either online/offline) using scapy.

Approach and data structures used →

For this Question i have just made a capture file using wireshark which captured my network traffic for some time. The capture file contains more than 10000 packets which is more than sufficient. I used scrappy to sniff the first 10000 packets of the capture file and find out the summary of packets.

Code →

```
# 1. Capture 10000 packets (either online/offline)

# 2. Count the number of distinct host IP addresses and display them.

# 3. For each distinct pair of source/destination host IP addresses determine the
number of
# TCP/UDP segments exchanged and also the average payload length.
from scapy.all import *


def capture_packets(count=100):
    list_of_packets = sniff(count=count, offline='scapydmp.pcapng')
    return list_of_packets

packets = capture_packets(10000)
print(packets.summary()) # question 1 capture 10k packets on/off
```

Output →

Only the partial result is shown here because it is not possible to show all 10k rows of outputs.

```
PS C:\Users\monda\OneDrive\Desktop\network_lab\set2> python q1.py
Ether / IP / TCP 192.168.0.198:54885 > 52.139.250.209:https A / Raw
Ether / IP / UDP 142.250.76.170:https > 192.168.0.198:63506 / Raw
Ether / IP / UDP 192.168.0.198:63506 > 142.250.76.170:https / Raw
Ether / IP / TCP 52.139.250.209:https > 192.168.0.198:54885 A
Ether / IP / UDP / NBNSHeader / NBNSNodeStatusRequest who has '\\*'
Ether / IP / UDP / DNS Qry "b'b1.nel.goog.'"
Ether / IP / UDP / DNS Qry "b'b1.nel.goog.'"
Ether / ARP who has 192.168.0.198 says 192.168.0.1 / Padding
Ether / ARP is at e0:d5:5e:85:ca:25 says 192.168.0.198
Ether / IP / UDP / DNS Ans "172.217.160.131"
Ether / IP / UDP 192.168.0.198:63216 > 172.217.160.131:https / Raw
Ether / IP / UDP 192.168.0.198:63216 > 172.217.160.131:https / Raw
Ether / IP / UDP 192.168.0.198:63216 > 172.217.160.131:https / Raw
Ether / IP / UDP / DNS Ans
Ether / IP / UDP 172.217.160.131:https > 192.168.0.198:63216 / Raw
Ether / IP / UDP 172.217.160.131:https > 192.168.0.198:63216 / Raw
Ether / IP / UDP 172.217.160.131:https > 192.168.0.198:63216 / Raw
Ether / IP / UDP 172.217.160.131:https > 192.168.0.198:63216 / Raw
Ether / IP / UDP 172.217.160.131:https > 192.168.0.198:63216 / Raw
Ether / IP / UDP 172.217.160.131:https > 192.168.0.198:63216 / Raw
Ether / IP / UDP 192.168.0.198:63216 > 172.217.160.131:https / Raw
Ether / IP / UDP 192.168.0.198:63216 > 172.217.160.131:https / Raw
Ether / IP / UDP 192.168.0.198:63216 > 172.217.160.131:https / Raw
Ether / IP / UDP 172.217.160.131:https > 192.168.0.198:63216 / Raw
Ether / IP / UDP 192.168.0.198:63216 > 172.217.160.131:https / Raw
Ether / IP / UDP 172.217.160.131:https > 192.168.0.198:63216 / Raw
Ether / IP / UDP 172.217.160.131:https > 192.168.0.198:63216 / Raw
Ether / IP / UDP 192.168.0.198:63216 > 172.217.160.131:https / Raw
Ether / IP / UDP 192.168.0.198:63216 > 172.217.160.131:https / Raw
Ether / IP / UDP / DNS Qry "b'beacons.gcp.gvt2.com.'"
Ether / IP / UDP / DNS Qry "b'beacons.gcp.gvt2.com.'"
Ether / IP / UDP / DNS Ans "172.217.167.131"
Ether / IP / TCP 192.168.0.198:55019 > 216.239.32.116:https FA
Ether / IP / TCP 192.168.0.198:55025 > 172.217.167.131:https S
Ether / IP / UDP / DNS Ans "b'beacons-handoff.gcp.gvt2.com.'"
```

Question 2:

Problem statement →

Count the number of distinct host IP addresses and display them.

