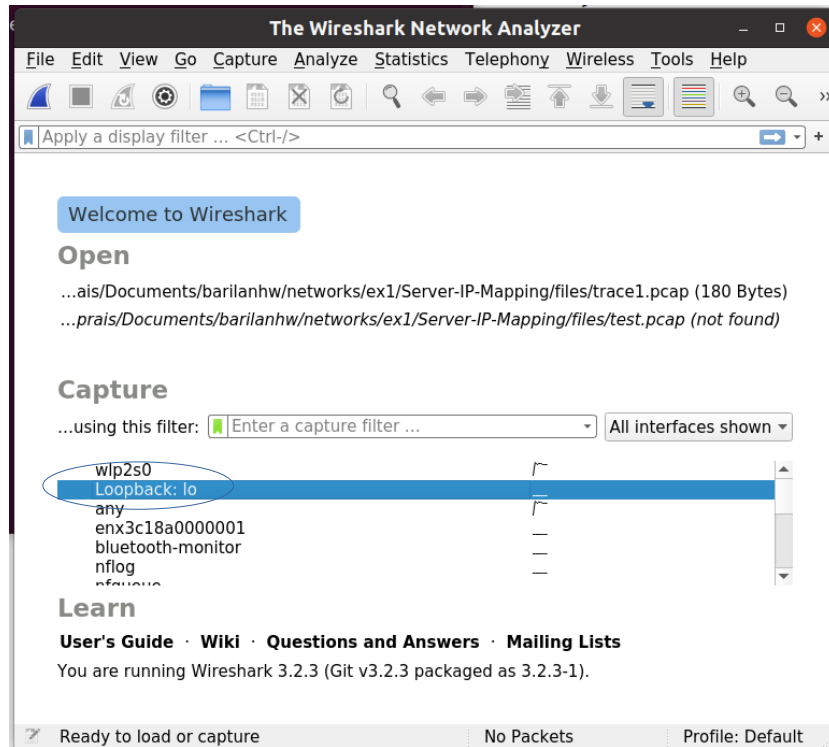


Networks - Assignment 1 Report

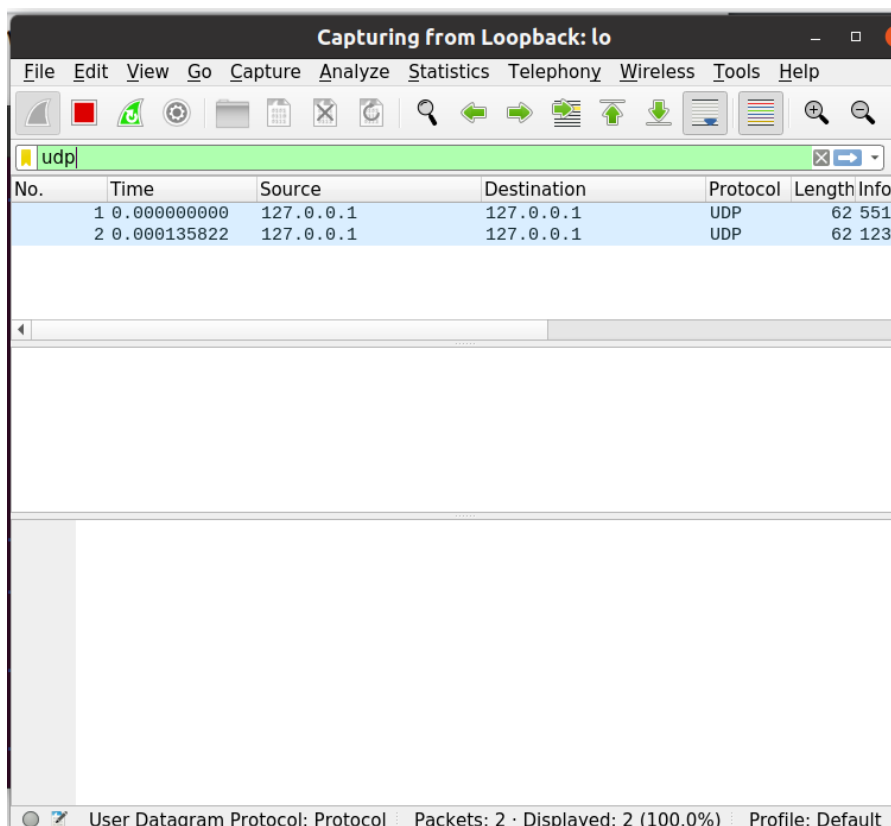
Part 1

How we Filtered the Packets

First, we opened the Wireshark and got into loopback to listen to the transport in the local machine.



