Isfahan University of technology

second subproject of computer networking

Professor:
Mr. Heydarpour

Teaching assistance:
Mr. Saeedi

Prepared by
Arefe Pour Mohammadi

1401-1402

Question 1

When I connect to iut.ac.ir, as well as DNS find the IP address of iut.ac.ir, it translates more URLs to their IPs, because there are some web objects URLs in the base HTML file of iut.ac.ir, and DNS should translate them to prepare the page.

udp.port == 53

No.	Time	Source	Destination	Protocol	Length	Info
98	10.695585124	192.168.233.73	192.168.233.243	DNS	103	Standard query response 0x2e19 A
139	11.463112540	192.168.233.243	192.168.233.73	DNS	103	Standard query 0xd37c AAAA prod.
140	11.467818547	192.168.233.73	192.168.233.243	DNS	103	Standard query response 0xd37c A
210	14.972460961	192.168.233.243	192.168.233.73	DNS	88	Standard query 0xea41 A contile.
211	14.972600308	192.168.233.243	192.168.233.73	DNS	88	Standard query 0x9024 AAAA conti
215	14.994837068	192.168.233.73	192.168.233.243	DNS	169	Standard query response 0x9024 A
216	15.000023035	192.168.233.73	192.168.233.243	DNS	104	Standard query response 0xea41 A
224	15.059144971	192.168.233.243	192.168.233.73	DNS	73	Standard query 0x9b8a AAAA ipv4o
226	15.063099049	192.168.233.73	192.168.233.243	DNS	73	Standard query response 0x9b8a A
233	15.130579604	192.168.233.243	192.168.233.73	DNS	95	Standard query 0xc6cb A content-
234	15.130755671	192.168.233.243	192.168.233.73	DNS	95	Standard query 0xb736 AAAA conte
239	15.161134100	192.168.233.73	192.168.233.243	DNS	249	Standard query response 0xc6cb A
240	15.163283789	192.168.233.243	192.168.233.73	DNS	74	Standard query 0xe59d A r3.o.len
241	15.163399066	192.168.233.243	192.168.233.73	DNS	74	Standard query 0xea69 AAAA r3.o.
242	15.166997178	192.168.233.73	192.168.233.243	DNS	176	Standard query response 0xe59d A
243	15.177428065	192.168.233.243	192.168.233.73	DNS	75	Standard query 0x91ae A yekta.iu
244	15.177552777	192.168.233.243	192.168.233.73	DNS	77	Standard query 0x0d2e A webauth.
245	15.177645323	192.168.233.243	192.168.233.73	DNS	75	Standard query 0x95a3 AAAA yekta
246	15.177742992	192.168.233.243	192.168.233.73	DNS	77	Standard query 0xb026 AAAA webau
247	15.185791320	192.168.233.73	192.168.233.243	DNS	261	Standard query response 0xb736 A
250	15.186654593	192.168.233.243	192.168.233.73	DNS	69	Standard query 0x24ae A iut.ac.i
251	15.186749786	192.168.233.243	192.168.233.73	DNS	69	Standard query 0x7e01 AAAA iut.a
265	15.403586825	192.168.233.243	192.168.233.73	DNS	97	Standard query 0xd600 A firefox.
268	15.403925014	192.168.233.243	192.168.233.73	DNS	97	Standard query 0x7bd8 AAAA firef
269	15.407506165	192.168.233.73	192.168.233.243	DNS	197	Standard query response 0xea69 A
272	15.427721263	192.168.233.73	192.168.233.243	DNS	113	Standard query response 0xd600 A
273	15.435743989	192.168.233.73	192.168.233.243	DNS	181	Standard query response 0x7bd8 A

Fig. 1 DNS packets when searching iut.ac.ir in the browser

Question 2

As it is obvious in the following image, DNS uses UDP to make connections.

```
▸ Internet Protocol Version 4, Src: 192.168.233.243, Dst: 192.168.233.73
▾ User Datagram Protocol, Src Port: 49296, Dst Port: 53
    Source Port: 49296
    Destination Port: 53
    Length: 39
    Checksum: 0x54c7 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 4]
  ▸ [Timestamps]
    UDP payload (31 bytes)
▸ Domain Name System (query)
```

Fig.2 Inside of a DNS packet. UDP is used in DNS packets.

As you saw in Fig.1, there are just two IP addresses! maybe you are wondering why? 192.168.233.243 IP address is my IP (see Fig. 3) and 192.168.233.73 IP address is the router's IP (see Fig. 4). The router which I connect to, is also the local DNS server(Fig. 5, my local DNS server IP and my routers are the same.) and as you know, my DNS query in the first step is sent to the local DNS server. The local DNS server finds the matched IP and sends it to me. So may you conclude there is no need for more than these two IPs, and that's right.

```
2: wlp0s20f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 84:c5:a6:a8:22:3a brd ff:ff:ff:ff:ff:ff
    inet 192.168.233.243/24 brd 192.168.233.255 scope global dynamic noprefixroute wlp0s20f3
       valid_lft 3592sec preferred_lft 3592sec
    inet6 fe80::d043:e0a0:d5f:c8dd/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Fig. 3 My IP address is 192.168.233.243

```
arefe@AREFE:~$ ip route
default via 192.168.233.73 dev wlp0s20f3 proto dhcp src 192.168.233.243 metric 600
```

Fig. 4 Router's IP is 192.168.233.73

```
arefe@AREFE:~$ resolvectl status
Global
       Protocols: -LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
resolv.conf mode: stub

Link 2 (wlp0s20f3)
    Current Scopes: DNS
         Protocols: +DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
Current DNS Server: 192.168.233.73
       DNS Servers: 192.168.233.73

Link 3 (ovs-system)
Current Scopes: none
     Protocols: -DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported

Link 4 (s1)
Current Scopes: none
     Protocols: -DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
arefe@AREFE:~$
```

Fig. 5, my local DNS server IP and my routers are the same.

Question 3

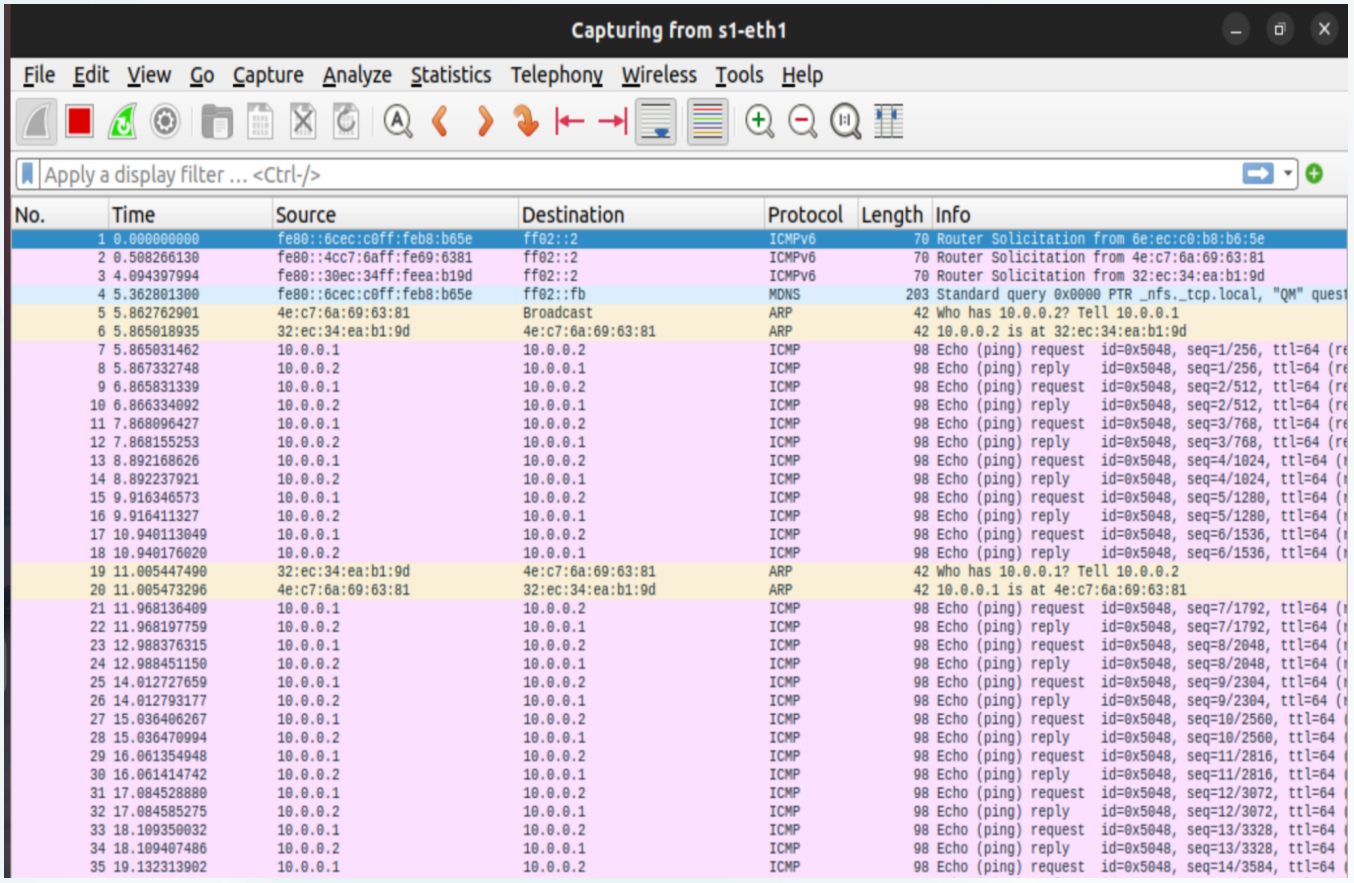

Fig. 6, Packets traveling in s1-eth1 link after h1 pings h2.

```
arefe@AREFE:~$ sudo mn
[sudo] password for arefe:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.44 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.581 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.113 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.100 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.099 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.110 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.111 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.132 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.111 ms
```

Fig. 7, h1 ping h2

Question 4

```
arefe@AREFE:~$ locate resolv.conf
/etc/.resolv.conf.systemd-resolved.bak
/etc/resolv.conf
/home/arefe/qt/qt-everywhere-src-6.5.0/coin/provisioning/common/linux/qnx_qemu_build_files/local/misc_files/etc/
resolv.conf
/snap/core18/2721/etc/resolv.conf
/snap/core18/2721/etc/cloud/templates/resolv.conf.tmpl
/snap/core18/2721/lib/systemd/resolv.conf
/snap/core18/2745/etc/resolv.conf
/snap/core18/2745/etc/cloud/templates/resolv.conf.tmpl
/snap/core18/2745/lib/systemd/resolv.conf
/snap/core20/1879/etc/resolv.conf
/snap/core20/1879/etc/cloud/templates/resolv.conf.tmpl
/snap/core20/1879/run/systemd/resolve/stub-resolv.conf
/snap/core20/1879/usr/lib/systemd/resolv.conf
/snap/core20/1891/etc/resolv.conf
/snap/core20/1891/etc/cloud/templates/resolv.conf.tmpl
/snap/core20/1891/run/systemd/resolve/stub-resolv.conf
/snap/core20/1891/usr/lib/systemd/resolv.conf
/snap/core22/617/etc/resolv.conf
/snap/core22/617/etc/cloud/templates/resolv.conf.tmpl
/snap/core22/617/run/systemd/resolve/stub-resolv.conf
/snap/core22/617/usr/lib/systemd/resolv.conf
/snap/core22/634/etc/resolv.conf
/snap/core22/634/etc/cloud/templates/resolv.conf.tmpl
/snap/core22/634/run/systemd/resolve/stub-resolv.conf
```

Fig. 8 Locate resolv.conf

```
arefe@AREFE:~$ cat /etc/resolv.conf
# This is /run/systemd/resolve/stub-resolv.conf managed by man:systemd-resolved(8).
# Do not edit.
#
# This file might be symlinked as /etc/resolv.conf. If you're looking at
# /etc/resolv.conf and seeing this text, you have followed the symlink.
#
# This is a dynamic resolv.conf file for connecting local clients to the
# internal DNS stub resolver of systemd-resolved. This file lists all
# configured search domains.
#
# Run "resolvectl status" to see details about the uplink DNS servers
# currently in use.
#
# Third party programs should typically not access this file directly, but only
# through the symlink at /etc/resolv.conf. To manage man:resolv.conf(5) in a
# different way, replace this symlink by a static file or a different symlink.
#
# See man:systemd-resolved.service(8) for details about the supported modes of
# operation for /etc/resolv.conf.