Approach and data structures used →

For this Question i have just made a capture file using wireshark which listens to my network traffic for some time. The capture file contains more than 10000 packets which is more than sufficient. I used scrappy to sniff the first 10000 packets of the capture file and find out the summary of packets. I used the Set to store the host ip address for each packet. At the end the

set will contain all the distinct host ip addresses as there are no duplicates allowed in set data-structure.

Code →

```
# 1. Capture 10000 packets (either online/offline)

# 2. Count the number of distinct host IP addresses and display them.

# 3. For each distinct pair of source/destination host IP addresses determine the
number of
# TCP/UDP segments exchanged and also the average payload length.

from scapy.all import *

def capture_packets(count=100):
    list_of_packets = sniff(count=count, offline='scapydmp.pcapng')
    return list_of_packets

def getHostIps(pkts):
    hosts = set()
    for pkt in pkts:
        # print(pkt)
        if pkt.haslayer('ARP'):
            # print("source: ", pkt['ARP'].psrc)
            continue
        else:
            # print("source: ", pkt['IP'].src)
            if pkt['IP'].src not in hosts:
                hosts.add(pkt['IP'].src)

    return hosts

packets = capture_packets(10000)

print(packets)

all_hosts = getHostIps(packets)
print("\n\n========= unique host IPs =========\n\n")
for (idx, host) in enumerate(all_hosts):
    print(idx + 1, " > ", host)
```

Output →

```
PS C:\Users\monda\OneDrive\Desktop\network_lab\set2> python q2.py
<Sniffed: TCP:391 UDP:9498 ICMP:101 Other:10>


========= unique host IPs =========


1  >  142.250.77.162
2  >  203.171.247.176
3  >  142.250.182.46
4  >  142.250.71.5
5  >  216.239.32.29
6  >  142.250.77.174
7  >  142.250.196.168
8  >  203.171.247.177
9  >  172.217.167.138
10  >  216.239.32.116
11  >  142.250.193.163
12  >  142.250.193.129
13  >  157.240.1.60
14  >  104.18.131.236
15  >  130.211.4.55
16  >  142.250.196.78
17  >  52.73.100.19
18  >  142.250.195.69
19  >  142.250.67.77
20  >  52.139.250.209
21  >  142.250.196.67
22  >  198.252.206.25
23  >  172.217.166.106
24  >  172.217.31.206
25  >  203.171.247.175
26  >  142.250.195.234
27  >  35.210.63.202
28  >  142.250.183.246
29  >  34.96.128.111
30  >  151.101.129.69
31  >  142.250.193.142
32  >  142.250.205.226
33  >  192.168.0.199
34  >  142.250.193.138
35  >  20.198.119.143
36  >  142.250.205.228
37  >  35.227.233.235
38  >  142.250.193.174
39  >  142.250.183.238
40  >  192.168.0.191
41  >  151.101.1.140
42  >  142.250.76.170
43  >  142.250.195.227
44  >  104.18.3.161
45  >  172.217.167.131
46  >  142.251.10.188
47  >  172.217.160.131
48  >  74.125.130.154
49  >  142.250.183.225
50  >  142.250.195.195
51  >  192.168.0.1
52  >  142.250.196.46
53  >  142.250.195.142
54  >  142.250.182.106
55  >  142.250.67.67
56  >  192.168.0.198
PS C:\Users\monda\OneDrive\Desktop\network_lab\set2>
```

Question 3:

Problem statement →

For each distinct pair of source/destination host IP addresses determine the number of TCP/UDP segments exchanged and also the average payload length.

