Part 2:

In this part we decided to run our server.py code twice on two separate terminals in Server's VM and client.py on terminal in Client's VM just like you guided us in the exercise.

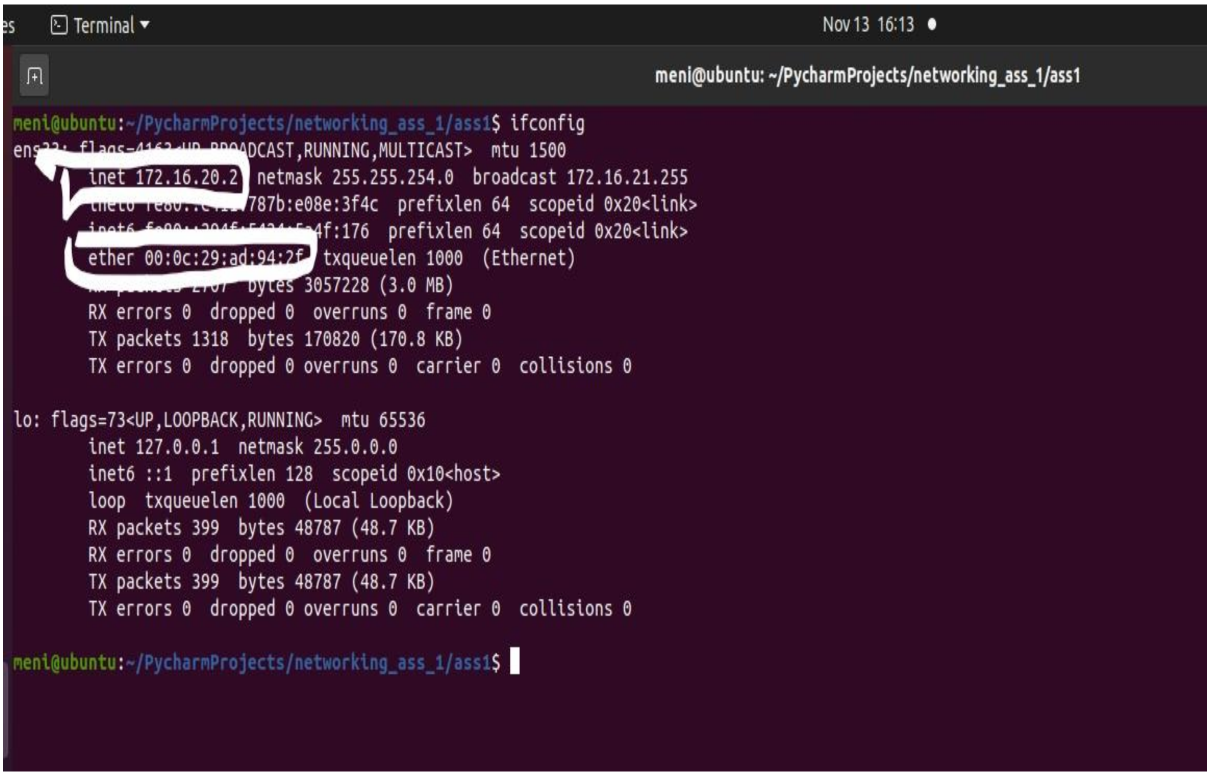
Now first lets determine the ip's and MAC's of the VM's:

Server VM:

Ip: 172.16.20.2

MAC : 00:0c:29:ad:94:2f

Print screen to show how I found these details:



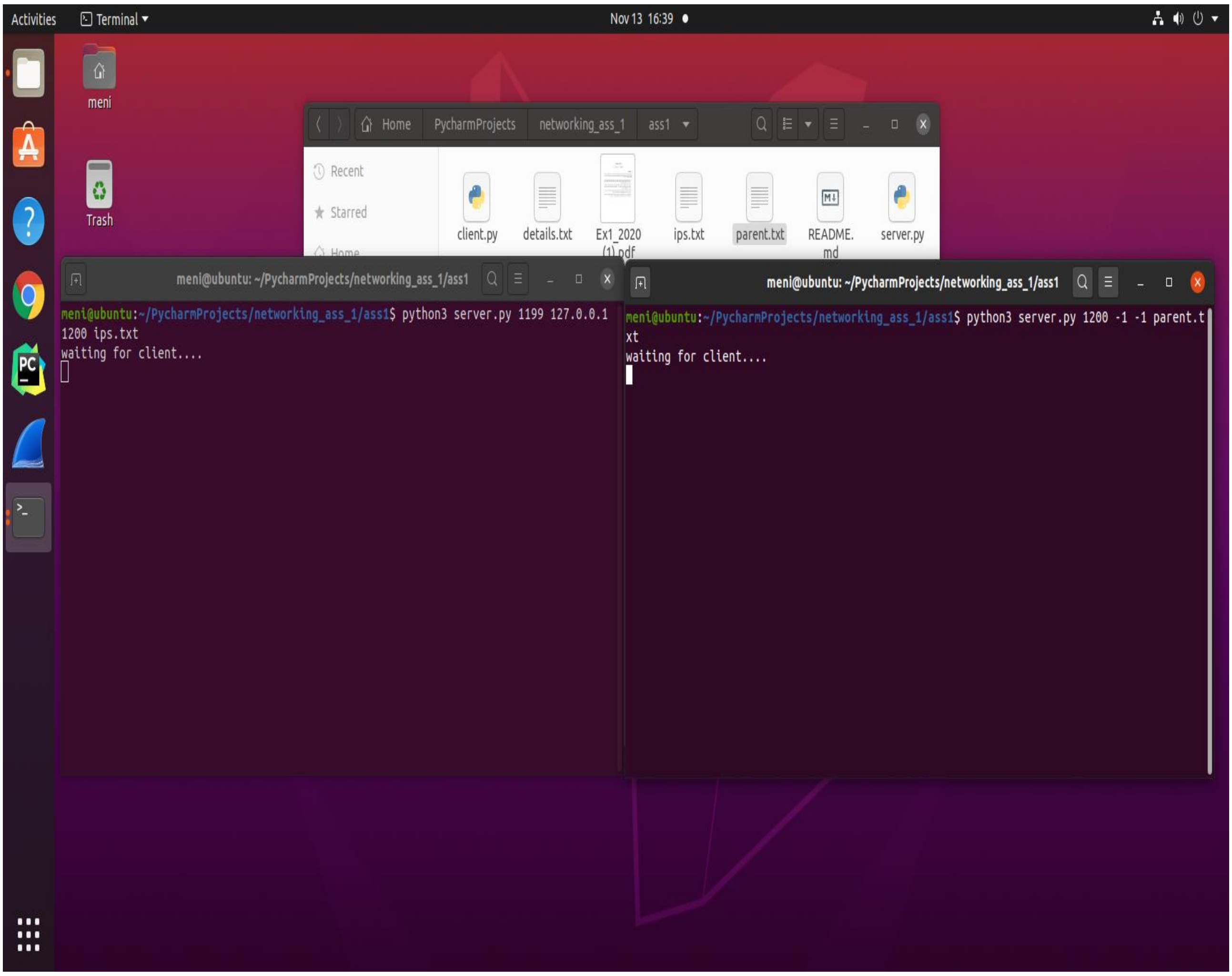
```
meni@ubuntu:~/PycharmProjects/networking_ass_1/ass1$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.20.2  netmask 255.255.254.0  broadcast 172.16.21.255
        inet6 fe80::c4[illegible]787b:e08e:3f4c  prefixlen 64  scopeid 0x20<link>
        inet6 fe80::204f:5424:5a4f:176  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:ad:94:2f  txqueuelen 1000  (Ethernet)
        RX packets 2707  bytes 3057228 (3.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1318  bytes 170820 (170.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 399  bytes 48787 (48.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 399  bytes 48787 (48.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

meni@ubuntu:~/PycharmProjects/networking_ass_1/ass1$ 
```