We filtered the packets according to their specific protocol, in purpose to find the UDP packets, where our packets are exist. We found two packets - the packet that the client sent to the server, and the packet that the server sent to the client.



About the Port Numbers

The ports are numbers that specify the actual resource in the computer the message is sent to or received from. For example, in client-server architecture, the server is bound to a specific port which the client knows, so when the client sends a message to the server, he sends it to the server ip address, on the server port.

In the Wireshark, we can see two port numbers in the packet header - the first port is the sender port on his local machine, and the second port is the receiver port on his local machine (for example, in the packet that the client sent to the server, the first port is the client port, and the second port is the server port). The port logic is in the Transport layer.

The image shows the Wireshark network protocol analyzer interface. The title bar reads "Capturing from Loopback: lo". The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The toolbar contains various icons for file operations, capture control, and analysis. The packet list pane shows a list of captured packets. Packet 2 is selected, showing a UDP packet from 127.0.0.1 to 127.0.0.1. The packet details pane shows the structure of the selected packet: Ethernet II, Internet Protocol Version 4, and User Datagram Protocol. The UDP section is expanded, showing Source Port: 12345 and Destination Port: 55177. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	62	55177
2	0.000135822	127.0.0.1	127.0.0.1	UDP	62	12345
3	72.461066153	127.0.0.1	127.0.0.53	DNS	100	Standard query
4	72.493237037	127.0.0.53	127.0.0.1	DNS	100	Standard query response
5	75.522876125	127.0.0.1	127.0.0.53	DNS	86	Standard query
6	75.540453122	127.0.0.53	127.0.0.1	DNS	102	Standard query response

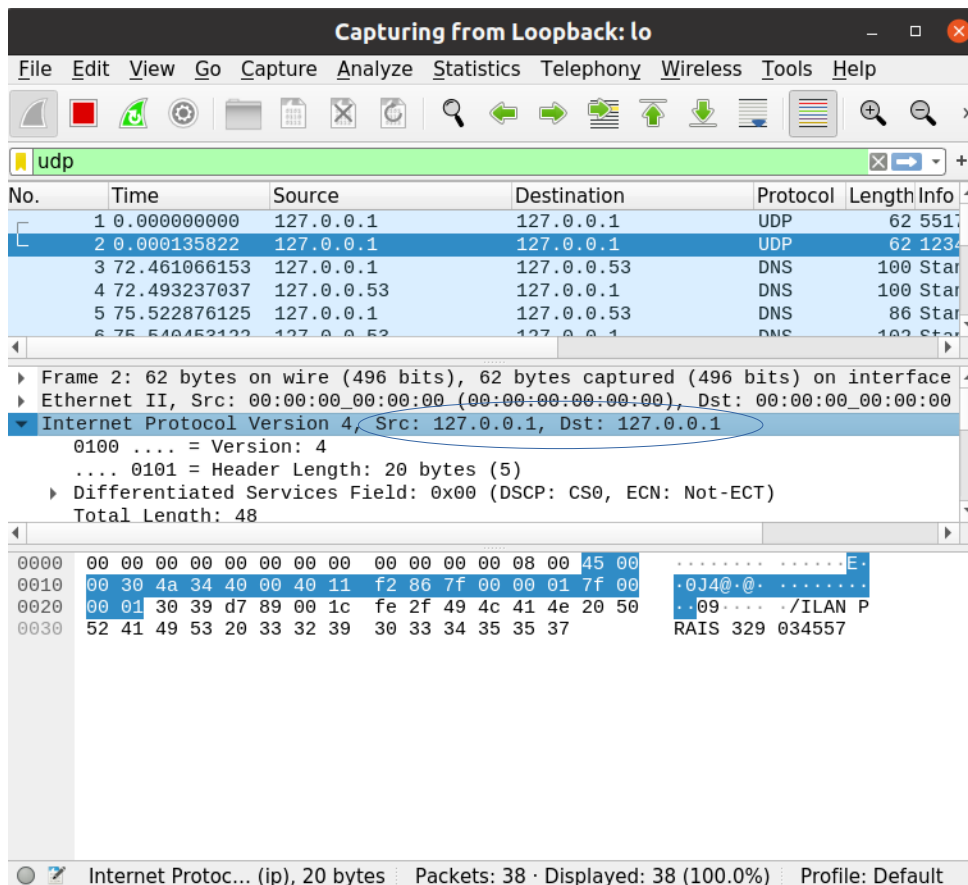
Frame 2: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 12345, Dst Port: 55177
Source Port: 12345
Destination Port: 55177
Length: 28

```
0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
0010  00 30 4a 34 40 00 40 11 f2 86 7f 00 00 01 7f 00  -0J4@-@-
0020  00 01 30 39 d7 89 00 1c fe 2f 49 4c 41 4e 20 50  --09-..-ILAN P
0030  52 41 49 53 20 33 32 39 30 33 34 35 35 37       RAIS 329 034557
```