nameserver 127.0.0.53
options edns0 trust-ad
search .
```

Fig. 9 cat /etc/resolv.conf

nameserver 127.0.0.53:

127.0.0.53 is a DNS server IP address that is used by the OS to name domains. It is set as a default DNS server. When a user searches for a URL, first, the OS asks the DNS server to give the IP of the relative domain name and then sends its request to that IP.

/usr/lib/systemd/resolv.conf:

It is a configuration file that is used by the system for setting the DNS system. It consists of the IP addresses of DNS servers, which are used for naming domains. Using this file ensures that we connect to DNS servers correctly.

Question 5

udp.port == 53

No.	Time	Source	Destination	Protocol	Length	Info
1667	13.243729835	185.51.200.2	192.168.233.243	DNS	114	Standard query response 0xc7be AA
1765	13.465464674	192.168.233.243	185.51.200.2	DNS	92	Standard query 0xa909 A cdn.freev
1779	13.504504432	185.51.200.2	192.168.233.243	DNS	97	Standard query response 0xa909 A
1780	13.504750906	192.168.233.243	185.51.200.2	DNS	92	Standard query 0x5601 AAAA cdn.fr
1795	13.540242935	185.51.200.2	192.168.233.243	DNS	97	Standard query response 0x5601 AA
1812	13.563173942	192.168.233.243	185.51.200.2	DNS	84	Standard query 0x0a53 A ocsp.pki.
1813	13.563208299	192.168.233.243	185.51.200.2	DNS	84	Standard query 0x1068 AAAA ocsp.p
1827	13.607020417	185.51.200.2	192.168.233.243	DNS	100	Standard query response 0x0a53 A
1828	13.616091708	185.51.200.2	192.168.233.243	DNS	84	Standard query response 0x1068 AA
1852	13.867339147	192.168.233.243	185.51.200.2	DNS	84	Standard query 0xe83b A ocsp.pki.
1856	13.912256063	185.51.200.2	192.168.233.243	DNS	100	Standard query response 0xe83b A
1889	14.452442680	192.168.233.243	185.51.200.2	DNS	85	Standard query 0x3faa A www.apara
1890	14.452464857	192.168.233.243	185.51.200.2	DNS	85	Standard query 0x70ae AAAA www.ap
1897	14.526683394	185.51.200.2	192.168.233.243	DNS	149	Standard query response 0x3faa A
1898	14.526683660	185.51.200.2	192.168.233.243	DNS	141	Standard query response 0x70ae AA
1899	14.545033153	192.168.233.243	185.51.200.2	DNS	98	Standard query 0xbc0d A static.cd
1900	14.545056286	192.168.233.243	185.51.200.2	DNS	98	Standard query 0x8609 AAAA static
1903	14.581845553	185.51.200.2	192.168.233.243	DNS	114	Standard query response 0xbc0d A
1904	14.597921831	185.51.200.2	192.168.233.243	DNS	154	Standard query response 0x8609 AA
1921	14.762482861	192.168.233.243	185.51.200.2	DNS	95	Standard query 0xf7fc A dvcasha2.
1922	14.762505285	192.168.233.243	185.51.200.2	DNS	95	Standard query 0x2081 AAAA dvcash
1924	14.814477962	185.51.200.2	192.168.233.243	DNS	239	Standard query response 0xf7fc A
1925	14.814478415	185.51.200.2	192.168.233.243	DNS	263	Standard query response 0x2081 AA

Fig. 10 DNS packets after adding shekans IP

As is evident from Fig. 9, the router's IP changed to Shecan's IP(Shecan has two IPs, I used the second IP), because after I added Shecan's IP to resolv.conf , DNS requests are sent to Shekans local DNS server. By doing this all DNS requests are sent to Shecan and then Shecan sends them to the DNS servers.

```
arefe@AREFE:~$ cat /etc/resolv.conf
# This is /run/systemd/resolve/stub-
# Do not edit.
#
# This file might be symlinked as /e
# /etc/resolv.conf and seeing this t
#
# This is a dynamic resolv.conf file
# internal DNS stub resolver of syst
# configured search domains.
#
# Run "resolvectl status" to see det
# currently in use.
#
# Third party programs should typica
# through the symlink at /etc/resolv
# different way, replace this symlin
#
# See man:systemd-resolved.service(8
# operation for /etc/resolv.conf.
nameserver 185.51.200.2
nameserver 127.0.0.53
```

Fig. 11, Adding shecan's IP to resolv.conf.

Question 6

/etc/resolv.conf :

It translates domain names to IP addresses by querying the Domain Name Server (DNS). The /etc/resolv. conf file is the file that configures the domain name resolver. On a high level, a process in Linux calls the getaddrinfo function to translate a domain name into its corresponding IP address.

/etc/hosts :

file contains the Internet Protocol (IP) host names and addresses for the local host and other hosts in the Internet network. This file is used to resolve a name into an address (that is, to translate a hostname into its Internet address).

/etc/nsswitch.conf:

file is used to configure which services are to be used to determine information such as hostnames, password files, and group files.

Question 7

```
arefe@AREFE:~$ sudo ip netns exec Host ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.038 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.059 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.053 ms
64 bytes from localhost (127.0.0.1): icmp_seq=6 ttl=64 time=0.054 ms
64 bytes from localhost (127.0.0.1): icmp_seq=7 ttl=64 time=0.055 ms
64 bytes from localhost (127.0.0.1): icmp_seq=8 ttl=64 time=0.054 ms
^C
--- localhost ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7161ms
rtt min/avg/max/mdev = 0.038/0.052/0.059/0.005 ms
```

Fig. 12 , localhost address is 127.0.0.1

Localhosts IP is 127.0.0.1, that is because localhosts IP in /etc/hosts is 127.0.0.1, and based on what we found from former question, we know that first we search for a hostname IP in the etc/hosts.

Question 8

For changing localhost IP just for netns Host, first I changed hosts file in /etc/netns/hosts directory

```
arefe@AREFE:~$ sudo ip netns exec Host ping localhost
PING localhost (127.0.0.3) 56(84) bytes of data.
64 bytes from localhost (127.0.0.3): icmp_seq=1 ttl=64 time=0.037 ms
64 bytes from localhost (127.0.0.3): icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from localhost (127.0.0.3): icmp_seq=3 ttl=64 time=0.066 ms
64 bytes from localhost (127.0.0.3): icmp_seq=4 ttl=64 time=0.067 ms
64 bytes from localhost (127.0.0.3): icmp_seq=5 ttl=64 time=0.074 ms
64 bytes from localhost (127.0.0.3): icmp_seq=6 ttl=64 time=0.065 ms
64 bytes from localhost (127.0.0.3): icmp_seq=7 ttl=64 time=0.066 ms
^C
--- localhost ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6128ms
rtt min/avg/max/mdev = 0.037/0.063/0.074/0.011 ms
arefe@AREFE:~$
```

Fig. 13, ping to localhost with a new IP address

Question 9

Host cannot ping 8.8.8.8, because it is not connected to the internet.

