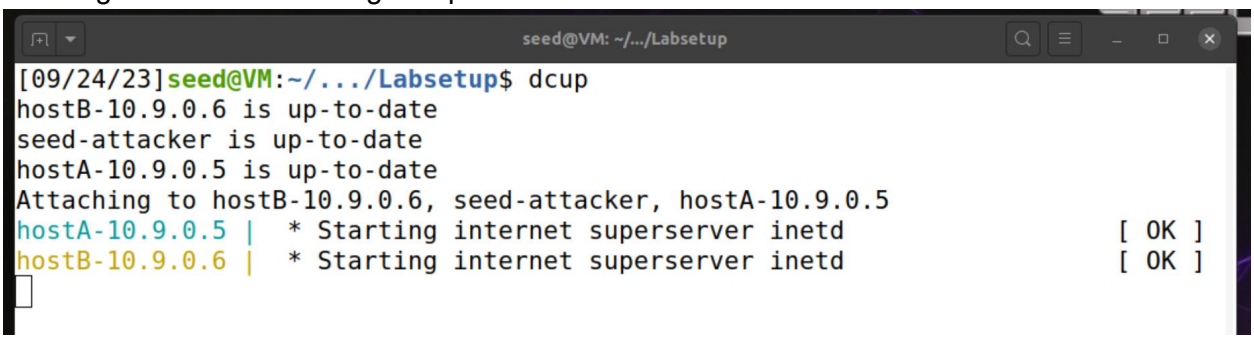
# Lab 1: Packet Sniffing and Spoofing

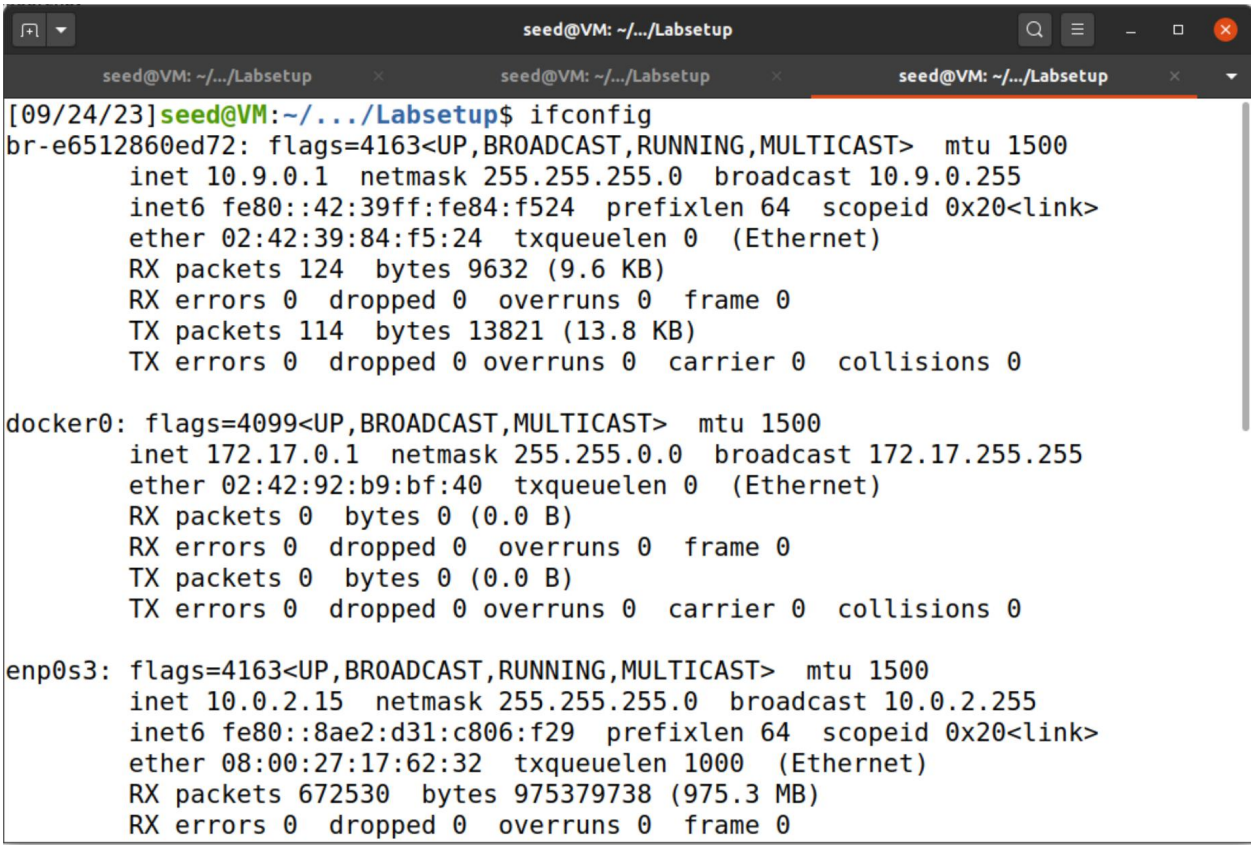
Name: Guo Yuchen
Student ID: 1004885

## Setup and preparation

Starting the containers using `dcup`




Using `ifconfig` to get network interface ID `br-e6512860ed72`



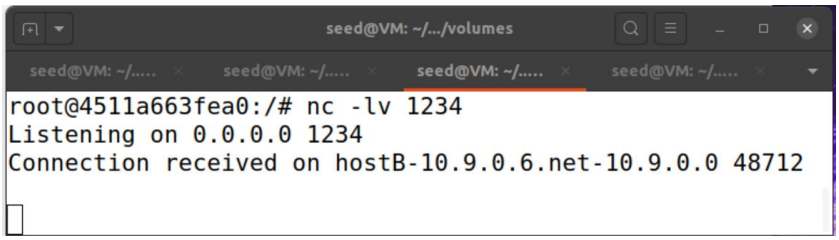
# Task 1.1A

Create a Python file `sniffer.py` and make it executable using `chmod a+x sniffer.py`.

```python
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface="br-e6512860ed72", prn=print_pkt)
```