Approach and data structures used →

For this Question i have just made a capture file using wireshark which listens to my network traffic for some time. The capture file contains more than 10000 packets which is more than sufficient. I used scrappy to sniff the first 10000 packets of the capture file.
I used a dictionary to to keep track of each each source ip, destination ip pair i.e
For each entry of the dictionary the source_ip, destinition_ip pair is the key and an object of protocol, exchanged segments and total_legnth is the value.
If a packet contains TCP or IP layer, add the packet info to dict as (src_ip, dest_ip) tuple as key and required info as value. Repeat this process for all 10000 packets.

Code →

```
from scapy.all import *


def capture_packets(count=100):
    list_of_packets = sniff(count=count, offline='scapydmp.pcapng')
    return list_of_packets

def get_packet_details(pkts):
    host_dst_pairs = {}
    for pkt in pkts:
        # print(pkt.show())
        if not pkt.haslayer('IP'):
            continue

        #key-pair
        src, dest = pkt['IP'].src, pkt['IP'].dst
        key = (src, dest)
        protocol = None
        payload_length = 0
```

```
        if pkt.haslayer('TCP'):
            protocol = 'TCP'
            payload_length = len(pkt['TCP'].payload)

        elif pkt.haslayer('UDP'):
            protocol = 'UDP'
            payload_length = len(pkt['UDP'].payload)

        else:
            # skip other packets that are not TCP or UDP
            continue

        if key not in host_dst_pairs:
            host_dst_pairs[key] = {
                "TCP": 0,
                "UDP": 0,
                "segments": 0,
                "total_length": 0
            }

        host_dst_pairs[key][protocol] += 1
        host_dst_pairs[key]['segments'] += 1
        host_dst_pairs[key]['total_length'] += payload_length

    return host_dst_pairs

packets = capture_packets(100)
print(packets)

pkt_info_dict = get_packet_details(packets)
# print(pkt_info_dict)
print("\n\n========= Output =========\n\n")
print("----------------|----------------------|----------------------|-------------
---------|----------------------|\t")
print("Source IP\t|\tDestination IP\t|\tProtocol\t|\tNo of segments\t|\tAvg payload
len |\t")
print("----------------|----------------------|----------------------|-------------
---------|----------------------|\t")

for key in pkt_info_dict:
    print(f"{key[0]}\t|\t{key[1]}\t|\t{'TCP'if pkt_info_dict[key]['TCP'] > 0 else
'UDP'
}\t\t|\t{pkt_info_dict[key]['segments']}\t\t|\t{pkt_info_dict[key]['total_length'] //
pkt_info_dict[key]['segments']}\t\t|\t")
```

```
print("----------------|----------------------|----------------------|-------------
---------|----------------------|\t")
```

Output →

```
PS C:\Users\monda\OneDrive\Desktop\network_lab\set2> python q3.py
<Sniffed: TCP:42 UDP:56 ICMP:0 Other:2>


========= Output =========


----------------|----------------------|----------------------|----------------------|----------------------|
Source IP       |       Destination IP  |       Protocol       |       No of segments  |       Avg payload len |
----------------|----------------------|----------------------|----------------------|----------------------|
192.168.0.198   |       52.139.250.209  |       TCP            |       1               |       1               |
142.250.76.170  |       192.168.0.198   |       UDP            |       1               |       316             |
192.168.0.198   |       142.250.76.170  |       UDP            |       1               |       33              |
52.139.250.209  |       192.168.0.198   |       TCP            |       1               |       0               |
192.168.0.199   |       255.255.255.255 |       UDP            |       1               |       50              |
192.168.0.198   |       192.168.0.1     |       UDP            |       6               |       33              |
192.168.0.1     |       192.168.0.198   |       UDP            |       6               |       78              |
192.168.0.198   |       172.217.160.131 |       UDP            |       9               |       227             |
172.217.160.131 |       192.168.0.198   |       UDP            |       10              |       567             |
192.168.0.198   |       216.239.32.116  |       TCP            |       2               |       0               |
192.168.0.198   |       172.217.167.131 |       TCP            |       13              |       103             |
216.239.32.116  |       192.168.0.198   |       TCP            |       1               |       6               |
172.217.167.131 |       192.168.0.198   |       TCP            |       14              |       463             |
192.168.0.198   |       142.250.196.46  |       UDP            |       11              |       526             |
142.250.196.46  |       192.168.0.198   |       UDP            |       11              |       392             |
192.168.0.198   |       130.211.4.55    |       TCP            |       1               |       1               |
130.211.4.55    |       192.168.0.198   |       TCP            |       1               |       0               |
142.250.195.234 |       192.168.0.198   |       TCP            |       2               |       39              |
192.168.0.198   |       142.250.195.234 |       TCP            |       2               |       0               |
142.250.196.168 |       192.168.0.198   |       TCP            |       2               |       39              |
192.168.0.198   |       142.250.196.168 |       TCP            |       2               |       0               |
----------------|----------------------|----------------------|----------------------|----------------------|
PS C:\Users\monda\OneDrive\Desktop\network_lab\set2>
```