Let's run the server and its father on Server VM:

Command to run father server: python3 server.py 1200 -1 -1 parent.txt

Command to run original server: python3 server.py 1199 127.0.0.1 1200 ips.txt

Now they are working, screenshot:



```
meni@ubuntu:~/PycharmProjects/networking_ass_1/ass1$ python3 server.py 1199 127.0.0.1
1200 ips.txt
waiting for client....
```

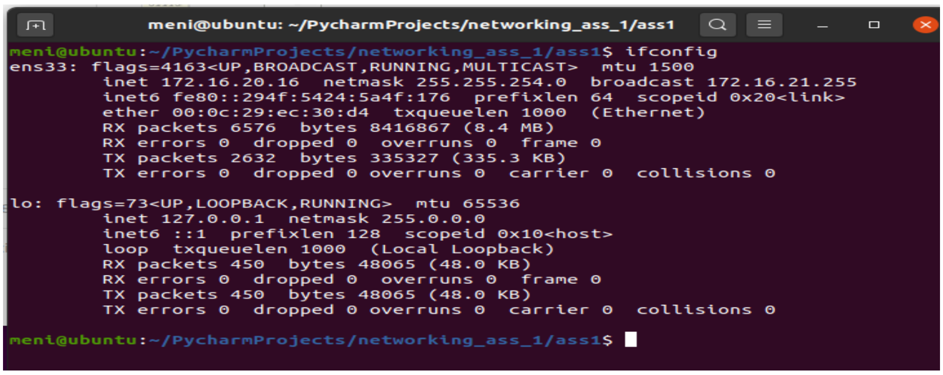
```
meni@ubuntu:~/PycharmProjects/networking_ass_1/ass1$ python3 server.py 1200 -1 -1 parent.t
xt
waiting for client....
```

Client VM:

Ip: 172.16.20.16

MAC: 00:0c:29:ec:30:d4

Print screen to show how I found these details:

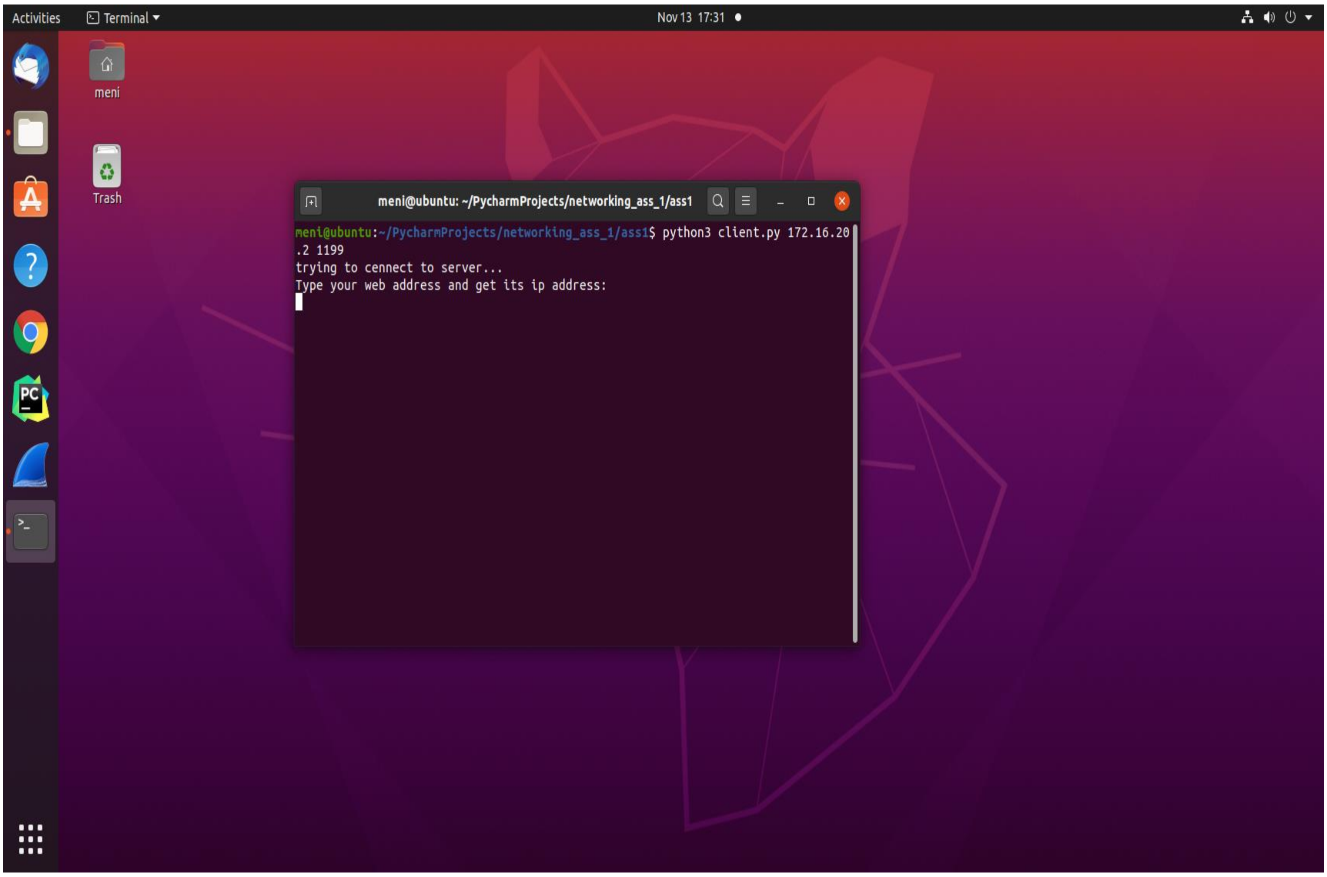


```
meni@ubuntu:~/PycharmProjects/networking_ass_1/ass1$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.20.16  netmask 255.255.254.0  broadcast 172.16.21.255
        inet6 fe80::294f:5424:5a4f:176  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:ec:30:d4  txqueuelen 1000  (Ethernet)
        RX packets 6576  bytes 8416867 (8.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2632  bytes 335327 (335.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 450  bytes 48065 (48.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 450  bytes 48065 (48.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

meni@ubuntu:~/PycharmProjects/networking_ass_1/ass1$ 
```

Let's run the client on Client VM:

Command to run client: python3 client.py 172.16.20.2 1199

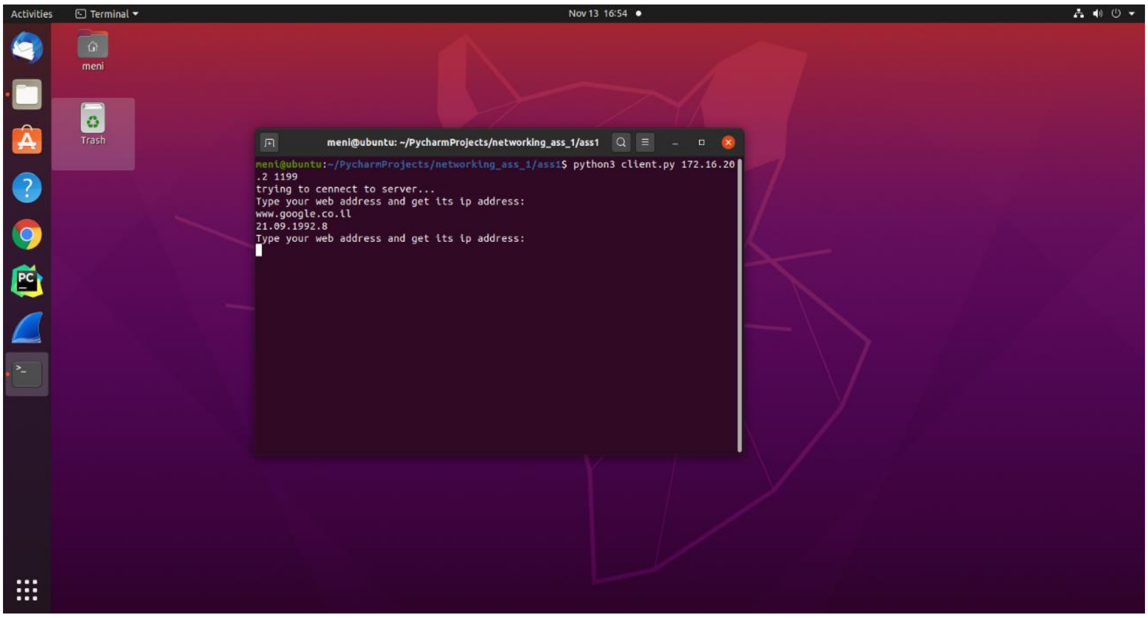
Screenshot:



```
meni@ubuntu:~/PycharmProjects/networking_ass_1/ass1$ python3 client.py 172.16.20
.2 1199
trying to cennect to server...
Type your web address and get its ip address:
```

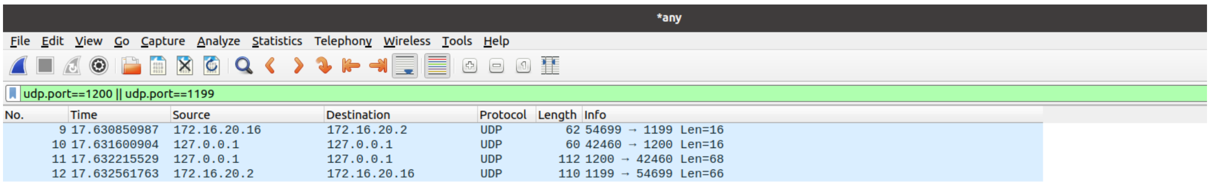
Now let's cut to the chase: try to send query to the server with an address that exists only in father server's file so the original server will have to ask for that info from the father server:

View from client:



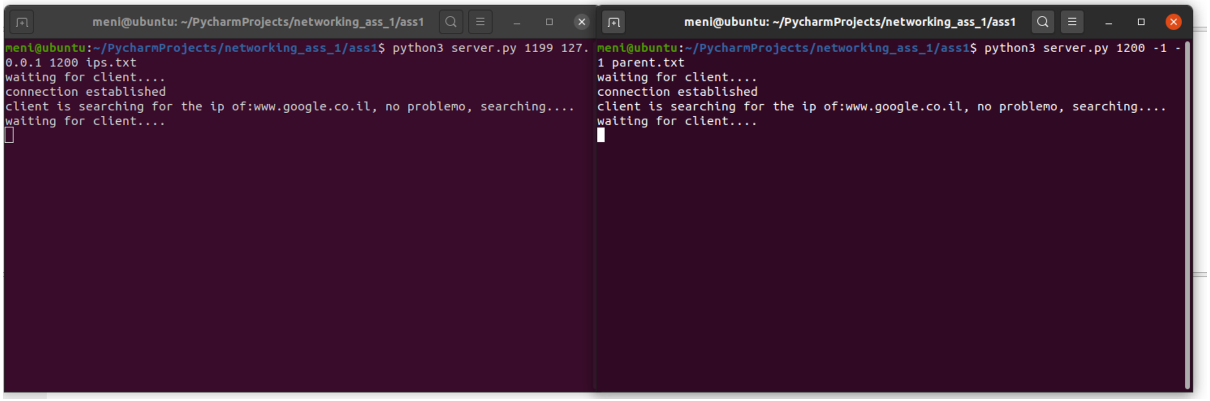
```
meni@ubuntu:~/PycharmProjects/networking_ass_1/ass1$ python3 client.py 172.16.20
.2 1199
trying to cennect to server...
Type your web address and get its ip address:
www.google.co.il
21.09.1992.8
Type your web address and get its ip address:
```

View from servers machine :


*any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

udp.port==1200 || udp.port==1199

```
No.   Time          Source         Destination    Protocol  Length  Info
    9 17.630850987  172.16.20.16   172.16.20.2    UDP           62  54699 → 1199 Len=16
   10 17.631600904  127.0.0.1      127.0.0.1      UDP           60  42460 → 1200 Len=16
   11 17.632215529  127.0.0.1      127.0.0.1      UDP          112  1200 → 42460 Len=68
   12 17.632561763  172.16.20.2    172.16.20.16   UDP          110  1199 → 54699 Len=66
```