## Running sniffer.py with root privilege

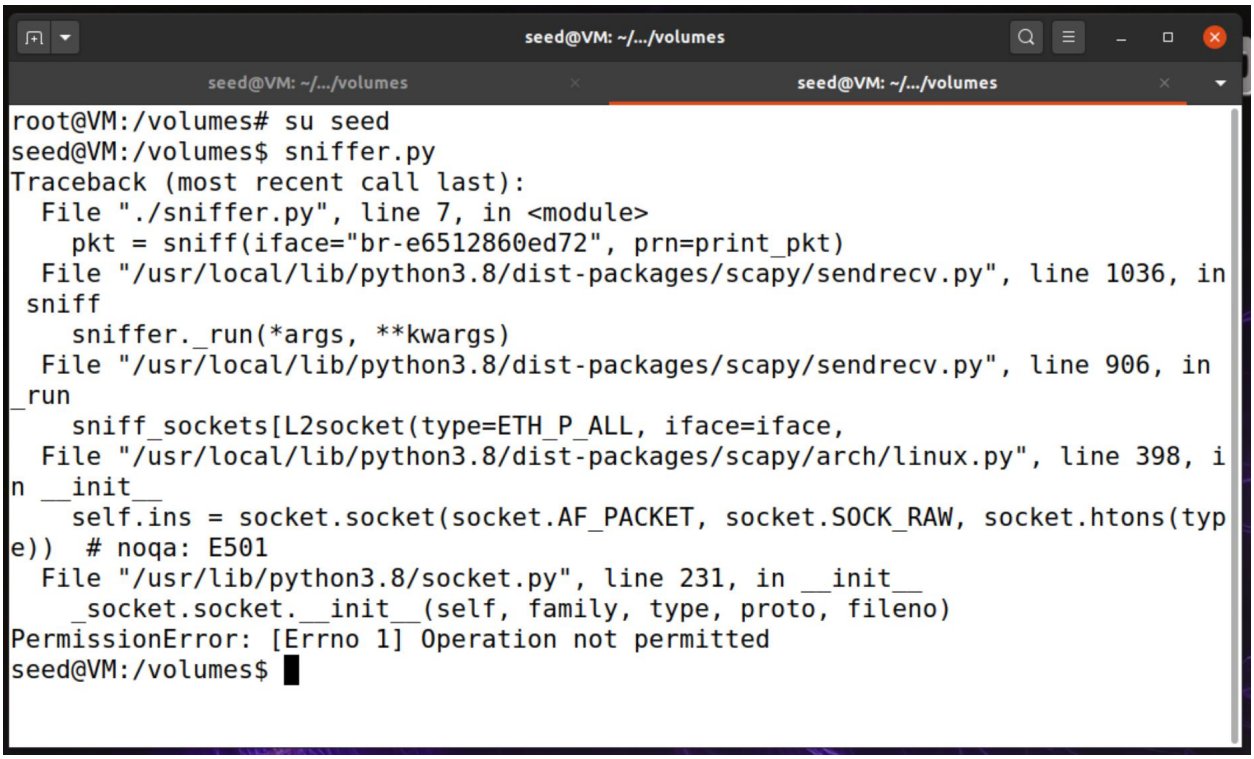
To create some traffic, use `nc` command on host 10.9.0.5 to listen to the connection from host 10.9.0.6




Traffic captured by Scapy:


```
###[ Ethernet ]###
  dst       = 02:42:39:84:f5:24
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 66
     id        = 46534
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = udp
     chksum    = 0xb7ca
     src       = 10.9.0.5
     dst       = 192.168.2.100
     \options   \
###[ UDP ]###
        sport     = 59663
        dport     = domain
        len       = 46
        chksum    = 0xcd59
###[ DNS ]###
           id        = 15085
           qr        = 0
           opcode    = QUERY
           aa        = 0
           tc        = 0
           rd        = 1
```

## Running sniffer.py without root privilege




## Observation

1. Only root can sniff packages.
2. Reasons: only root is allowed to open raw sockets, which is used by sniff to receive packets such that the packet won't be intercepted by the normal socket.