NOTE: I have used scrappy to print only the first 100 packets for this case because the result is too long for 10k packets.

Question 4:

Problem statement →

For each distinct quadruple of source/destination host IP addresses and source/destination port numbers determine the number of TCP/UDP segments exchanged and also the average payload length.

Approach and data structures used →

For this Question i have just made a capture file using wireshark which listens to my network traffic for some time. The capture file contains more than 10000 packets which is more than sufficient. I used scrappy to sniff the first 10000 packets of the capture file.
I used a dictionary to to keep track of each each source ip, destination ip, source_port, destinition_port pair i.e
For each entry of the dictionary the source_ip, destinition_ip, source_port, destinition_port pair is the key and an object of protocol, exchanged segments and total_legnth is the value.
If a packet contains TCP or IP layer, add the packet info to dict as (src_ip, dest_ip, src_port, dest_port) tuple as key and required info as value. Repeat this process for all 10000 packets.

Code →

```
from scapy.all import *


def capture_packets(count=100):
    list_of_packets = sniff(count=count, offline='scapydmp.pcapng')
    return list_of_packets

def get_packet_details(pkts):
    host_dst_pairs = {}
    for pkt in pkts:
        # print(pkt.show())
        if not pkt.haslayer('IP'):
            continue

        #key-pair
        src, dest = pkt['IP'].src, pkt['IP'].dst
        sport, dport = None,None


        protocol = None
        payload_length = 0

        if pkt.haslayer('TCP'):
            protocol = 'TCP'
```

```
            sport = pkt[protocol].sport
            dport = pkt[protocol].dport
            payload_length = len(pkt['TCP'].payload)

        elif pkt.haslayer('UDP'):
            protocol = 'UDP'
            sport = pkt[protocol].sport
            dport = pkt[protocol].dport
            payload_length = len(pkt['UDP'].payload)

        else:
            # skip other packets that are not TCP or UDP
            continue

        key = (src, dest, sport, dport)

        if key not in host_dst_pairs:
            host_dst_pairs[key] = {
                "TCP": 0,
                "UDP": 0,
                "segments": 0,
                "total_length": 0
            }

        host_dst_pairs[key][protocol] += 1
        host_dst_pairs[key]['segments'] += 1
        host_dst_pairs[key]['total_length'] += payload_length

    return host_dst_pairs

packets = capture_packets(10000)
print(packets)

pkt_info_dict = get_packet_details(packets)
# print(pkt_info_dict)
print("\n\n========= Output =========\n\n")
print("+----------------------+----------------------+--------------+-------------
-+----------------------+----------------------+----------------------+\t")
print("| Source IP\t\t|\tDestination IP\t|\tS-port\t|\tD-port\t|\tProtocol\t|\tNo of
segments\t|\tAvg payload len |\t")
print("+----------------------+----------------------+--------------+-------------
-+----------------------+----------------------+----------------------+\t")

for key in pkt_info_dict:
    print(f"|{str(key[0]).ljust(23)}|\t{key[1]}\t|\t{key[2]}\t|\t{key[3]}\t|\t{'TCP'if
```

```
pkt_info_dict[key]['TCP'] > 0 else 'UDP'
}\t\t|\t{pkt_info_dict[key]['segments']}\t\t|\t{pkt_info_dict[key]['total_length'] //
pkt_info_dict[key]['segments']}\t\t|\t")

print("+----------------------+----------------------+---------------+--------------
-+----------------------+----------------------+----------------------+\t")
```