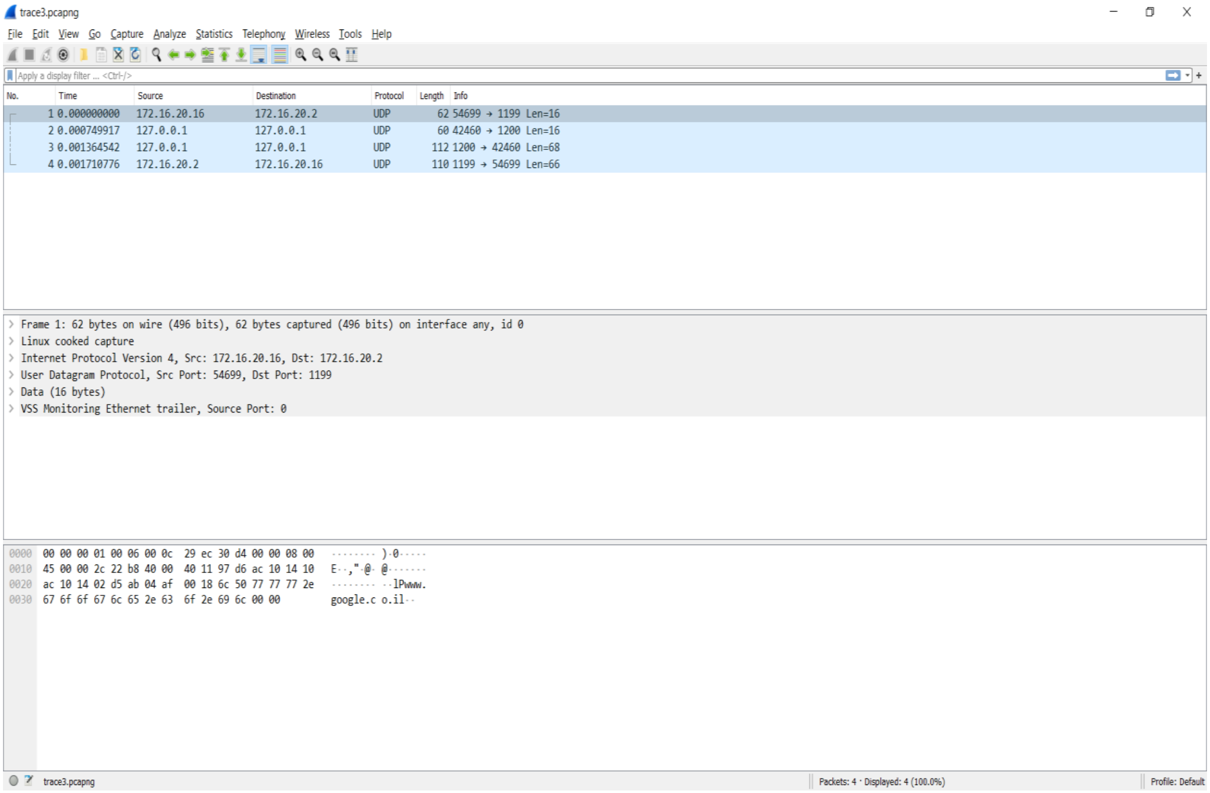

meni@ubuntu: ~/PycharmProjects/networking_ass_1/ass1
meni@ubuntu: ~/PycharmProjects/networking_ass_1/ass1

```
meni@ubuntu:~/PycharmProjects/networking_ass_1/ass1$ python3 server.py 1199 127.
0.0.1 1200 ips.txt
waiting for client....
connection established
client is searching for the ip of:www.google.co.il, no problemo, searching....
waiting for client....
```

```
meni@ubuntu:~/PycharmProjects/networking_ass_1/ass1$ python3 server.py 1200 -1 -
1 parent.txt
waiting for client....
connection established
client is searching for the ip of:www.google.co.il, no problemo, searching....
waiting for client....
```

To conclude: let's see the sniffing from wireshark and analyze the data from it:

Open the trace2.pcap file:


trace3.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time         Source        Destination   Protocol Length Info
   1 0.000000000  172.16.20.16  172.16.20.2   UDP          62 54699 → 1199 Len=16
   2 0.000749917  127.0.0.1     127.0.0.1     UDP          60 42460 → 1200 Len=16
   3 0.001364542  127.0.0.1     127.0.0.1     UDP         112 1200 → 42460 Len=68
   4 0.001710776  172.16.20.2   172.16.20.16  UDP         110 1199 → 54699 Len=66
```

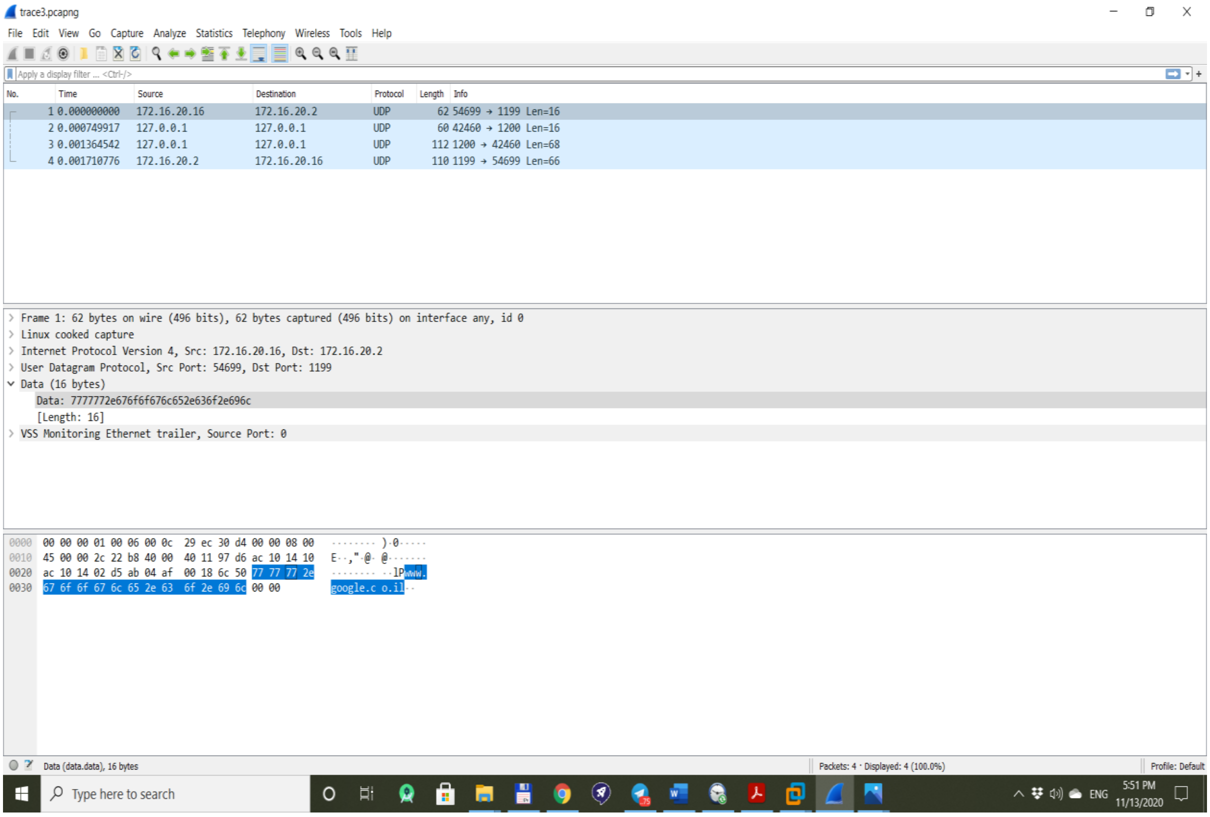
```
> Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface any, id 0
> Linux cooked capture
> Internet Protocol Version 4, Src: 172.16.20.16, Dst: 172.16.20.2
> User Datagram Protocol, Src Port: 54699, Dst Port: 1199
> Data (16 bytes)
> VSS Monitoring Ethernet trailer, Source Port: 0
```