```
arefe@AREFE:~$ sudo ip netns exec Host ping 8.8.8.8
ping: connect: Network is unreachable
```

Fig. 14 ping namespace tp 8.8.8.8

```
arefe@AREFE:~$ sudo ip netns exec Host ping google.com
ping: google.com: Temporary failure in name resolution
arefe@AREFE:~$
```

Fig. 15 it cannot find ping google.com .

When Host wants to ping google.com because there is no assigned IP for google.com in /etc/hosts, it cannot ping it, to solve this problem add the IP and hostname of google.com to file /etc/hosts

```
arefe@AREFE:~$ cat /etc/hosts
8.8.8.8         google.com
127.0.0.1       localhost
127.0.1.1       AREFE
```

Fig. 16, mapping google.com with its IP

```
arefe@AREFE:~$ sudo ip netns exec Host ping google.com
ping: connect: Network is unreachable
arefe@AREFE:~$
```

Fig. 17 , after adding google.com 8.8.8.8 to etc/hosts

Question 11

The veth devices are virtual Ethernet devices. They can act as tunnels between network namespaces to create a bridge to a physical network device in another namespace, but can also be used as standalone network devices.

veth devices are always created in interconnected pairs. A pair can be created using the command:

ip link add <p1-name> type veth peer name <p2-name>

Packets transmitted on one device in the pair are immediately received on the other device. When either devices is down the link state of the pair is down.

Question 12

```
arefe@AREFE:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: wlp0s20f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 84:c5:a6:a8:22:3a brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.105/24 brd 192.168.1.255 scope global dynamic noprefixroute wlp0s20f3
       valid_lft 85065sec preferred_lft 85065sec
    inet6 fe80::b96f:f533:5069:381c/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: veth-default@if3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 4e:85:dd:c7:a6:e3 brd ff:ff:ff:ff:ff:ff link-netns Host
arefe@AREFE:~$
```

Fig. 17 ip a

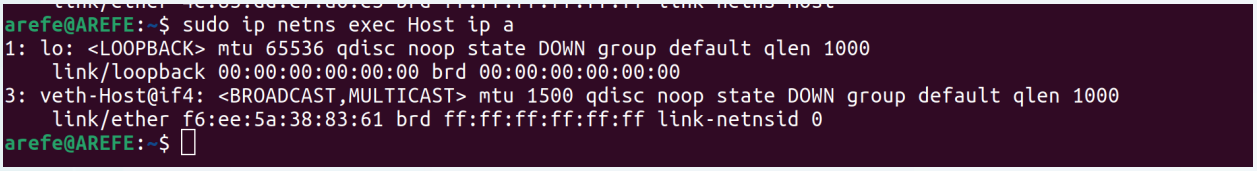

Fig. 18 interfaces of Host netns

By looking at both Figs, we find out that veth-Host is added as the Host interface. and veth-default is added as default natns.

Question 13

```
# iptables -A FORWARD  -o wlp2s0 -i veth - default -j ACCEPT
```

iptables:

Iptables is a firewall utility that is used to manage network traffic on Linux operating systems. It allows administrators to define rules that control how incoming and outgoing traffic is processed, including filtering, blocking, and forwarding packets based on various criteria such as IP addresses, ports, and protocols.

-A:

It is a rule-specification, Rule specification in iptables refers to defining the conditions and actions that will be applied to network traffic passing through the firewall.
-A Append one or more rules to the end of the selected chain.

FORWARD:

The FORWARD chain filters incoming packets that are being forwarded to a different end location.

-o:

Name of an interface via which a packet is going to be sent.

-i:

Name of an interface via which a packet was received.

-j:

This specifies the target of the rule; i.e., what to do if the packet matches it.

ACCEPT:

means that the default policy for that chain, if there are no matching rules, is to allow the traffic

Now we kind of made a tunnel between **`wlp2s0 and veth-default,`** in which, veth-default sends packets and wlp0s2, receives it. What we need to do is making such a somewhat virtual tunnel where wlp2s0 sends packets and veth-default receives, we do it by the following command:

```
# iptables -A FORWARD  -o veth - default -i wlp2s0 -j ACCEPT
```

Iptables -L - v shows a list all iptables chains and number of packets which were affected by each chain

```
arefe@AREFE:~$ sudo iptables -L -v
[sudo] password for arefe:
# Warning: iptables-legacy tables present, use iptables-legacy to see them
Chain INPUT (policy DROP 85 packets, 122K bytes)
 pkts bytes target     prot opt in     out     source               destination
 634K  237M ufw-before-logging-input  all  --  any    any     anywhere             anywhere
 634K  237M ufw-before-input  all  --  any    any     anywhere             anywhere
  229  145K ufw-after-input  all  --  any    any     anywhere             anywhere
  165  128K ufw-after-logging-input  all  --  any    any     anywhere             anywhere
  165  128K ufw-reject-input  all  --  any    any     anywhere             anywhere
  165  128K ufw-track-input  all  --  any    any     anywhere             anywhere

Chain FORWARD (policy DROP 8 packets, 412 bytes)
 pkts bytes target     prot opt in     out     source               destination
   20  1420 ufw-before-logging-forward  all  --  any    any     anywhere             anywhere
   20  1420 ufw-before-forward  all  --  any    any     anywhere             anywhere
    8   412 ufw-after-forward  all  --  any    any     anywhere             anywhere
    8   412 ufw-after-logging-forward  all  --  any    any     anywhere             anywhere
    8   412 ufw-reject-forward  all  --  any    any     anywhere             anywhere
    8   412 ufw-track-forward  all  --  any    any     anywhere             anywhere
    0     0 ACCEPT     all  --  veth-defaukt wlp0s20fs  anywhere             anywhere
    0     0 ACCEPT     all  --  wlp0s20fs veth-defaukt  anywhere             anywhere
```

Fig. 18 iptables -L -v shows a list all iptables chains

Question 14

Seeing the following image, make us conclude that router knows the IP which, each interface is connected to. But how does it know? Let's overview each step we have taken to achieve this result.

```
root@AREFE:/home/arefe# ip netns exec R ip route
10.0.1.0/24 dev R-eth1 proto kernel scope link src 10.0.1.1
10.0.2.0/24 dev R-eth2 proto kernel scope link src 10.0.2.1
root@AREFE:/home/arefe#
```

Fig. 19 router knows the IP of its interfaces.

Step 1:

First we create a virtual ethernet link which is called H1-eth0, and connect it to another link, R-eth1.