Output →

```
PS C:\Users\monda\OneDrive\Desktop\network_lab\set2> python q4.py
<Sniffed: TCP:42 UDP:56 ICMP:0 Other:2>


========= Output =========


+----------------------+----------------------+---------------+---------------+----------------------+----------------------+----------------------+
| Source IP            |      Destination IP  |      S-port   |      D-port   |      Protocol        |      No of segments  |      Avg payload len |
+----------------------+----------------------+---------------+---------------+----------------------+----------------------+----------------------+
|192.168.0.198         |      52.139.250.209  |      54885    |      443      |      TCP             |      1               |      1               |
|142.250.76.170        |      192.168.0.198   |      443      |      63506    |      UDP             |      1               |      316             |
|192.168.0.198         |      142.250.76.170  |      63506    |      443      |      UDP             |      1               |      33              |
|52.139.250.209        |      192.168.0.198   |      443      |      54885    |      TCP             |      1               |      0               |
|192.168.0.199         |      255.255.255.255 |      50633    |      137      |      UDP             |      1               |      50              |
|192.168.0.198         |      192.168.0.1     |      56783    |      53       |      UDP             |      1               |      29              |
|192.168.0.198         |      192.168.0.1     |      62323    |      53       |      UDP             |      1               |      29              |
|192.168.0.1           |      192.168.0.198   |      53       |      56783    |      UDP             |      1               |      45              |
|192.168.0.198         |      172.217.160.131 |      63216    |      443      |      UDP             |      9               |      227             |
|192.168.0.1           |      192.168.0.198   |      53       |      62323    |      UDP             |      1               |      89              |
|172.217.160.131       |      192.168.0.198   |      443      |      63216    |      UDP             |      10              |      567             |
|192.168.0.198         |      192.168.0.1     |      57693    |      53       |      UDP             |      1               |      38              |
|192.168.0.198         |      192.168.0.1     |      55401    |      53       |      UDP             |      1               |      38              |
|192.168.0.1           |      192.168.0.198   |      53       |      57693    |      UDP             |      1               |      54              |
|192.168.0.198         |      216.239.32.116  |      55019    |      443      |      TCP             |      2               |      0               |
|192.168.0.198         |      172.217.167.131 |      55025    |      443      |      TCP             |      13              |      103             |
|192.168.0.1           |      192.168.0.198   |      53       |      55401    |      UDP             |      1               |      151             |
|216.239.32.116        |      192.168.0.198   |      443      |      55019    |      TCP             |      1               |      6               |
|172.217.167.131       |      192.168.0.198   |      443      |      55025    |      TCP             |      14              |      463             |
|192.168.0.198         |      192.168.0.1     |      61343    |      53       |      UDP             |      1               |      33              |
|192.168.0.198         |      192.168.0.1     |      63748    |      53       |      UDP             |      1               |      33              |
|192.168.0.1           |      192.168.0.198   |      53       |      61343    |      UDP             |      1               |      49              |
|192.168.0.1           |      192.168.0.198   |      53       |      63748    |      UDP             |      1               |      83              |
|192.168.0.198         |      142.250.196.46  |      56645    |      443      |      UDP             |      11              |      526             |
|142.250.196.46        |      192.168.0.198   |      443      |      56645    |      UDP             |      11              |      392             |
|192.168.0.198         |      130.211.4.55    |      54635    |      443      |      TCP             |      1               |      1               |
|130.211.4.55          |      192.168.0.198   |      443      |      54635    |      TCP             |      1               |      0               |
|142.250.195.234       |      192.168.0.198   |      443      |      55005    |      TCP             |      2               |      39              |
|192.168.0.198         |      142.250.195.234 |      55005    |      443      |      TCP             |      2               |      0               |
|142.250.196.168       |      192.168.0.198   |      443      |      55008    |      TCP             |      2               |      39              |
|192.168.0.198         |      142.250.196.168 |      55008    |      443      |      TCP             |      2               |      0               |
+----------------------+----------------------+---------------+---------------+----------------------+----------------------+----------------------+
PS C:\Users\monda\OneDrive\Desktop\network_lab\set2>
```