```
0000  00 00 00 01 00 06 00 0c  29 ec 30 d4 00 00 08 00   ········ )·0·····
0010  45 00 00 2c 22 b8 40 00  40 11 97 d6 ac 10 14 10   E··,"·@· @·······
0020  ac 10 14 02 d5 ab 04 af  00 18 6c 50 77 77 77 2e   ········ ··lPwww.
0030  67 6f 6f 67 6c 65 2e 63  6f 2e 69 6c 00 00         google.c o.il··
```

Now I will show you how to data was sent from the client's machine to the server's machine to the destinated port – top to bottom and vice versa:

First the client sends data to the main server so I expect to see in the Application layer the data that was sent to the main server

First line indicates that socket opened between the original server and the client, at the Data tab we can see the string of the query



```
No.   Time          Source          Destination     Protocol  Length  Info
    1 0.000000000   172.16.20.16    172.16.20.2     UDP           62 54699 → 1199 Len=16
    2 0.000749917   127.0.0.1       127.0.0.1       UDP           60 42460 → 1200 Len=16
    3 0.001364542   127.0.0.1       127.0.0.1       UDP          112 1200 → 42460 Len=68
    4 0.001710776   172.16.20.2     172.16.20.16    UDP          110 1199 → 54699 Len=66
```

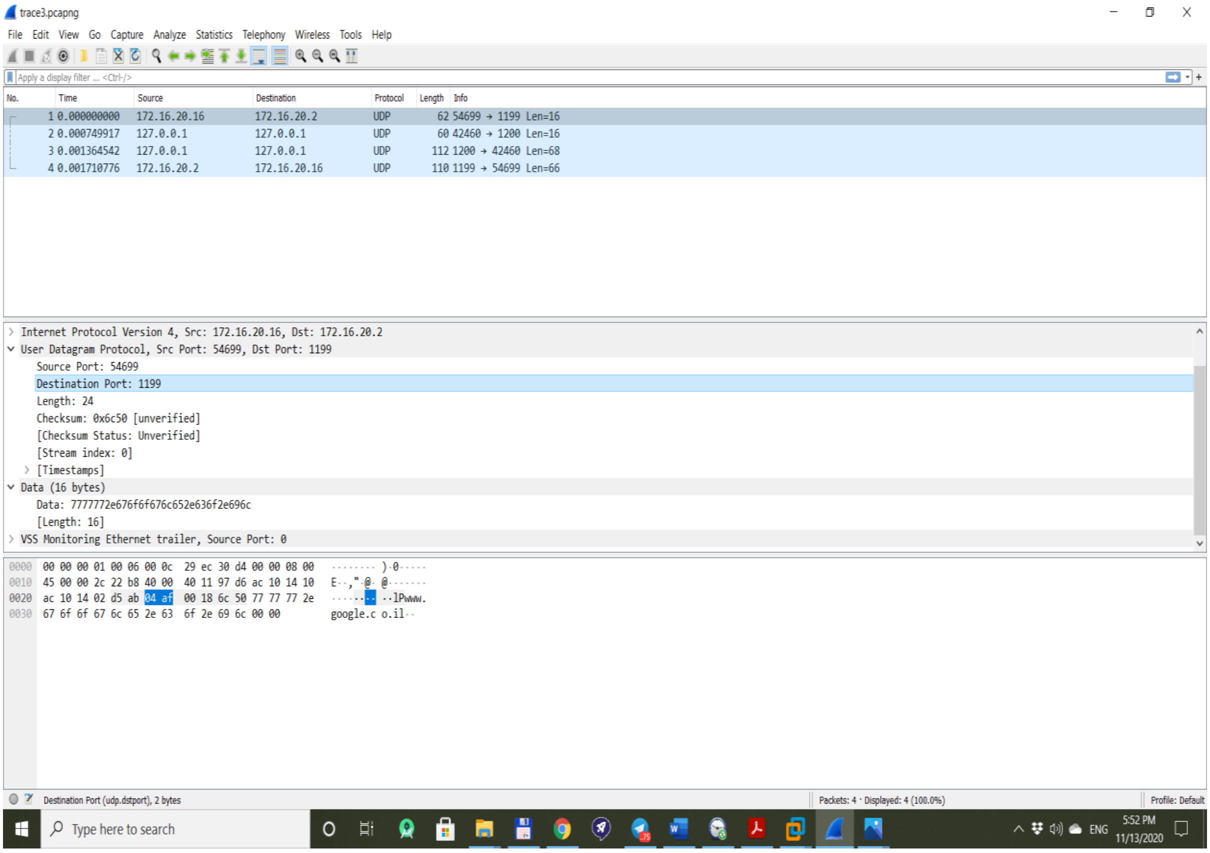
```
> Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface any, id 0
> Linux cooked capture
> Internet Protocol Version 4, Src: 172.16.20.16, Dst: 172.16.20.2
> User Datagram Protocol, Src Port: 54699, Dst Port: 1199
v Data (16 bytes)
    Data: 7777772e676f6f676c652e636f2e696c
    [Length: 16]
> VSS Monitoring Ethernet trailer, Source Port: 0
```

```
0000  00 00 00 01 00 06 00 0c  29 ec 30 d4 00 00 08 00   ········ )·0·····
0010  45 00 00 2c 22 b8 40 00  40 11 97 d6 ac 10 14 10   E··,"·@· @·······
0020  ac 10 14 02 d5 ab 04 af  00 18 6c 50 77 77 77 2e   ········ ··lPwww.
0030  67 6f 6f 67 6c 65 2e 63  6f 2e 69 6c 00 00         google.c o.il··
```

```
Data (data.data), 16 bytes          Packets: 4 · Displayed: 4 (100.0%)          Profile: Default
```

Now lets get to the lower tab – the Datagram tab which is related to the determination which network conversation protocol (TCP or UDP) is used for this conversation.

If you open the Datagram tab which is related to the transport layer you can see that the dest port is 1199 which is the port of our server program


trace3.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.     Time           Source          Destination     Protocol  Length Info
      1 0.000000000    172.16.20.16    172.16.20.2     UDP           62 54699 → 1199 Len=16
      2 0.000749917    127.0.0.1       127.0.0.1       UDP           60 42460 → 1200 Len=16
      3 0.001364542    127.0.0.1       127.0.0.1       UDP          112 1200 → 42460 Len=68
      4 0.001710776    172.16.20.2     172.16.20.16    UDP          110 1199 → 54699 Len=66
```