```
root@AREFE:/home/arefe# ip link add H1-eth0 type  veth peer name R-eth1
```

Then we create another virtual ethernet link called H2-eth0 and connect it to another link, R-eth2.

```
root@AREFE:/home/arefe# ip link add H2-eth0 type  veth peer name R-eth2
```

Step 2:

Now connect each link to its corresponding network namespace. If you look carefully you see that now H1 is connected to R via R-eth1 and H2 is connected via R-eth2.

```
root@AREFE:/home/arefe# ip link set H1-eth0 netns H1
root@AREFE:/home/arefe# ip link set H2-eth0 netns H2
root@AREFE:/home/arefe# ip link set R-eth1 netns R
root@AREFE:/home/arefe# ip link set R-eth2 netns R
```

Step 3:

Assign an IP to each link.

```
root@AREFE:/home/arefe# ip netns exec H1 ifconfig lo up
root@AREFE:/home/arefe# ip netns exec H1 ifconfig H1-eth0 10.0.1.2 netmask 255.255.255.0
root@AREFE:/home/arefe# ip netns exec H1 ifconfig H1-eth0 up
root@AREFE:/home/arefe# ip netns exec H2 ifconfig lo up
root@AREFE:/home/arefe# ip netns exec H2 ifconfig H2-eth0 10.0.2.2 netmask 255.255.255.0
root@AREFE:/home/arefe# ip netns exec H2 ifconfig H2-eth0 up
```

```
root@AREFE:/home/arefe# ip netns exec R ifconfig R-eth1 10.0.1.1 netmask 255.255.255.0
root@AREFE:/home/arefe# ip netns exec R ifconfig R-eht1 up
R-eht1: ERROR while getting interface flags: No such device
root@AREFE:/home/arefe# ip netns exec R ifconfig R-eth1 up
root@AREFE:/home/arefe# ip netns exec R ifconfig R-eth2 10.0.2.1 netmask 255.255.255.0
root@AREFE:/home/arefe# ip netns exec R ifconfig R-eth2 up
```

To put in a nutshell what has happened until now, see the following lines:

H1-eth0 IP address = 10.0.1.2 —> peer with —> R-eth1 IP address = 10.0.1.1

H1-eth0 IP address = 10.0.2.2 —> peer with —> R-eth2 IP address = 10.0.2.1

Step 4:

check whether network namespaces are connected to the router or not:

```
root@AREFE:/home/arefe# ip netns exec H1 ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.104 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.072 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.070 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=64 time=0.071 ms
64 bytes from 10.0.1.1: icmp_seq=5 ttl=64 time=0.069 ms
64 bytes from 10.0.1.1: icmp_seq=6 ttl=64 time=0.095 ms
64 bytes from 10.0.1.1: icmp_seq=7 ttl=64 time=0.057 ms
64 bytes from 10.0.1.1: icmp_seq=8 ttl=64 time=0.061 ms
64 bytes from 10.0.1.1: icmp_seq=9 ttl=64 time=0.067 ms
64 bytes from 10.0.1.1: icmp_seq=10 ttl=64 time=0.064 ms
64 bytes from 10.0.1.1: icmp_seq=11 ttl=64 time=0.065 ms
64 bytes from 10.0.1.1: icmp_seq=12 ttl=64 time=0.064 ms
^C
--- 10.0.1.1 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11265ms
rtt min/avg/max/mdev = 0.057/0.071/0.104/0.013 ms
root@AREFE:/home/arefe# ip netns exec R ping 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=0.050 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=0.059 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=64 time=0.071 ms
64 bytes from 10.0.1.2: icmp_seq=5 ttl=64 time=0.071 ms
64 bytes from 10.0.1.2: icmp_seq=6 ttl=64 time=0.093 ms
64 bytes from 10.0.1.2: icmp_seq=7 ttl=64 time=0.082 ms
64 bytes from 10.0.1.2: icmp_seq=8 ttl=64 time=0.094 ms
^C
--- 10.0.1.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7173ms
rtt min/avg/max/mdev = 0.039/0.069/0.094/0.018 ms
```

Step 5:

Last step is to set R as the default router for H1 and H2.

```
root@AREFE:/home/arefe# ip netns exec H1 ip route add default via 10.0.1.1 dev H1-eth0
root@AREFE:/home/arefe# ip netns exec H2 ip route add default via 10.0.2.1 dev H2-eth0
```

Assume 🙂 is our packet, which H1 wants to send to H2, H1 knows its router is R which is connected to by the H1-eth0 link, So H1 gets rid of 🙂 by sending it to this link, the router takes the packet and sees it must be sent to H2, and the router knows it is connected to H2 by R-eth2, So router put 🙂 to that link. Finally, H2 receives it.

Now after these steps, you, the router, and I know there is no need for complicated exhausting routing algorithms to find the route of a packet traveling between these two network namespaces.

```
root@AREFE:/home/arefe# ip netns exec H1 ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=63 time=0.126 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=63 time=0.090 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=63 time=0.091 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=63 time=0.090 ms
^C
--- 10.0.2.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3056ms
rtt min/avg/max/mdev = 0.090/0.099/0.126/0.015 ms
```

Question 15

see Fig. 20, there are switches interfaces and their mac addresses:

```
root@AREFE:/home/arefe# ovs-ofctl show s
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000c2039bdf154c
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst mod_nw_src mod_nw_dst mod_nw_
tos mod_tp_src mod_tp_dst
 1(s-eth1): addr:6a:24:22:ed:65:c7
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s-eth2): addr:d6:13:04:4b:92:0d
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s): addr:c2:03:9b:df:15:4c
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
root@AREFE:/home/arefe#
```

Fig. 20, ovs-ofclt show s

Capabilities:

FLOW_STATE:

It is used to enable OVS to keep track of the packets that belong to a particular flow and ensure that they are processed correctly, even if they arrive out of order are lost in transit.

TABLE_STATE:

The table state in ovs-ofctl capabilities refers to the ability of Open vSwitch (OVS) to track the state of flow tables. This means that OVS can keep track of the rules that are currently installed in each flow table and ensure that they are processed correctly.

PORT_STATE:

Usage of PORT_STATE in OVS is to enable Open vSwitch (OVS) to track the state of ports on network switches. This means that OVS can monitor the status of each port, including whether it is up or down, and whether it is connected to another device. With port state tracking, OVS can detect and respond to changes in the network topology, optimize network performance, and provide enhanced security by enforcing policies that control how traffic is processed on each port.

Actions:

output:

The output in ovs-ofctl actions refers to the set of actions that can be performed on packets by Open vSwitch (OVS) when they are received on a particular port. These actions can include forwarding the packet to a specific port, dropping the packet, modifying the packet's header fields, and sending the packet to a controller for further processing. The output in ovs-ofctl actions is typically used to configure the behavior of OVS switches in response to different types of traffic, such as prioritizing certain types of traffic over others or redirecting traffic to specific destinations based on its source or destination address.

enqueue:

Enqueue in ovs-ofctl actions refers to the action of placing a packet in a specific queue for transmission. This action is typically used in Quality of Service (QoS) configurations to prioritize certain types of traffic over others by assigning them to different queues with different levels of priority. By using the enqueue action, packets can be directed to the appropriate queue based on their type or other characteristics, ensuring that they receive the appropriate level of service and that network resources are used efficiently

set_vlan_vid:

Set_vlan_vid in ovs-ofctl actions refers to the action of setting the VLAN ID of a packet. This action is typically used in Virtual LAN (VLAN) configurations to assign a specific VLAN ID to a packet, allowing it to be properly identified and routed within a VLAN network. By using the set_vlan_vid action, packets can be directed to the appropriate VLAN based on their VLAN ID, ensuring that they are properly segregated and managed within the network.

set_vlan_pcp:

Set_vlan_pcp in ovs-ofctl actions refers to the action of setting the Priority Code Point (PCP) value of a VLAN tag. The PCP value is used to prioritize traffic within a VLAN network and is typically used in Quality of Service (QoS) configurations. By using the set_vlan_pcp action, packets can be assigned a specific PCP value, allowing them to be prioritized and managed within the network based on their importance or type of traffic.

strip_vlan:

Strip_vlan in ovs-ofctl actions refers to the action of removing the VLAN tag from a packet. This action is used when a packet needs to be forwarded to a destination that does not support VLAN tagging. By stripping the VLAN tag, the packet can be sent to its destination without any issues. This action is commonly used when packets are being sent between different types of networks or devices that do not support VLAN tagging.

Question 16

```
root@AREFE:/home/arefe# ovs-ofctl dump-flows s
 cookie=0x0, duration=736.840s, table=0, n_packets=39, n_bytes=3090, priority=0 actions=NORMAL
root@AREFE:/home/arefe#
```

Fig. 21, ovs-ofctl dump-flows s

Each record in ovs-ofctl dump-flows represents a flow entry in the OpenFlow switch's flow table.

Cookie:

An opaque value tha the controller can use to identifiy the flow entry.

duration:

The amount of time that the flow exits.

table:

the table in wich flow entry is installed.

n_packets:

number of packets that have been transmitted in the flow.

n_bytes:

The amount of data that have been trasmitted in the flow .

priority:

The priority of the flow entry.

actions:

The actions that the switch should take when a packet matches the flow entry. This can include forwarding the packet to a specific port, dropping the packet, or modifying the packet header. Normal means forwarding the packet.

Question 17

```
root@AREFE:/home/arefe# ovs-ofctl dump-flows s
 cookie=0x0, duration=448.342s, table=0, n_packets=0, n_bytes=0, dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=269.393s, table=0, n_packets=0, n_bytes=0, dl_dst=6a:24:22:ed:65:c7 actions=output:1
 cookie=0x0, duration=226.067s, table=0, n_packets=0, n_bytes=0, dl_dst=d6:13:04:4b:92:0d actions=output:2
 cookie=0x0, duration=186.453s, table=0, n_packets=0, n_bytes=0, arp actions=NORMAL
root@AREFE:/home/arefe#
```

```
root@AREFE:/home/arefe# ip netns exec H1 ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.464 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.098 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.098 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.100 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4089ms
rtt min/avg/max/mdev = 0.090/0.170/0.464/0.147 ms
root@AREFE:/home/arefe#
```

```
root@AREFE:/home/arefe# ip netns exec H1 ifconfig
H1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::f8e3:eeff:fec1:d06c  prefixlen 64  scopeid 0x20<link>
        ether fa:e3:ee:c1:d0:6c  txqueuelen 1000  (Ethernet)
        RX packets 66  bytes 7334 (7.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 61  bytes 4982 (4.9 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 8  bytes 584 (584.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 584 (584.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
root@AREFE:/home/arefe# ip netns exec H2 ifconfig
H2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.2  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::e0b4:b6ff:fec7:25b6  prefixlen 64  scopeid 0x20<link>
        ether e2:b4:b6:c7:25:b6  txqueuelen 1000  (Ethernet)
        RX packets 61  bytes 6888 (6.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 35  bytes 2602 (2.6 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 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
```

```
root@AREFE:/home/arefe# ifconfig
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 158114  bytes 11567879 (11.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 158114  bytes 11567879 (11.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::6824:22ff:feed:65c7  prefixlen 64  scopeid 0x20<link>
        ether 6a:24:22:ed:65:c7  txqueuelen 1000  (Ethernet)
        RX packets 61  bytes 4982 (4.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 66  bytes 7334 (7.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

s-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::d413:4ff:fe4b:920d  prefixlen 64  scopeid 0x20<link>
        ether d6:13:04:4b:92:0d  txqueuelen 1000  (Ethernet)
        RX packets 35  bytes 2602 (2.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 61  bytes 6888 (6.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp0s20f3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.105  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::b96f:f533:5069:381c  prefixlen 64  scopeid 0x20<link>
        ether 84:c5:a6:a8:22:3a  txqueuelen 1000  (Ethernet)
        RX packets 567966  bytes 248961808 (248.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 425432  bytes 273506196 (273.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

second part

task 1: Creating Switch, client, server, and their connection.

First of all let's create the client, server and their links.

```
root@AREFE:/home/arefe# ip netns add Client
root@AREFE:/home/arefe# ip netns add Server
root@AREFE:/home/arefe# ip link add H1-eth0 type veth peer name s-eth1
root@AREFE:/home/arefe# ip link add H2-eth0 type veth peer name s-eth2
root@AREFE:/home/arefe# ip link set H1-eth0 netns Client
root@AREFE:/home/arefe# ip link set H2-eth0 netns Server
```

Fig. 1, Creating netns and their links.

Now connect them to each other.

```
root@AREFE:/home/arefe# ovs-vsctl add-br s
```

```
root@AREFE:/home/arefe# ovs-vsctl add-port s s-eth1
root@AREFE:/home/arefe# ovs-vsctl add-port s s-eth2
```

Fig. 2, create the switch and connect its ports.