Assignment - V

Packet Crafting →

Packet crafting is the process of manually creating and customising network packets with specific headers, payloads, and other fields. This technique is commonly used in network security, network testing, and network debugging scenarios. In this exercise you will be using the scapy library to create custom packets from scratch. The scapy library provides a high-level interface to construct packets, allowing you to set fields in various network layers, such as Ethernet, IP, TCP, UDP, and more.

Using scapy do the following:

1. Write a python program which gets a network address from the user and generates all possible host IP addresses within that network. Then it sends a dummy ICMP echo request message to all the hosts. Display only those hosts from which you receive corresponding ICMP echo reply messages within a predefined time out period.

2. Write a python program which gets a host IP address from the user and sends TCP SYN segments to all the ports within the range 0 to 1023. Display only those port numbers from which you receive corresponding TCP SYN+ACK segments within a predefined time out period.

Question 1:

Problem statement →

Write a python program which gets a network address from the user and generates all possible host IP addresses within that network. Then it sends a dummy ICMP echo request message to all the hosts. Display only those hosts from which you receive corresponding ICMP echo reply messages within a predefined time out period.

Approach and data structures used →

First I have made a function (network_info) that takes a cidr address and returns the netmask , network_address and other info that is used for finding the whole host ip range for that network address.
Next I have made another function (get_all_network_host) that takes network info returned by the above function to generate all the host ip addresses belonging to that network address.
Then I made the driver function that takes the generated list of ip addresses and sends ICMP packets to each of them, if it gets a response of the ICMP request, means that the host is alive and it adds the ip to the list of alive host addresses which is the final result.