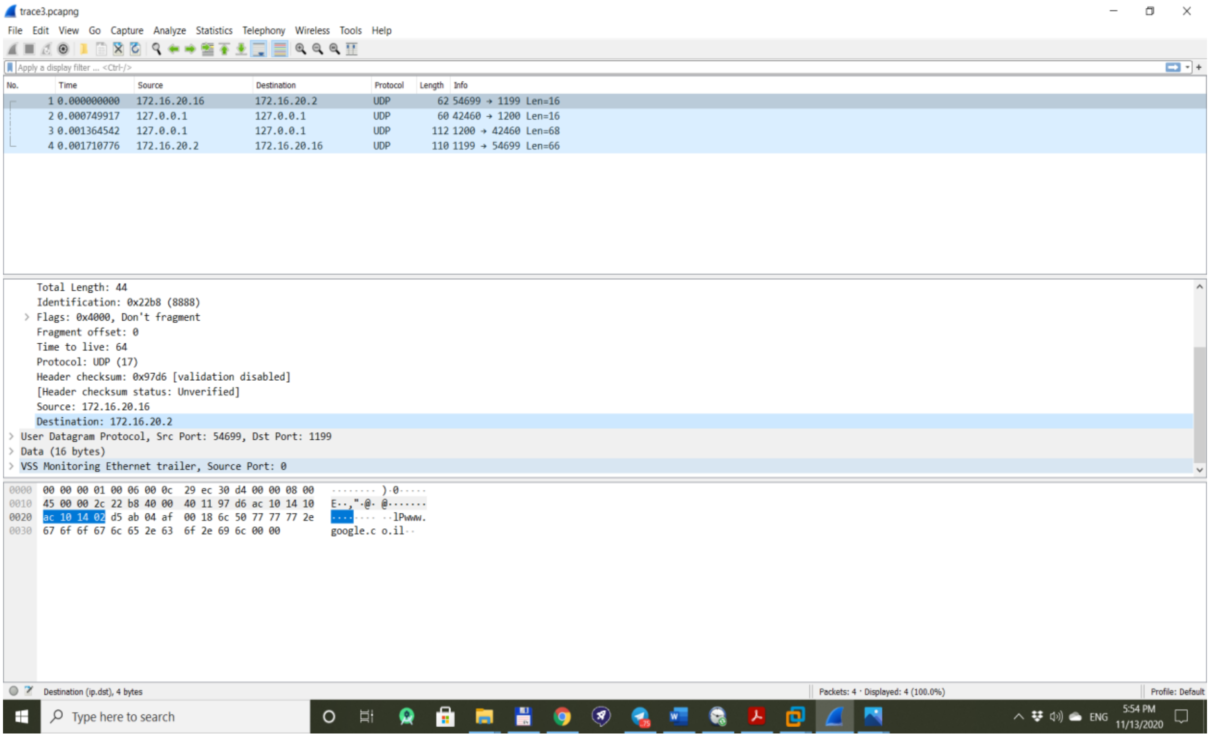
```
> Internet Protocol Version 4, Src: 172.16.20.16, Dst: 172.16.20.2
v User Datagram Protocol, Src Port: 54699, Dst Port: 1199
    Source Port: 54699
    Destination Port: 1199
    Length: 24
    Checksum: 0x6c50 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  > [Timestamps]
v Data (16 bytes)
    Data: 7777772e676f6f676c652e636f2e696c
    [Length: 16]
> VSS Monitoring Ethernet trailer, Source Port: 0
```

```
0000  00 00 00 01 00 06 00 0c  29 ec 30 d4 00 00 08 00   ········ )·0·····
0010  45 00 00 2c 22 b8 40 00  40 11 97 d6 ac 10 14 10   E··,"·@· @·······
0020  ac 10 14 02 d5 ab 04 af  00 18 6c 50 77 77 77 2e   ········ ··lPwww.
0030  67 6f 6f 67 6c 65 2e 63  6f 2e 69 6c 00 00         google.c o.il··
```

Destination Port (udp.dstport), 2 bytes
Packets: 4 · Displayed: 4 (100.0%)
Profile: Default
Type here to search
ENG
5:52 PM
11/13/2020

Now lets get lower to the internet tab and then we will see that the des tip is the ip of our main server machine and the source ip is the ip of the client's machine

If you open the Internet tab which is related to network layer you will see that the src ip is the ip of the client machine and the dest ip is the ip of the server machine

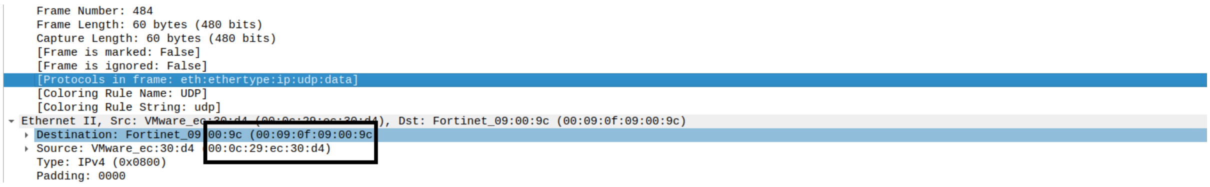

trace3.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Destination (ip.dst), 4 bytes
Packets: 4 · Displayed: 4 (100.0%)
Profile: Default
Type here to search
ENG
5:54 PM
11/13/2020

```
No.  Time         Source        Destination   Protocol Length Info
  1 0.000000000  172.16.20.16  172.16.20.2   UDP        62 54699 → 1199 Len=16
  2 0.000749917  127.0.0.1     127.0.0.1     UDP        60 42460 → 1200 Len=16
  3 0.001364542  127.0.0.1     127.0.0.1     UDP       112 1200 → 42460 Len=68
  4 0.001710776  172.16.20.2   172.16.20.16  UDP       110 1199 → 54699 Len=66
```

```
    Total Length: 44
    Identification: 0x22b8 (8888)
  > Flags: 0x4000, Don't fragment
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
    Header checksum: 0x97d6 [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.16.20.16
    Destination: 172.16.20.2
> User Datagram Protocol, Src Port: 54699, Dst Port: 1199
> Data (16 bytes)
> VSS Monitoring Ethernet trailer, Source Port: 0
```

```
0000  00 00 00 01 00 06 00 0c  29 ec 30 d4 00 00 08 00   ········ )·0·····
0010  45 00 00 2c 22 b8 40 00  40 11 97 d6 ac 10 14 10   E··,"·@· @·······
0020  ac 10 14 02 d5 ab 04 af  00 18 6c 50 77 77 77 2e   ········ ··lPwww.
0030  67 6f 6f 67 6c 65 2e 63  6f 2e 69 6c 00 00         google.c o.il··
```

Lets get lower to the lowest layer – the data link layer where I expect to see the Mac Addresses of the source, aka the client's machine, and the dest Mac Adress, aka the main server's machine