User Datagram ...ocol: Protocol | Packets: 32 · Displayed: 32 (100.0%) | Profile: Default

About the IP Addresses

The first ip address in the packet header is the ip address of the sender, and the second ip address is the ip address of the receiver.



The image shows a Wireshark packet capture window titled "Capturing from Loopback: lo". The packet list shows a UDP packet (No. 2) with source and destination IP addresses of 127.0.0.1. The packet details pane shows the Internet Protocol Version 4 header with Source: 127.0.0.1 and Destination: 127.0.0.1. The packet bytes pane shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	62	5517
2	0.000135822	127.0.0.1	127.0.0.1	UDP	62	1234
3	72.461066153	127.0.0.1	127.0.0.53	DNS	100	Star
4	72.493237037	127.0.0.53	127.0.0.1	DNS	100	Star
5	75.522876125	127.0.0.1	127.0.0.53	DNS	86	Star
6	75.540452122	127.0.0.53	127.0.0.1	DNS	102	Star

Frame 2: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00
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: 48

0000 00 00 0a 34 40 00 00 11 f2 86 7f 00 00 01 7f 00E..
0010 00 30 4a 34 40 00 00 11 f2 86 7f 00 00 01 7f 00E..
0020 00 01 30 39 d7 89 00 1c fe 2f 49 4c 41 4e 20 50/ILAN P
0030 52 41 49 53 20 33 32 39 30 33 34 35 35 37RAIS 329 034557