Code →

```
import ipaddress
import concurrent.futures
from scapy.all import Ether, IP, TCP, ICMP, sr, sr1, srloop



def network_info(ip_cidr_addr):
    ip, cidr = ip_cidr_addr.split('/')
    cidr_num =  int(cidr)
    octates = ip.split('.')
    ip_bin_octates = [bin(int(ele)) for ele in octates]
    q = []
    # calculate netmask in bin str i.e how many 1, and 0s
    for i in range(cidr_num):
        q.append('1')
        if (i + 1) % 8 == 0 and i != 32 - 1:
            q.append('.')
    for i in range(cidr_num, 32):
        q.append('0')
        if (i + 1) % 8 == 0 and i != 32 - 1:
            q.append('.')
    # print(q)
    netmask_bin_octates = [bin(int(ele, 2)) for ele in "".join(list(q)).split(".")]
    netmask = ".".join([str(int(ele, 2)) for ele in netmask_bin_octates])

    net_id_octates = []
```

```
    for e1, e2 in zip(ip_bin_octates, netmask_bin_octates):
        net_id_octates.append(int(e1, 2) & int(e2, 2))


    net_id = ".".join([str(ele) for ele in net_id_octates])

    res_obj = {
        "ip" : ip,
        "cidr": cidr_num,
        "netmask": netmask,
        "network_addr": net_id
    }

    return res_obj

def get_all_network_hosts(ip_cidr):
    net_info = network_info(ip_cidr)
    cidr , network_addr = net_info['cidr'], net_info['network_addr']
    all_possible_ip = []
    all_possible_ip.append(network_addr)
    total_ip = 2 ** (32 - cidr) - 1

    net_id_octates = [int(ele) for ele in network_addr.split(".")]

    i = 0
    o1, o2, o3, o4 = net_id_octates

    while i < total_ip:
        if o4 + 1 > 255:
            o4 = 0
            if o3 + 1 > 255:
                o3 = 0
                if o2 + 1 > 255:
                    o2 = 0
                    if o1 + 1 > 255:
                        raise ValueError("Value out of range!")
                    else:
                        o1 += 1
                        all_possible_ip.append(f"{o1}.{o2}.{o3}.{o4}")
                else:
                    o2 += 1
                    all_possible_ip.append(f"{o1}.{o2}.{o3}.{o4}")
            else:
                o3 += 1
```

```
                all_possible_ip.append(f"{o1}.{o2}.{o3}.{o4}")
        else:
            o4 += 1
            all_possible_ip.append(f"{o1}.{o2}.{o3}.{o4}")
        i += 1


    # print(total_ip, network_addr)
    # print(all_possible_ip)
    if cidr <= 30:
        all_possible_ip.pop()
        all_possible_ip.pop(0)

    # all_possible_ip.pop()
    # all_possible_ip.pop(0)

    return all_possible_ip

def network_scan(hosts):
    alive = []
    for host in hosts:
        packet = IP(dst=host) / ICMP() / "Hello"

        # response = srloop(packet, count=1, verbose=False, timeout=1)[0]
        response = sr1(packet, verbose=False, timeout=1)

        if response:
            print(f"{host} --> responsive")
            alive.append(host)


    print("All responsive host ip address -----")
    print(alive)


ip = input("enter ip: ")
hosts = get_all_network_hosts(ip)

print("All host ip addresses in the network range -----")
print(hosts)

print("Scanning network to find responsive hosts -----")
network_scan(hosts)
```

Output →

```
PS C:\Users\monda\OneDrive\Desktop\network_lab\set3> python q1.py
enter ip: 192.168.0.138/25
All host ip addresses in the network range -----
['192.168.0.129', '192.168.0.130', '192.168.0.131', '192.168.0.132', '192.168.0.133',
'192.168.0.134', '192.168.0.135', '192.168.0.136', '192.168.0.137', '192.168.0.138',
'192.168.0.139', '192.168.0.140', '192.168.0.141', '192.168.0.142', '192.168.0.143',
'192.168.0.144', '192.168.0.145', '192.168.0.146', '192.168.0.147', '192.168.0.148',
'192.168.0.149', '192.168.0.150', '192.168.0.151', '192.168.0.152', '192.168.0.153',
'192.168.0.154', '192.168.0.155', '192.168.0.156', '192.168.0.157', '192.168.0.158',
'192.168.0.159', '192.168.0.160', '192.168.0.161', '192.168.0.162', '192.168.0.163',
'192.168.0.164', '192.168.0.165', '192.168.0.166', '192.168.0.167', '192.168.0.168',
'192.168.0.169', '192.168.0.170', '192.168.0.171', '192.168.0.172', '192.168.0.173',
'192.168.0.174', '192.168.0.175', '192.168.0.176', '192.168.0.177', '192.168.0.178',
'192.168.0.179', '192.168.0.180', '192.168.0.181', '192.168.0.182', '192.168.0.183',
'192.168.0.184', '192.168.0.185', '192.168.0.186', '192.168.0.187', '192.168.0.188',
'192.168.0.189', '192.168.0.190', '192.168.0.191', '192.168.0.192', '192.168.0.193',
'192.168.0.194', '192.168.0.195', '192.168.0.196', '192.168.0.197', '192.168.0.198',
'192.168.0.199', '192.168.0.200', '192.168.0.201', '192.168.0.202', '192.168.0.203',
'192.168.0.204', '192.168.0.205', '192.168.0.206', '192.168.0.207', '192.168.0.208',
'192.168.0.209', '192.168.0.210', '192.168.0.211', '192.168.0.212', '192.168.0.213',
'192.168.0.214', '192.168.0.215', '192.168.0.216', '192.168.0.217', '192.168.0.218',
'192.168.0.219', '192.168.0.220', '192.168.0.221', '192.168.0.222', '192.168.0.223',
'192.168.0.224', '192.168.0.225', '192.168.0.226', '192.168.0.227', '192.168.0.228',
'192.168.0.229', '192.168.0.230', '192.168.0.231', '192.168.0.232', '192.168.0.233',
'192.168.0.234', '192.168.0.235', '192.168.0.236', '192.168.0.237', '192.168.0.238',
'192.168.0.239', '192.168.0.240', '192.168.0.241', '192.168.0.242', '192.168.0.243',
'192.168.0.244', '192.168.0.245', '192.168.0.246', '192.168.0.247', '192.168.0.248',
'192.168.0.249', '192.168.0.250', '192.168.0.251', '192.168.0.252', '192.168.0.253',
'192.168.0.254']
Scanning network to find responsive hosts -----
192.168.0.192 --> responsive
192.168.0.198 --> responsive
192.168.0.200 --> responsive
All responsive host ip address -----
['192.168.0.192', '192.168.0.198', '192.168.0.200']
PS C:\Users\monda\OneDrive\Desktop\network_lab\set3>
```