Now I create a link and set one of its points to the internet and the other to the switch.

```
root@AREFE:/home/arefe# ip link add veth-default type veth peer name veth-switch
root@AREFE:/home/arefe# ovs-vsctl add-port s veth-switch
root@AREFE:/home/arefe# ifconfig veth-switch up
root@AREFE:/home/arefe# ifconfig veth-default up
root@AREFE:/home/arefe# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@AREFE:/home/arefe# iptables -A FORWARD -o wlp0s20f3 -i veth-default -j ACCEPT
root@AREFE:/home/arefe# iptables -A FORWARD -i wlp0s20f3 -o veth-default -j ACCEPT
```

Fig. 3, connect the switch to the internet.

task 2: Set ip for client and server

Set IP for network namespaces

```
root@AREFE:/home/arefe# ip netns exec Client ifconfig lo up
root@AREFE:/home/arefe# ip netns exec Client ifconfig H1-eth0 100.1.0.1
root@AREFE:/home/arefe# ip netns exec Server ifconfig lo up
root@AREFE:/home/arefe# ip netns exec Server ifconfig H2-eth0 100.1.0.2
```

Fig. 4,Set IP for network namespaces

task 3: Set default router for client and server

```
root@AREFE:/home/arefe# ifconfig s-eth1 100.1.0.3/24
root@AREFE:/home/arefe# ifconfig s-eth2 100.1.0.4/24
```

```
root@AREFE:/home/arefe# ip netns exec Client ip route add default via 100.1.0.3
root@AREFE:/home/arefe# ip netns exec Server ip route add default via 100.1.0.4
root@AREFE:/home/arefe# ip netns exec Client route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         100.1.0.3       0.0.0.0         UG    0      0        0 H1-eth0
100.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 H1-eth0
root@AREFE:/home/arefe# ip netns exec Server route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         100.1.0.4       0.0.0.0         UG    0      0        0 H2-eth0
100.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 H2-eth0
```

Fig. 5,Set router for each netns.

task 4: connect switch to the internet

```
root@AREFE:/home/arefe# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@AREFE:/home/arefe# iptables -A FORWARD -o wlp0s20f3 -i veth-default -j ACCEPT
root@AREFE:/home/arefe# iptables -A FORWARD -i wlp0s20f3 -o veth-default -j ACCEPT
root@AREFE:/home/arefe# ifconfig veth-switch 100.1.0.5/24
root@AREFE:/home/arefe# iptables -t nat -A POSTROUTING -s 100.1.0.5 -o wlp0s20f3 -j MASQUERADE
```

Fig. 6, forward received packets to the switch.

task 5: assign port for ip adresses

for detemining destination of each pachet based on their IP address, I set priority for packets destination. When a packet arives, first of all switch checks weather its IP is Client's IP or not, if it is, sends the apcket in the port 1 else if it is not then switch checkers if it is Serrver or not. Finally if it is neither for the Client nor the Server, the Switch sends it to the third port, which is the internet.

```
root@AREFE:/home/arefe# ovs-ofctl add-flow Switch "priority=101,ip_dst=100.1.0.1,actions=output:1"
root@AREFE:/home/arefe# ovs-ofctl add-flow Switch "priority=100,ip_dst=100.1.0.2,actions=output:2"
root@AREFE:/home/arefe# ovs-ofctl add-flow Switch "priority=50,actions=output:3"
```

Fig. 6, determine destination of each packet.

task 6: change /etc/hosts/

```
root@AREFE:/home/arefe# cat /etc/hosts
100.1.0.2       server.com
8.8.8.8         google.com
127.0.0.1       localhost
127.0.1.1       arefe-WRTB-WXX9
```

```
root@AREFE:/home/arefe# ip netns exec Client ping server.com
PING server.com (100.1.0.2) 56(84) bytes of data.
64 bytes from server.com (100.1.0.2): icmp_seq=1 ttl=64 time=0.518 ms
64 bytes from server.com (100.1.0.2): icmp_seq=2 ttl=64 time=0.096 ms
64 bytes from server.com (100.1.0.2): icmp_seq=3 ttl=64 time=0.095 ms
64 bytes from server.com (100.1.0.2): icmp_seq=4 ttl=64 time=0.087 ms
64 bytes from server.com (100.1.0.2): icmp_seq=5 ttl=64 time=0.086 ms
^C
--- server.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4085ms
rtt min/avg/max/mdev = 0.086/0.176/0.518/0.170 ms
```

Fig. 7, ping Server by its name.

task 7: make index.html file

As written below of index.html, the file is in the home directory.

Open — index.html — ~/

```
<! DOCTYPE html >
<html lang="en">
  <head >
    <meta charset ="UTF -8" />
    <title >Document </title >
  </head >
  <body >
    <h1>Arefe Pourmohammadi </h1>
    <h2>40005783</h2>
  </body >
</html >
```

task 8: let's get file

first run server to listen for a request. And then send a request from Client to the Server.

```
root@AREFE:/home/arefe# sudo ip netns exec Server python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
100.1.0.1 - - [02/Jun/2023 14:09:09] "GET / HTTP/1.1" 200 -
```

```
arefe@AREFE:~$ sudo ip netns exec Client wget server.com
[sudo] password for arefe:
--2023-06-02 14:09:09--  http://server.com/
Resolving server.com (server.com)... 100.1.0.2
Connecting to server.com (server.com)|100.1.0.2|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 204 [text/html]
Saving to: ‘index.html.5’

index.html.5              100%[=========================================>]     204  --.-KB/s    in 0s

2023-06-02 14:09:09 (43.2 MB/s) - ‘index.html.5’ saved [204/204]
```

Fig. 8, transmitting index.html, between Server and Client via switch.

task 9: Client pings github

Now we can connect Client to the internet via switch. I connected to the github as an example.

```
root@AREFE:/home/arefe# cat /etc/hosts
192.30.255.113  github.com
100.1.0.2       server.com
8.8.8.8         google.com
127.0.0.1       localhost
127.0.1.1       arefe-WRTB-WXX9

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Fig. 9, write github.com and its IP in /etc/hosts

```
root@AREFE:/home/arefe# ip netns exec Client wget github.com
--2023-06-04 22:18:24--  http://github.com/
Resolving github.com (github.com)... 192.30.255.113
Connecting to github.com (github.com)|192.30.255.113|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://github.com/ [following]
--2023-06-04 22:18:24--  https://github.com/
Connecting to github.com (github.com)|192.30.255.113|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: ‘index.html.8’

index.html.8                [         <=>                                ] 222.25K  57.1KB/s    in 3.9s