As we can see, the ip addresses in the packet header are similar to the ip addresses of the local machine, that we found using the ifconfig command.

```
ilandovprais@ilandovprais-linux: ~/Documents/barilanhw/networks/...
enx3c18a0000001: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
ether 3c:18:a0:00:00:01 txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

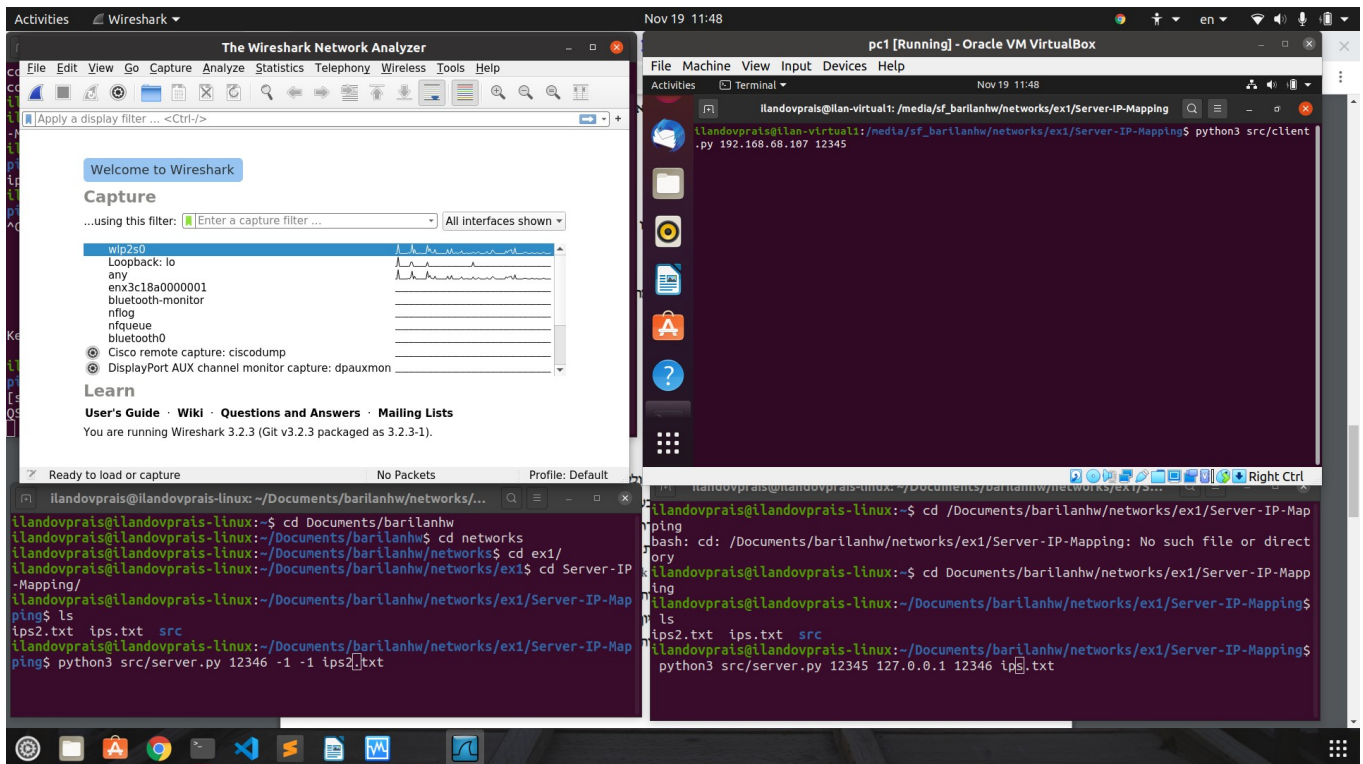
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 30536 bytes 3122523 (3.1 MB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 30536 bytes 3122523 (3.1 MB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.68.107 netmask 255.255.255.0 broadcast 192.168.68.255
inet6 fe80::a64c:df1d:31d1:d4af prefixlen 64 scopeid 0x20<link>
ether f8:94:c2:68:e6:83 txqueuelen 1000 (Ethernet)
RX packets 3626784 bytes 3694989141 (3.6 GB)
RX errors 0 dropped 42 overruns 0 frame 0
TX packets 1767863 bytes 552973870 (552.9 MB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

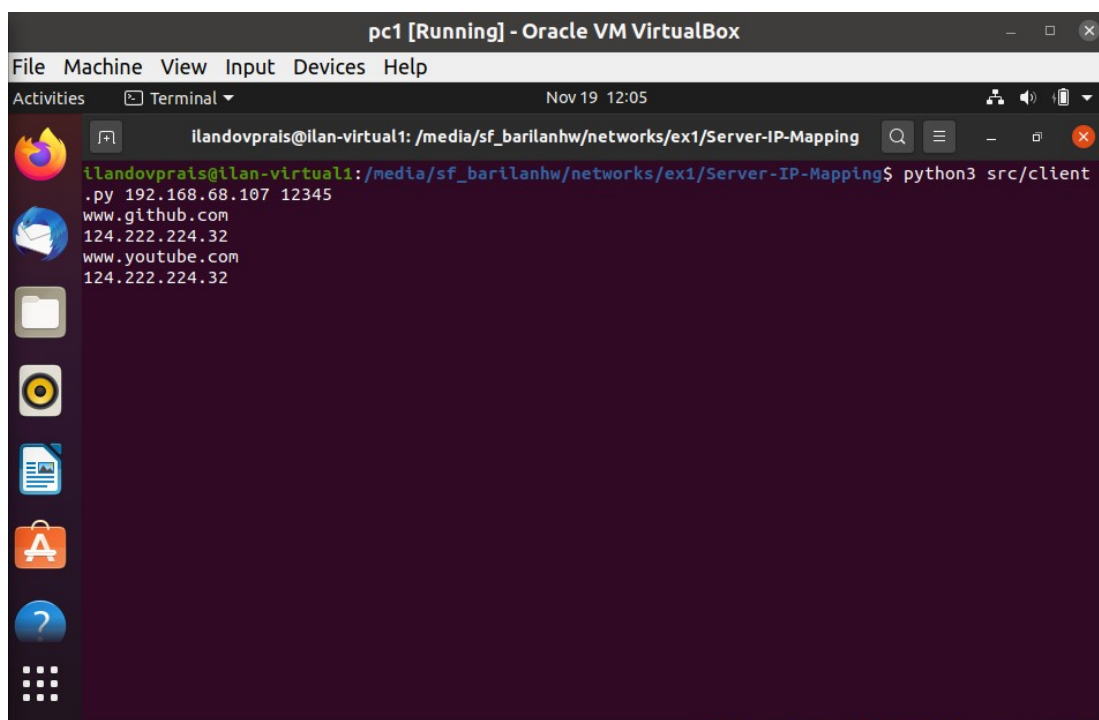
Part 2

Running the servers and the client

First, we opened 2 terminals in the host computer, 1 terminal for each server, and also started the virtual machine. Then we opened the parent server on 1 terminal, the child server on the other and the client on the virtual machine. We used ifconfig to find the ip adress of the server so the client could connect to it. Then we opened wireshark.



We opened the server and ran the client. First, we gave it the input "www.github.com" which the child server knows, so the ip was returned. Afterwards, the client requested the ip of "www.youtube.com", which the child does not have, but the parent does so the child requested it from his father and returned it to the client.



Capturing the packets in Wireshark

while running the servers and the client, we sniffed the packages using wireshark. Let's look at the first request: "www.github.com" -> "124.222.224.32"

No.	Time	Source	Destination	Protocol
1	0.000000	192.168.68.107	192.168.68.107	UDP
2	0.000126	192.168.68.107	192.168.68.107	UDP
3	24.702709	192.168.68.107	192.168.68.107	UDP
4	24.703012	192.168.68.107	192.168.68.107	UDP

▶ Frame 1: 58 bytes on wire (464 bits), 58 bytes captured (464 bits)
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 192.168.68.107, Dst: 192.168.68.107
▶ User Datagram Protocol, Src Port: 56878, Dst Port: 12345
▶ Data (14 bytes)