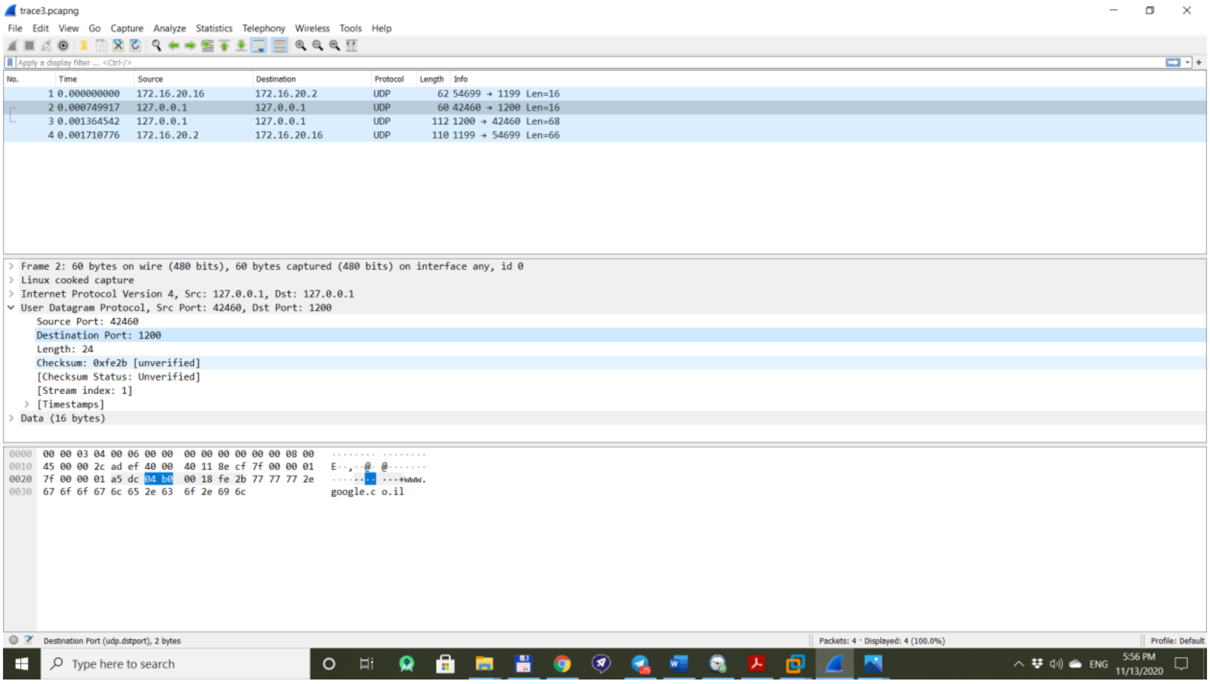


```
    Frame Number: 484
    Frame Length: 60 bytes (480 bits)
    Capture Length: 60 bytes (480 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:udp:data]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
▾ Ethernet II, Src: VMware_ec:30:d4 (00:0c:29:ec:30:d4), Dst: Fortinet_09:00:9c (00:09:0f:09:00:9c)
  ▸ Destination: Fortinet_09:00:9c (00:09:0f:09:00:9c)
  ▸ Source: VMware_ec:30:d4 (00:0c:29:ec:30:d4)
    Type: IPv4 (0x0800)
    Padding: 0000
```

Here we can see that the dst Mac marked is the Mac of the main server's machine and the src Mac is the Mac of the client's machine

That was how the data was transferred from the client to server's machine top to bottom and vice versa, in the next records in the wireshark sniff we will see the conversation between the main server and the father server and the response sent to the client

Move on to the next line in wireshark sniff – we can see that the original server didn't know the ip address of the given query so he goes to his father in order to get more info from it so in the Datagram tab we can see that the dest port is the port of the father server:


trace3.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Type here to search
ENG
5:56 PM
11/13/2020

```
No.   Time          Source          Destination     Protocol  Length  Info
    1 0.000000000   172.16.20.16    172.16.20.2     UDP           62 54699 → 1199 Len=16
    2 0.000749917   127.0.0.1       127.0.0.1       UDP           60 42460 → 1200 Len=16
    3 0.001364542   127.0.0.1       127.0.0.1       UDP          112 1200 → 42460 Len=68
    4 0.001710776   172.16.20.2     172.16.20.16    UDP          110 1199 → 54699 Len=66
```

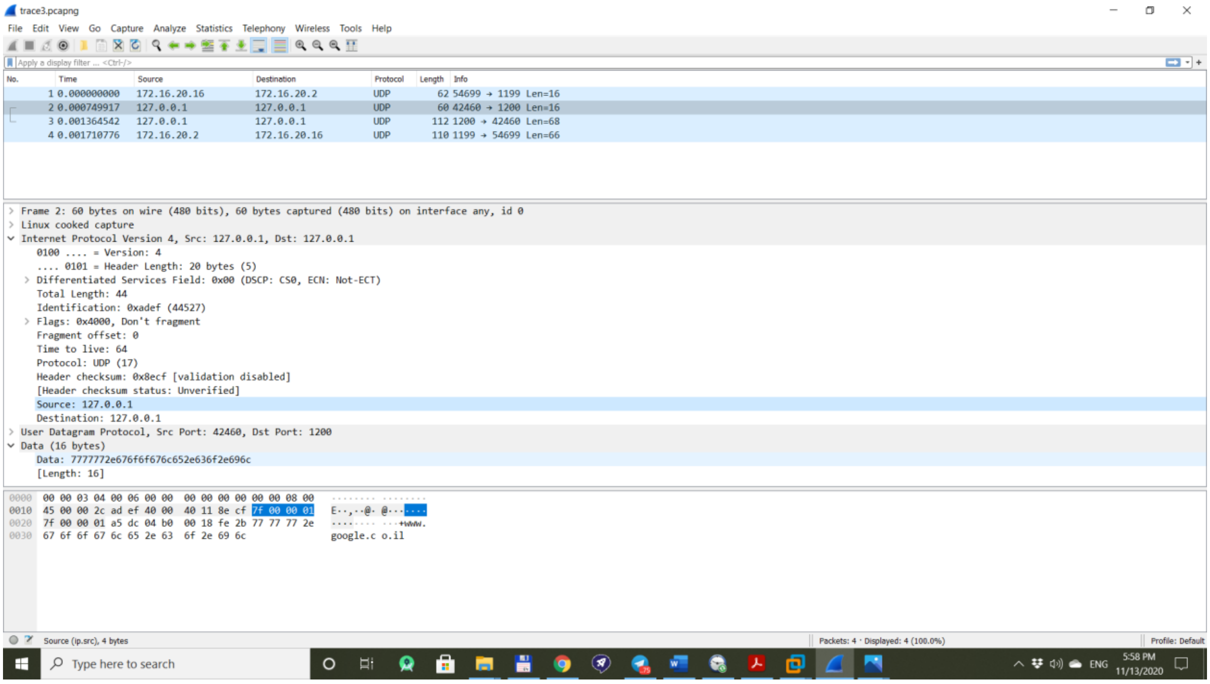
```
> Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface any, id 0
> Linux cooked capture
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v User Datagram Protocol, Src Port: 42460, Dst Port: 1200
    Source Port: 42460
    Destination Port: 1200
    Length: 24
    Checksum: 0xfe2b [unverified]
    [Checksum Status: Unverified]
    [Stream index: 1]
  > [Timestamps]
> Data (16 bytes)
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ········ ········
0010  45 00 00 2c ad ef 40 00  40 11 8e cf 7f 00 00 01   E··,··@· @·······
0020  7f 00 00 01 a5 dc 04 b0  00 18 fe 2b 77 77 77 2e   ········ ···+www.
0030  67 6f 6f 67 6c 65 2e 63  6f 2e 69 6c               google.c o.il
```

```
Destination Port (udp.dstport), 2 bytes        Packets: 4 · Displayed: 4 (100.0%)        Profile: Default
```

*the source port is some random that the OS on the Server VM gave the original server program after it asked to open new socket to the father server