# Task 1.1B

## Capture icmp packet

Modify the filter parameter to be “icmp” to capture only the ICMP packet. Ping from 10.9.0.5 to 10.9.0.6 since ping sends icmp package.

```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface="br-e6512860ed72", filter="icmp", prn=print_pkt)
```

```
seed@VM: ~/.../volumes
seed@VM: ~/.../Labsetup  seed@VM: ~/.../Labsetup  seed@VM: ~/.../volumes
[09/24/23]seed@VM:~/.../volumes$ docksh 45
root@4511a663fea0:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.287 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.053 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.191 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.280 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.362 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.114 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.098 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.216 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=64 time=0.092 ms
```

The packets are captured by the sniffer as shown:

```
seed@VM: ~/.../volumes
seed@VM: ~/.../volumes  seed@VM: ~/.../volumes  seed@VM: ~/.../volumes  seed@VM: ~/.../volumes
root@VM:/volumes# sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 48906
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x6782
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x9897
        id        = 0x3f
        seq       = 0x1
###[ Raw ]###
           load      = '\x94\x91\x12e\x00\x00\x00\x00\xf6^\x03\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\
x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

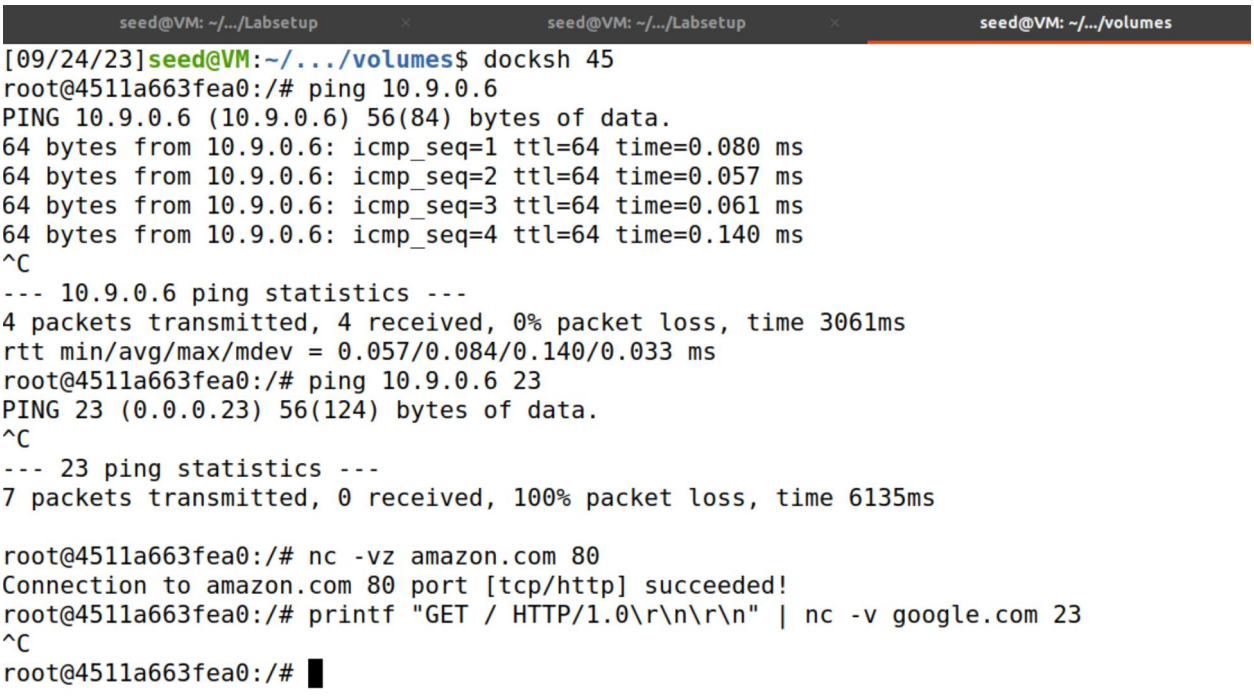
## Capture any TCP packet that comes from a particular IP and with a destination port number 23.

Change the filter parameter accordingly.

```python
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface="br-e6512860ed72",
            filter="tcp and src net 10.9.0.5 and dst port 23",
            prn=print_pkt)
```

Then try several different command:

- ping 10.9.0.6: wrong type, wrong dst port
- ping 10.9.0.6/23: wrong type
- nc -vz amazon.com 80: wrong dst port
- nc -v google.com 23



Sniffer captures:

```
###[ Ethernet ]###
  dst       = 02:42:39:84:f5:24
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 60
     id        = 21530
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x79b2
     src       = 10.9.0.5
     dst       = 74.125.24.101
     \options   \
###[ TCP ]###
        sport     = 55966
        dport     = telnet
        seq       = 1706082090
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x6d1e
```

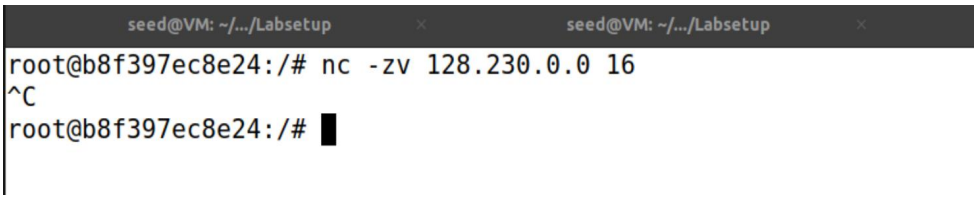
It only responds for the last case, when the port number, src ip, and packet type fit the filter.

## Capture packets comes from or to go to a particular subnet

Choose the subnet to be 128. 230.0.0/16 and change to filter parameter accordingly.