0000	00 00 03 04 00 06 00 00 00 00 00 00 00 06 08 00
0010	45 00 00 2a 00 00 40 00 3f 11 31 9c c0 a8 44 6b	E..*..@.?..1..Dk
0020	c0 a8 44 6b de 2e 30 39 00 16 0a 4f 77 77 77 2e	..Dk..09...0www.
0030	67 69 74 68 75 62 2e 63 6f 6d	github.c om

As we can see, "www.github.com" appears in the data in the application layer. In the transport layer, we can see that udp protocol was used, and the packet was sent from port 56878 on the virtual machine client to port 12345 on the server. In the network layer, we can see ipv4 is used, and that the package was sent from the virtual machine with ip 192.168.68.107 to the server, 192.168.68.107 (they have the same ip because the vm uses the host network). In the data link layer the mac is specified, and in the physical layer we can see that 58 bytes were sent in total.

Similarly, we captured the server's respond:

No.	Time	Source	Destination	Protocol
1	0.000000	192.168.68.107	192.168.68.107	UDP
2	0.000126	192.168.68.107	192.168.68.107	UDP
3	24.702709	192.168.68.107	192.168.68.107	UDP
4	24.703012	192.168.68.107	192.168.68.107	UDP

▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 192.168.68.107, Dst: 192.168.68.107
▶ User Datagram Protocol, Src Port: 12345, Dst Port: 56878
▶ Data (34 bytes)

Data: 7777772e6769746875622e636f6d2c3132342e3232322e32...
[Length: 34]

0000	00 00 03 04 00 06 00 00 00 00 00 00 00 00 08 00
0010	45 00 00 3e 6b 63 40 00 40 11 c5 24 c0 a8 44 6b	E..>kc@..@..\$.Dk
0020	c0 a8 44 6b 30 39 de 2e 00 2a 0a 63 77 77 77 2e	..Dk09..*..cwww.
0030	67 69 74 68 75 62 2e 63 6f 6d 2c 31 32 34 2e 32	github.c om,124.2
0040	32 32 2e 32 32 34 2e 33 32 2c 33 30 30 30 30	22.224.3 2,3000

The packet is similar to the previous one, however this time the src and dest port and ip are swapped, and the message is the domain with it's ip and ttl.

Next, we capture the second request - where the server had to request the IP from his parent server

The screenshot shows the Wireshark interface with the following details:

- Packet List:**

No.	Time	Source	Destination	Protocol
5	62.519637	192.168.68.107	192.168.68.107	UDP
6	62.519742	127.0.0.1	127.0.0.1	UDP
7	62.519821	127.0.0.1	127.0.0.1	UDP
8	62.520014	192.168.68.107	192.168.68.107	UDP
- Packet Details (Frame 5):**
 - Linux cooked capture
 - Internet Protocol Version 4, Src: 192.168.68.107, Dst: 192.168.68.107
 - User Datagram Protocol, Src Port: 56878, Dst Port: 12345
 - Data (14 bytes)
- Packet Bytes:**