Note: Here I have used my home network range, these are the actual devices that were connected to the network at that time and `192.168.0.192 is the private IP of my device in the network`.

Question 2:

Problem statement →

Write a python program which gets a host IP address from the user and sends TCP SYN segments to all the ports within the range 0 to 1023. Display only those port numbers from which you receive corresponding TCP SYN+ACK segments within a predefined time out period.

Approach and data structures used →

First I have made a function that takes a host ip address string and a range of port numbers, and a set of open ports then it sends TCP packets to the ip for each port in that range with a syn flag. It receives a syn+ack packet then that port is open and it adds that port to the set of open ports.
Next I have made the driver function that spawns 16 threads and for each threads executes the above function and scans a port range of 64 port in such a way that 1st thread scans port 0-63, 2nd thread scans 64-137 and so on until all ports from 0-1023 are scanned. After all the threads are finished executing, it returns the set of ports that are open.

Code →

```
from scapy.all import IP, TCP, sr1
import threading



def port_scan(host, start_port, end_port, open_ports):
    for port in range(start_port, end_port):
        # TCP SYN packet
        # print("scanning ", port)
        tcp_packet = IP(dst=host) / TCP(dport=port, flags="S")

        response = sr1(tcp_packet, timeout=1, verbose=False)
```

```
        if response is not None and response['TCP'].flags == "SA":
            print("open --> ", port)
            open_ports.add(port)

def concurrent_port_scan(ip_addr, num_threads=16, ports_per_thread=64):

    open_ports = set()
    threads = []

    for i in range(num_threads):

        start_port = i * ports_per_thread
        end_port = start_port + ports_per_thread

        thread = threading.Thread(target=port_scan, args=(ip_addr, start_port,
end_port, open_ports))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    return open_ports

if __name__ == "__main__":

    ip_addr = input("Enter the Network Address : ")

    print("scanning top ports -->")

    open_ports = concurrent_port_scan(ip_addr)

    print("all open Ports : ")

    print(open_ports)
```

Output →

```
PS C:\Users\monda\OneDrive\Desktop\network_lab\set3> python q2.py
Enter the Network Address : 115.187.42.228
scanning top ports -->
WARNING: Mac address to reach destination not found. Using broadcast.
open -->  80
open -->  25
open -->  53
all open Ports :
{80, 25, 53}
PS C:\Users\monda\OneDrive\Desktop\network_lab\set3>
```