```python
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface="br-e6512860ed72",
            filter="net 128.230.0.0/16",
            prn=print_pkt)
```

Let one of the host sends a tcp packet to this ip address:



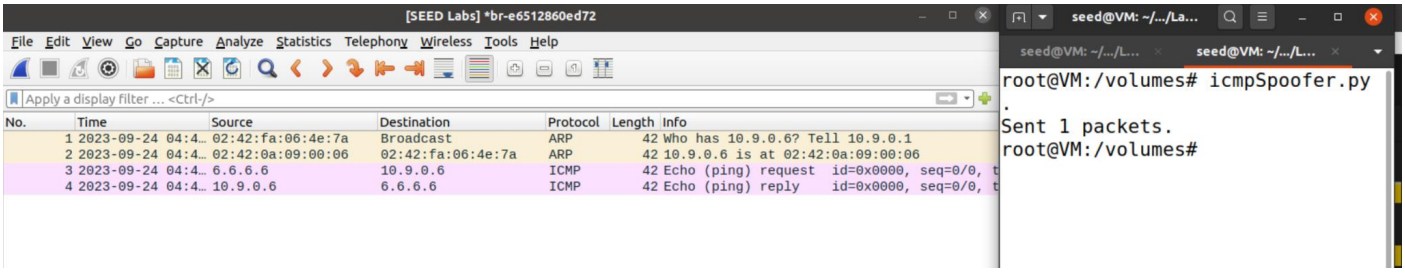

Then the sniffer program captures the packet which has dst and port as specified:

```
###[ Ethernet ]###
  dst       = 02:42:39:84:f5:24
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 60
     id        = 41093
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xf42
     src       = 10.9.0.6
     dst       = 128.230.0.0
     \options   \
###[ TCP ]###
        sport     = 56732
        dport     = 16
        seq       = 2464214675
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x8b23
        urgptr    = 0
```

# Task 1.2

icmpSpoofer.py:

```
#!/usr/bin/env python3
from scapy.all import *
a = IP()
a.src = "6.6.6.6"
a.dst = "10.9.0.6"
b = ICMP()
p = a/b
send(p)
```

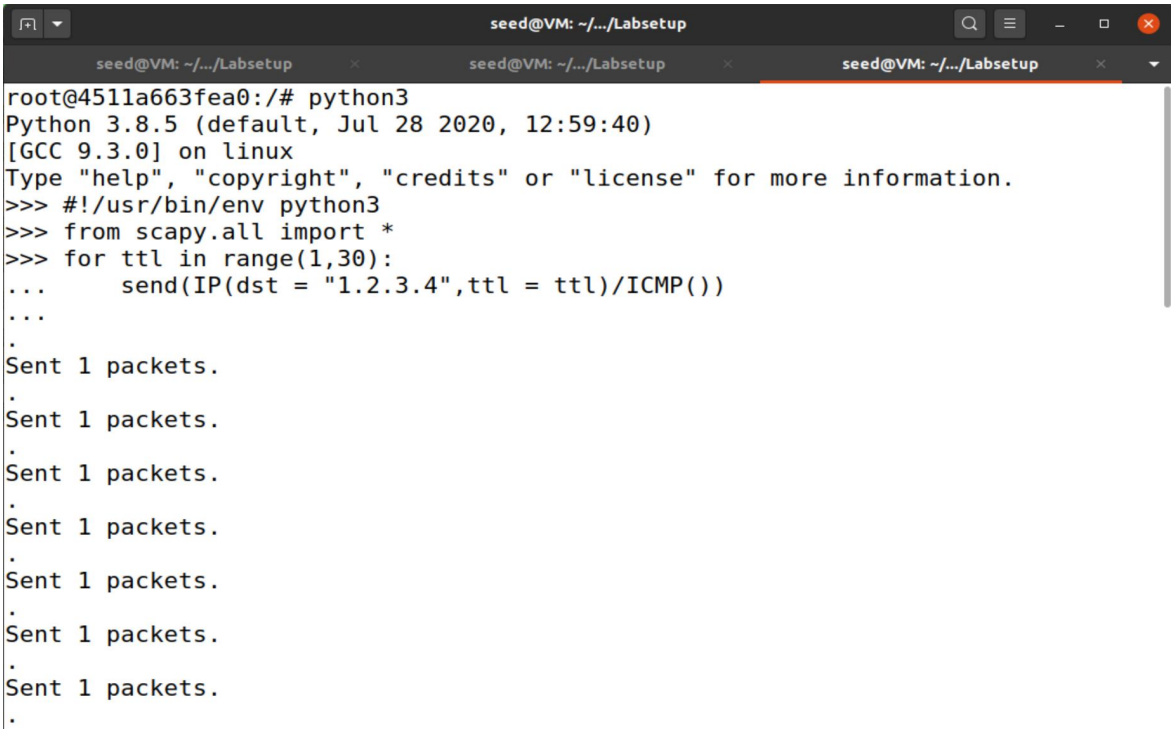



In the icmpSpoofer.py, the program sends a spoofed icmp echo request with source ip 6.6.6.6 to the host 10.9.0.6. And as observed in the Wireshark (the last line), the host 10.9.0.6 receives an icmp echo reply from the spoofed source 6.6.6.6, which is specified in our spoofer program.

# Task 1.3

To estimate the distance between the VM and a selected destination in terms of number of routers, let one of the hosts to run the program as shown in the terminal.

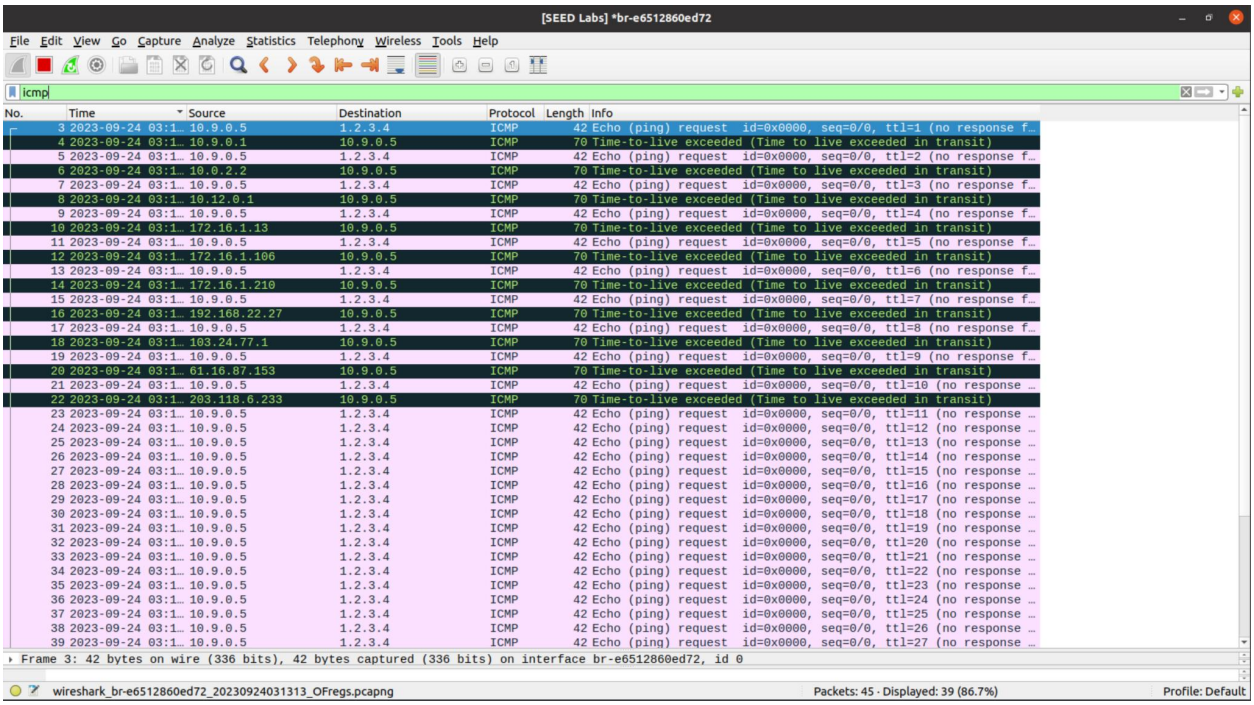
Basically, we are creating a sequence of icmp packets with a destination ip address to be 1.2.3.4, and repeatedly increased ttl from 1 to 30.




Then as observed and recorded in the Wireshark, the routes are shown in the black lines source column, which I summarized as follows:

1. 10.9.0.1
2. 10.0.2.2
3. 10.12.0.1
4. 172.16.1.13
5. 172.16.1.106
6. 172.16.1.210
7. 192.168.22.27
8. 103.24.77.1
9. 61.16.87.153
10. 203.118.6.233

And since 1.2.3.4 does not exist, the host could not get an echo reply from there.


[SEED Labs] *br-e6512860ed72
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmp
No. Time Source Destination Protocol Length Info
3 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response f…
4 2023-09-24 03:1… 10.9.0.1 10.9.0.5 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
5 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=2 (no response f…
6 2023-09-24 03:1… 10.0.2.2 10.9.0.5 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
7 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=3 (no response f…
8 2023-09-24 03:1… 10.12.0.1 10.9.0.5 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
9 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=4 (no response f…
10 2023-09-24 03:1… 172.16.1.13 10.9.0.5 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
11 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=5 (no response f…
12 2023-09-24 03:1… 172.16.1.106 10.9.0.5 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
13 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response f…
14 2023-09-24 03:1… 172.16.1.210 10.9.0.5 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