```

0000  00 00 03 04 00 06 00 00 00 00 00 00 7c d2 08 00  ....*..@..?..1..Dk
0010  45 00 00 2a 00 00 40 00 3f 11 31 9c c0 a8 44 6b  ..Dk..09...0www.
0020  c0 a8 44 6b de 2e 30 39 00 16 0a 4f 77 77 77 2e  google.c om
0030  67 6f 6f 67 6c 65 2e 63 6f 6d

```

As we can see, this time there are 4 packets - the client request, the child server request, the parent response, and then the child server response.

(packet 5) The client request was sent from ip 192.168.68.107 port 56878 to the child server, at 192.168.68.107 port 12345.

(packet 6) The child server did not have the requested ip in it's ips.txt file, so it requested the ip from his parent server- and as they are on the same machine, the request was sent from and to ip 127.0.0.1 (but different ports).

```
(packet 7) The parent server had the requested ip, so it sent it to the child
server, again at ip 127.0.0.1.
```

```
(packet 8) Finally, the child server had the requested ip so it sent it to the client.
```

No.	Time	Source	Destination	Protocol
5	62.519637	192.168.68.107	192.168.68.107	UDP
6	62.519742	127.0.0.1	127.0.0.1	UDP
7	62.519821	127.0.0.1	127.0.0.1	UDP
8	62.520014	192.168.68.107	192.168.68.107	UDP

▶ Frame 7: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
 ▶ Linux cooked capture
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 ▶ User Datagram Protocol, Src Port: 12346, Dst Port: 36658
 ▶ Data (30 bytes)

0000	00 00 03 04 00 06 00 00	00 00 00 00 b5 d7 08 00
0010	45 00 00 3a bf 28 40 00	40 11 7d 88 7f 00 00 01	E...: (@ @ }
0020	7f 00 00 01 30 3a 8f 32	00 26 fe 39 77 77 77 2e	...0: 2 & 9www.
0030	67 6f 6f 67 6c 65 2e 63	6f 6d 2c 31 31 30 2e 33	google.c om, 110.3
0040	33 2e 32 2e 37 2c 31 30	30 30	3.2.7, 10 00