2023-06-04 22:18:30 (57.1 KB/s) - ‘index.html.8’ saved [227582]
```

Fig. 10, Client can wget github.com

task 10: final result in wireshark

In Fig. 12, you see that the source mac address is fa:e3:ee:c1:d0:6c, and destination mac address is e2:b4:b6:c7:25:b6, Now see Fig.13 ,those mac addresses are actually Client and Server mac addresses, again assume 😣 is our poor packet, which is sent by Client to the switch, we know that Client write mac address of next hop after switch in the packet to help switch to figure out in which port it must send it, the next hop is Server so Client writes Server's mac address as destination mac address. That is a fair reason to prove that packets, which are supposed to go to the Server, go directly to the Switch and then directly to the Server.

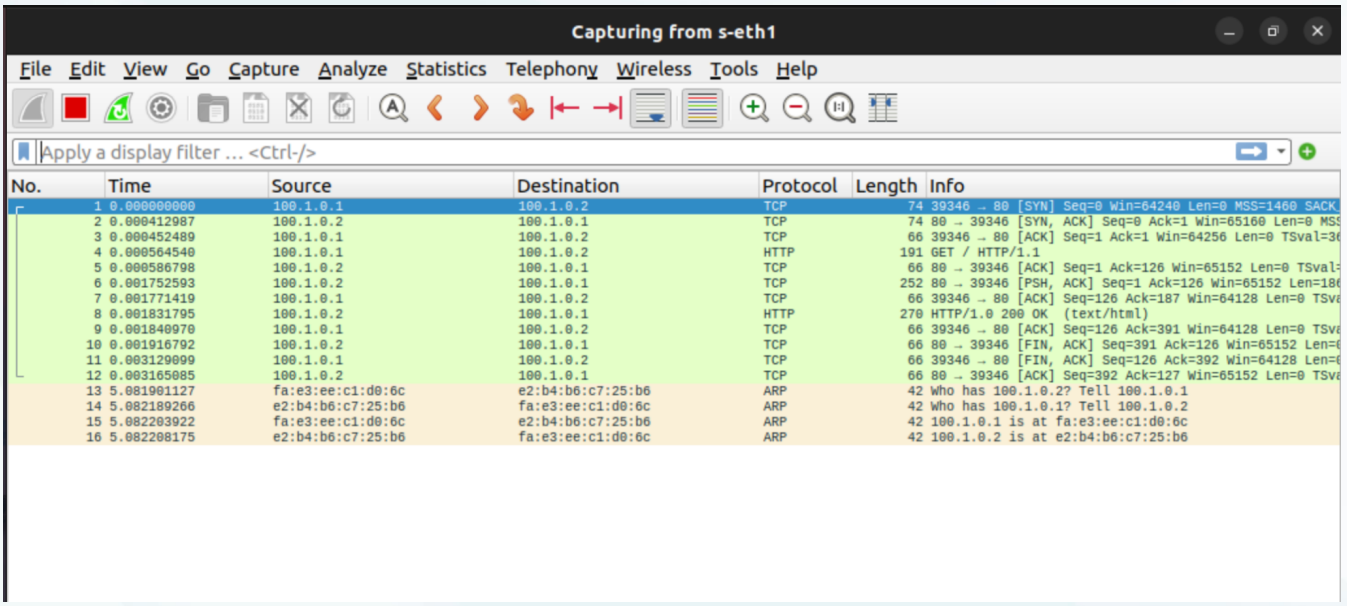

Fig. 11, packets which are sent from client to the server

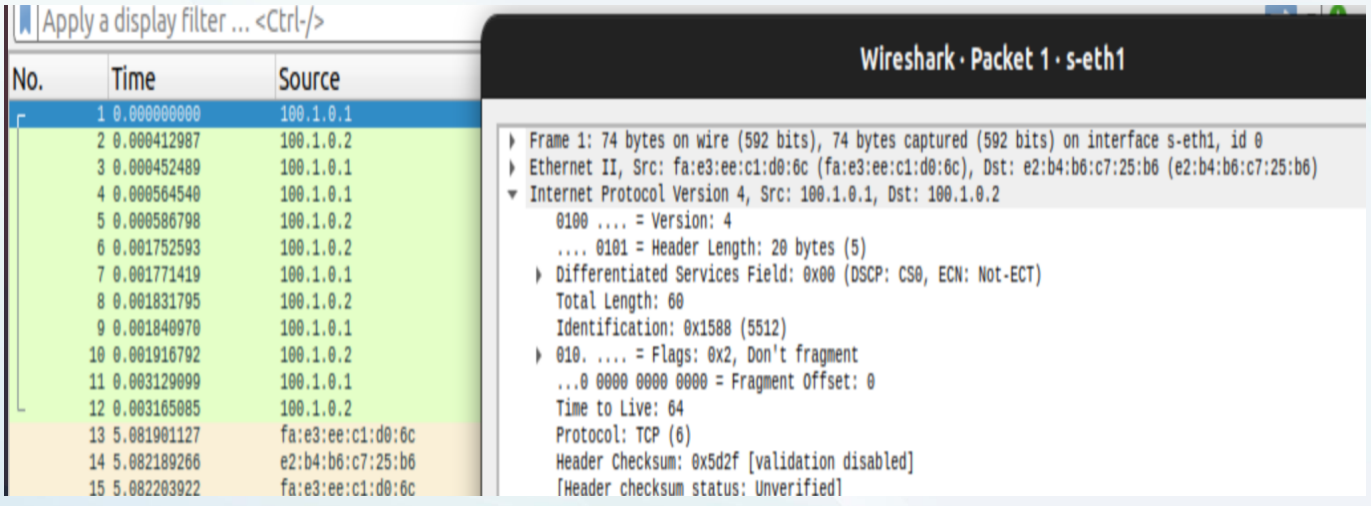

Fig. 12, notice source and destination mac addresses of a packet

```
arefe@AREFE:~$ sudo ip netns exec Client ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
8: H1-eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether fa:e3:ee:c1:d0:6c brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 100.1.0.1/24 brd 100.1.0.255 scope global H1-eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::f8e3:eeff:fec1:d06c/64 scope link
       valid_lft forever preferred_lft forever
arefe@AREFE:~$ sudo ip netns exec Server ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
10: H2-eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether e2:b4:b6:c7:25:b6 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 100.1.0.2/24 brd 100.1.0.255 scope global H2-eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::e0b4:b6ff:fec7:25b6/64 scope link
       valid_lft forever preferred_lft forever
```

Fig. 13, notice mac address of Client and Server.