15 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response f…
16 2023-09-24 03:1… 192.168.22.27 10.9.0.5 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
17 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=8 (no response f…
18 2023-09-24 03:1… 103.24.77.1 10.9.0.5 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
19 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=9 (no response f…
20 2023-09-24 03:1… 61.16.87.153 10.9.0.5 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
21 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=10 (no response …
22 2023-09-24 03:1… 203.118.6.233 10.9.0.5 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
23 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=11 (no response …
24 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=12 (no response …
25 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=13 (no response …
26 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=14 (no response …
27 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=15 (no response …
28 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=16 (no response …
29 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=17 (no response …
30 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=18 (no response …
31 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=19 (no response …
32 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=20 (no response …
33 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=21 (no response …
34 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=22 (no response …
35 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=23 (no response …
36 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=24 (no response …
37 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=25 (no response …
38 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=26 (no response …
39 2023-09-24 03:1… 10.9.0.5 1.2.3.4 ICMP 42 Echo (ping) request id=0x0000, seq=0/0, ttl=27 (no response …
Frame 3: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-e6512860ed72, id 0
wireshark_br-e6512860ed72_20230924031313_OFregs.pcapng
Packets: 45 · Displayed: 39 (86.7%)
Profile: Default

# Task 1.4

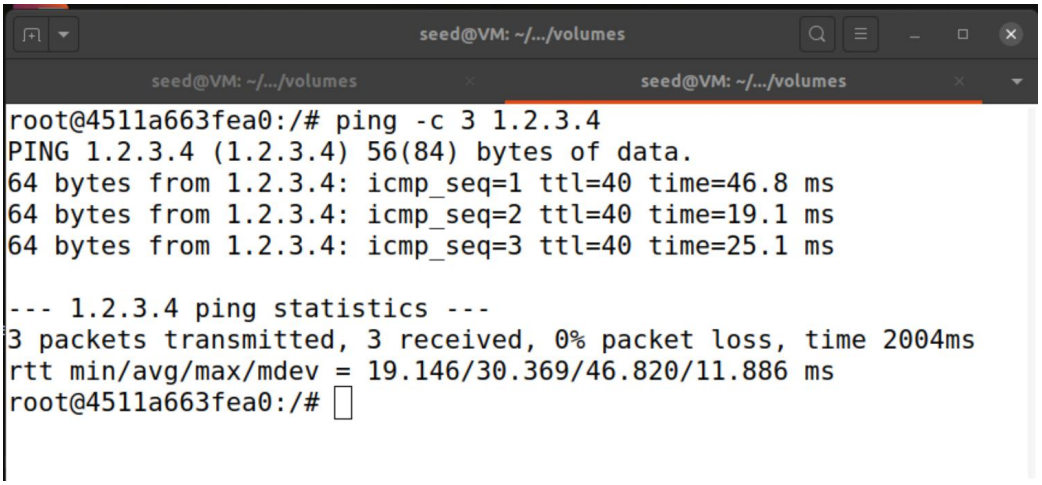
The sniff and spoof program ss.py is as follows:

```
#!/usr/bin/env python3
from scapy.all import *

# def print_pkt(pkt):
#   pkt.show()

def spoof_pkt(pkt):
    src = pkt[IP].dst
    dst = pkt[IP].src
    a = IP(src=src,dst=dst,ttl=40)
    b = ICMP(type=0,id=pkt[ICMP].id, seq=pkt[ICMP].seq, code=0)
    raw = pkt[Raw]
    send(a/b/raw)

pkt = sniff(iface="br-e6512860ed72",
            filter="icmp and host 10.9.0.5",
            prn=spoof_pkt)
```

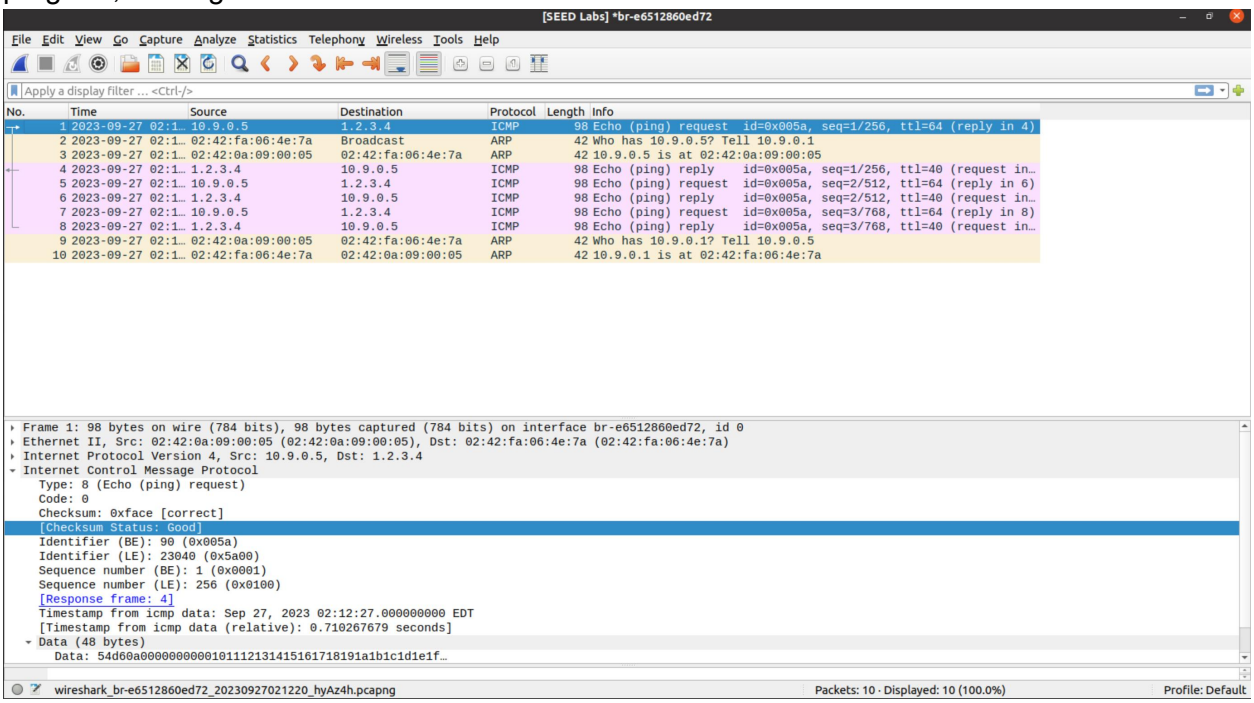
After a packet is sniffed, the callback function spoof_pkt is called to spoof the packet. The spoof_pkt will reverse the destination and source ip address, take the icmp id and seq number, then construct an icmp echo reply message to send it back.

# ping 1.2.3.4

# a non-existing host on the Internet

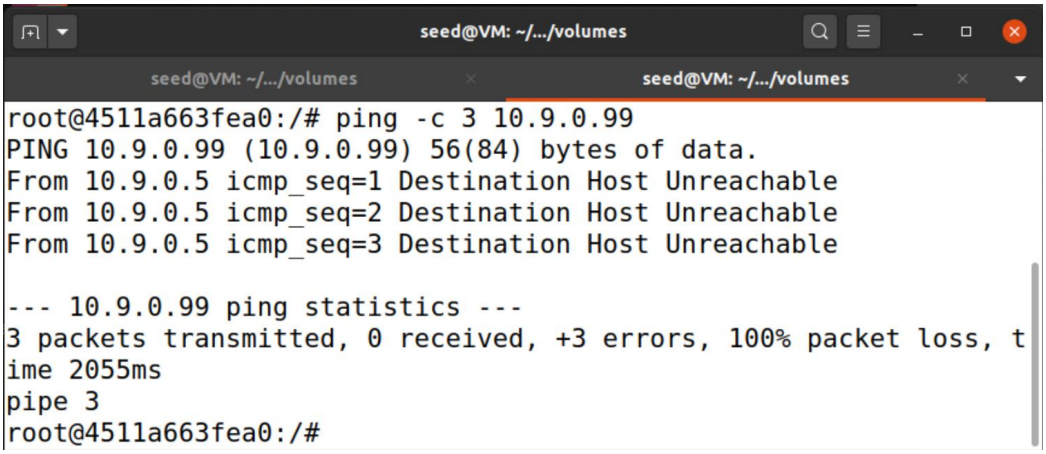


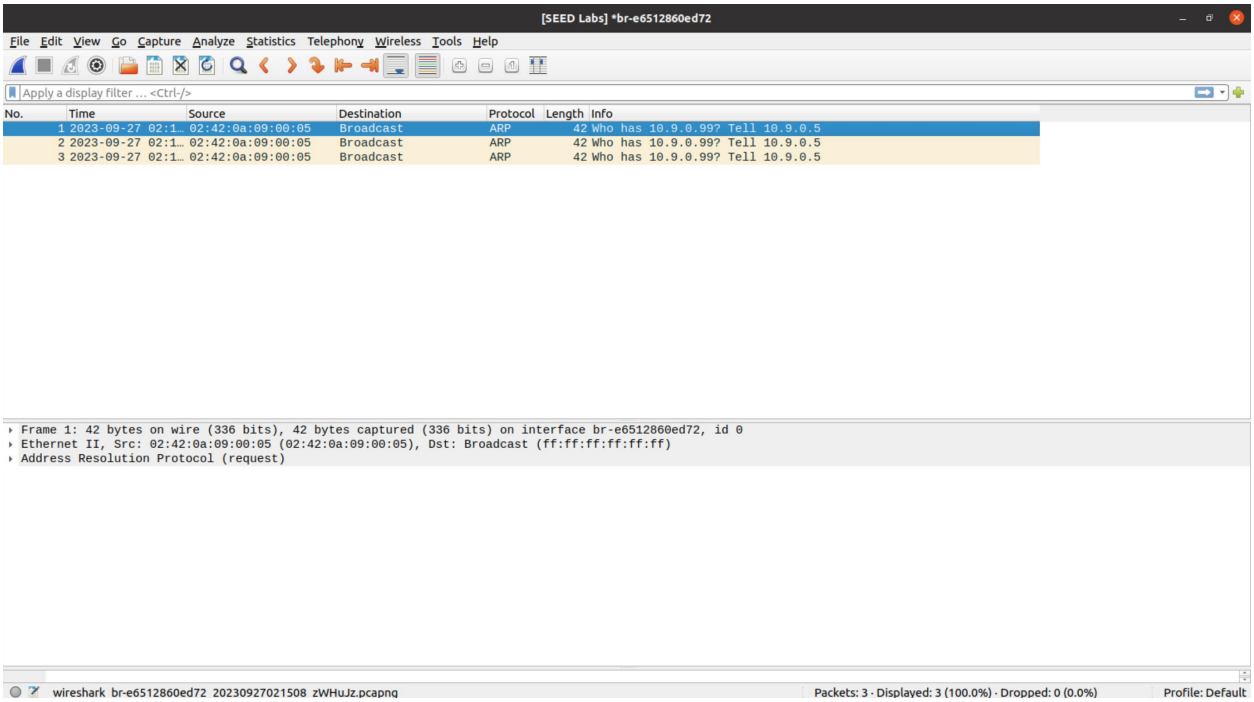

Observation: the ping program receives 3 echo replies, which are spoofed by the spoofer program, although 1.2.3.4 does not exist.