In the Internet tab we can see that the src ip and des tip is the same: 127.0.0.1 – meaning that we work on the local machine :


trace3.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Type here to search
ENG
5:58 PM
11/13/2020

```
No.   Time          Source          Destination     Protocol  Length  Info
    1 0.000000000   172.16.20.16    172.16.20.2     UDP           62 54699 → 1199 Len=16
    2 0.000749917   127.0.0.1       127.0.0.1       UDP           60 42460 → 1200 Len=16
    3 0.001364542   127.0.0.1       127.0.0.1       UDP          112 1200 → 42460 Len=68
    4 0.001710776   172.16.20.2     172.16.20.16    UDP          110 1199 → 54699 Len=66
```

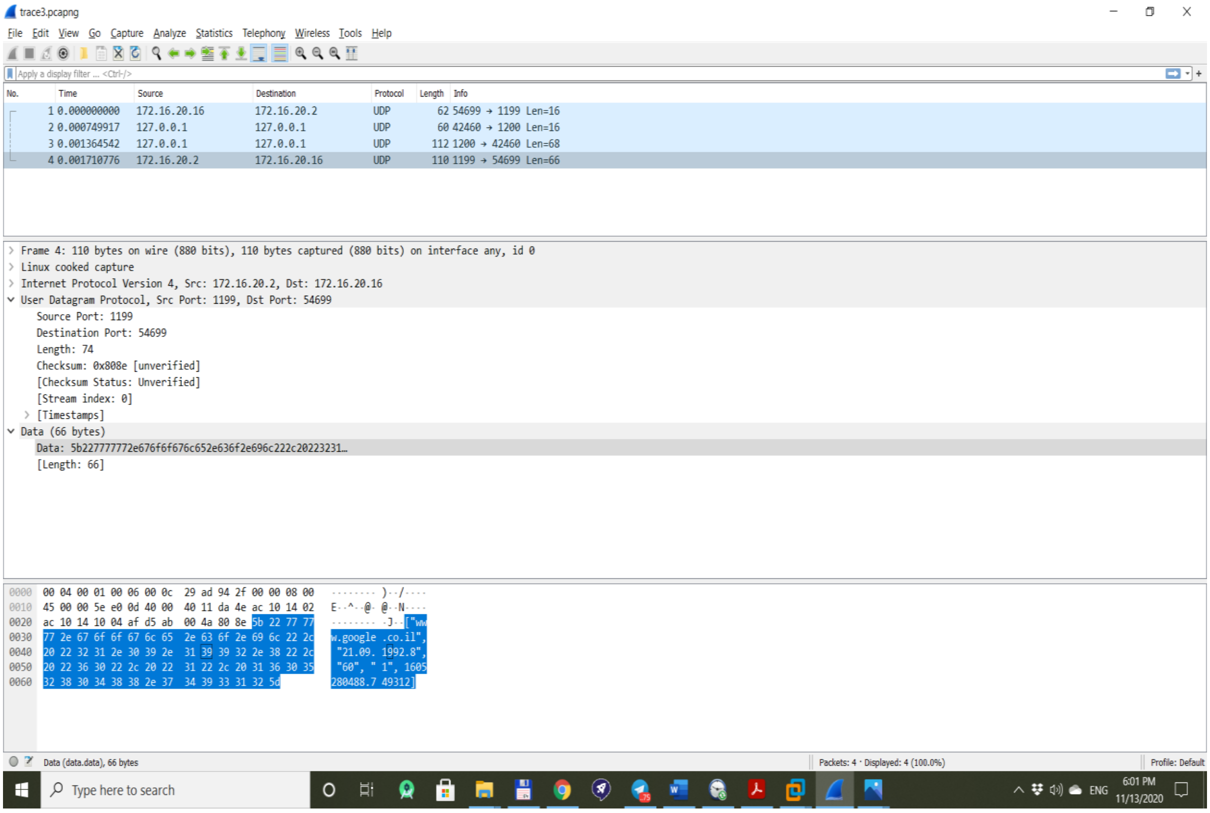
```
> Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface any, id 0
> Linux cooked capture
v Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 44
    Identification: 0xadef (44527)
  > Flags: 0x4000, Don't fragment
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
    Header checksum: 0x8ecf [validation disabled]
    [Header checksum status: Unverified]
    Source: 127.0.0.1
    Destination: 127.0.0.1
> User Datagram Protocol, Src Port: 42460, Dst Port: 1200
v Data (16 bytes)
    Data: 7777772e676f6f676c652e636f2e696c
    [Length: 16]
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ········ ········
0010  45 00 00 2c ad ef 40 00  40 11 8e cf 7f 00 00 01   E··,··@· @·······
0020  7f 00 00 01 a5 dc 04 b0  00 18 fe 2b 77 77 77 2e   ········ ···+www.
0030  67 6f 6f 67 6c 65 2e 63  6f 2e 69 6c               google.c o.il
```

```
Source (ip.src), 4 bytes        Packets: 4 · Displayed: 4 (100.0%)        Profile: Default
```

Finally, lets take a look at the last line which is the data sent from the server to the client:

Screenshot from the last line in the Data tab:


trace3.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Type here to search
ENG
6:01 PM
11/13/2020

```
No.   Time          Source          Destination     Protocol  Length  Info
    1 0.000000000   172.16.20.16    172.16.20.2     UDP           62  54699 → 1199 Len=16
    2 0.000749917   127.0.0.1       127.0.0.1       UDP           60  42460 → 1200 Len=16
    3 0.001364542   127.0.0.1       127.0.0.1       UDP          112  1200 → 42460 Len=68
    4 0.001710776   172.16.20.2     172.16.20.16    UDP          110  1199 → 54699 Len=66
```

```
> Frame 4: 110 bytes on wire (880 bits), 110 bytes captured (880 bits) on interface any, id 0
> Linux cooked capture
> Internet Protocol Version 4, Src: 172.16.20.2, Dst: 172.16.20.16
v User Datagram Protocol, Src Port: 1199, Dst Port: 54699
    Source Port: 1199
    Destination Port: 54699
    Length: 74
    Checksum: 0x808e [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  > [Timestamps]
v Data (66 bytes)
    Data: 5b227777772e676f6f676c652e636f2e696c222c20223231…
    [Length: 66]
```

```
0000  00 04 00 01 00 06 00 0c  29 ad 94 2f 00 00 08 00   ········ )··/····
0010  45 00 00 5e e0 0d 40 00  40 11 da 4e ac 10 14 02   E··^··@· @··N····
0020  ac 10 14 10 04 af d5 ab  00 4a 80 8e 5b 22 77 77   ········ ·J··["ww
0030  77 2e 67 6f 6f 67 6c 65  2e 63 6f 2e 69 6c 22 2c   w.google .co.il",
0040  20 22 32 31 2e 30 39 2e  31 39 39 32 2e 38 22 2c    "21.09. 1992.8",
0050  20 22 36 30 22 2c 20 22  31 22 2c 20 31 36 30 35    "60", " 1", 1605
0060  32 38 30 34 38 38 2e 37  34 39 33 31 32 5d         280488.7 49312]
```

Data (data.data), 66 bytes | Packets: 4 · Displayed: 4 (100.0%) | Profile: Default

Above we can see that the data sent back to the client was the full details about the address of www.google.co.il which is its web address,ip address and TTL.

We can conclude that the ip address of www.google.co.il is "21.09.1992.8" , and TTL is 60s.