# ping 10.9.0.99

# a non-existing host on the LAN

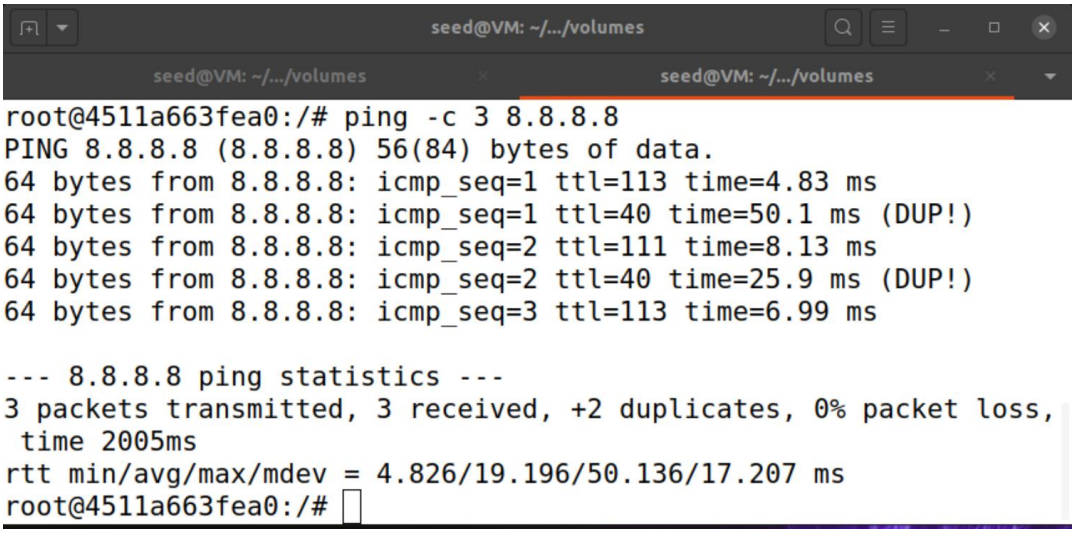


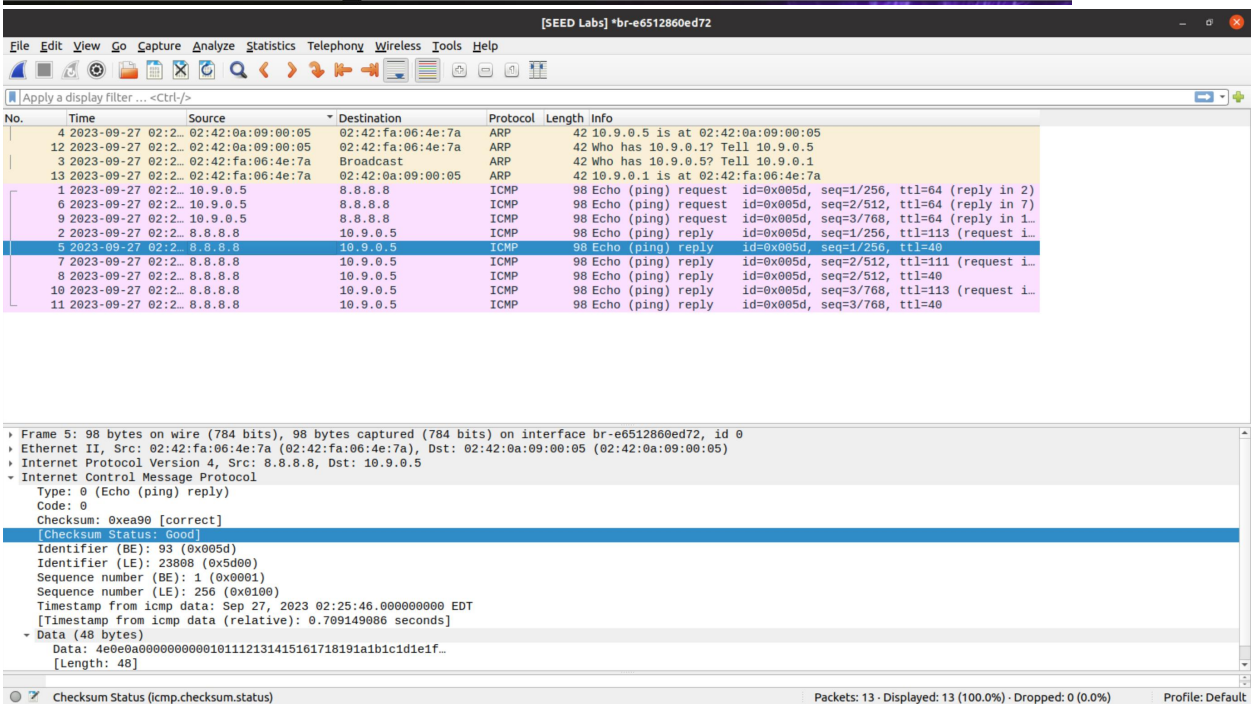




Observation: no packets is received, means that the spoofer could not spoof any message. Because the 10.9.0.99 is on the same LAN with the host 10.9.0.5, thus before they can exchange message, it needs the MAC address of 10.9.0.99. Since it's not stored in the ARP cache, it sends out a broadcast message to ask for reply, and the real communication only takes place after it acquires the destination device's MAC address. In our case, there is neither ARP reply or spoofed ARP reply, our spoofed icmp packet will not be received.

# ping 8.8.8.8

# an existing host on the Internet




Observation: both ping requests (the real reply and the spoofed reply) are received by the ping program. But the ones that are spoofed by me(ttl is set to 40 to distinguish) are marked as duplicated since they come later than the real one.