LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)

L.B. REDDY NAGAR, MYLAVARAM, KRISHNA DIST., A.P.-521 230.

DEPARTMENT
OF
INFORMATION TECHNOLOGY

COMPUTER NETWORKS (20CS60)
B.Tech. V SEM

Mr.K.Rajasekhar

Experiment-1

Aim: To gain familiarity with the basic 'networks' commands & utilities available in the Linux OS.

Some general tips:

- If any command is not installed on your system, do **sudo apt-get update** on the Terminal followed by **sudo apt-get install <name>** to install it.
- To see more details about any command, type **man <name>** on the Terminal.
- Running **<name> -h** on your Terminal will display the help menu of that command.

Procedure:

Run the following commands on your Terminal window.

1) Troubleshooting network hosts
   a) ping <address>

   Short for **Packet InterNet Groper,** the ping command is used to test the ability of your computer to reach a specified destination computer. The ping command is usually used as a simple way to verify that a computer can communicate over the network with another computer or network device. The ping utility is commonly used to check for network errors, and works by sending **ICMP ECHO_REQUEST** to network hosts.

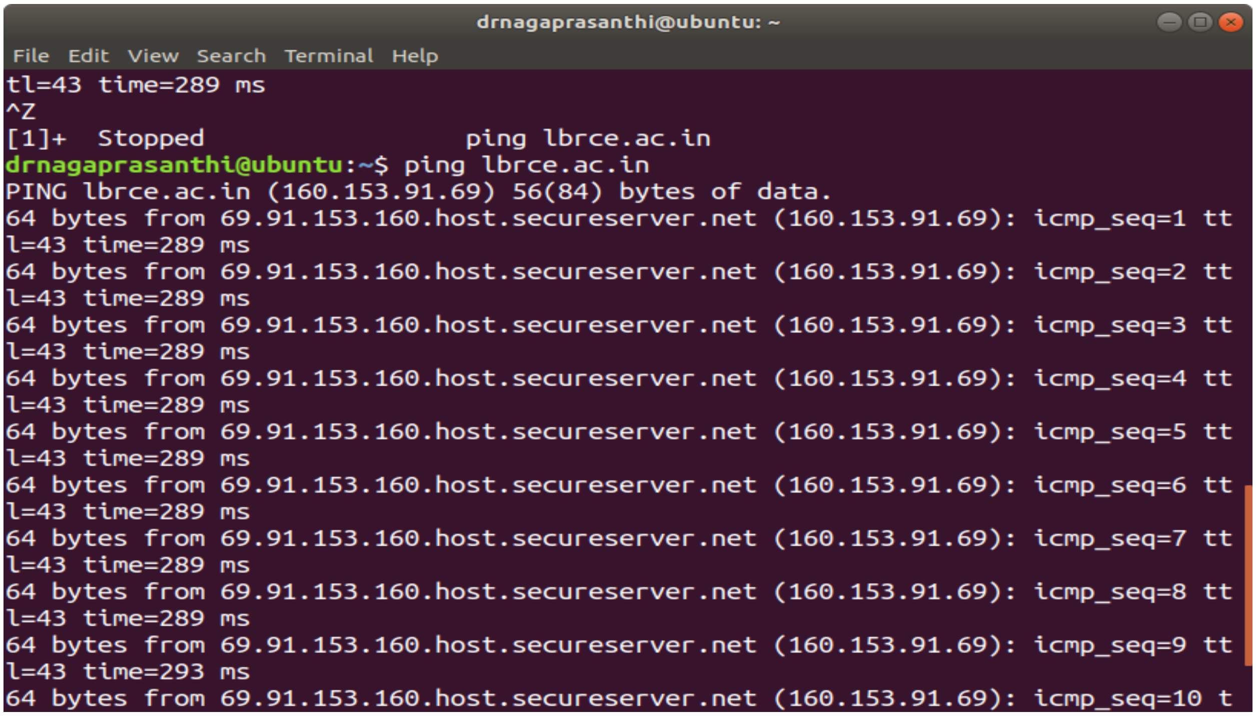


Figure.1. Output of ping command

Ping two different machines, one within India and the other one outside India, and observe the latency.

Understanding about network interfaces

b) ifconfig [options]

ifconfig is used to configure the network interfaces. It is used at boot time to set up interfaces as necessary. After that, it is usually only needed when debugging or when system tuning is needed.

If no arguments are given, **ifconfig** displays the status of the currently active interfaces.

```
ipc1@ipc1-ThinkCentre-M71e:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 8c:89:a5:21:47:18
          inet addr:172.16.4.100  Bcast:172.16.4.127  Mask:255.255.255.128
          inet6 addr: fe80::8e89:a5ff:fe21:4718/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:239800 errors:0 dropped:0 overruns:0 frame:0
          TX packets:116557 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:289189259 (289.1 MB)  TX bytes:15052528 (15.0 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:8207 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8207 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:858625 (858.6 KB)  TX bytes:858625 (858.6 KB)

ipc1@ipc1-ThinkCentre-M71e:~$
```

Figure.2. The output of “ifconfig” command

If a single interface argument is given, it displays the status of the given interface only;

```
ipc1@ipc1-ThinkCentre-M71e:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 8c:89:a5:21:47:18
          inet addr:172.16.4.100  Bcast:172.16.4.127  Mask:255.255.255.128
          inet6 addr: fe80::8e89:a5ff:fe21:4718/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:239848 errors:0 dropped:0 overruns:0 frame:0
          TX packets:116606 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:289194989 (289.1 MB)  TX bytes:15065874 (15.0 MB)
```

Figure.3. The output displayed on running “ifconfig eth0” if a single **-a** argument is given, it displays the status of all interfaces, even those that are down.

```
ipc1@ipc1-ThinkCentre-M71e:~$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr 8c:89:a5:21:47:18
          inet addr:172.16.4.100  Bcast:172.16.4.127  Mask:255.255.255.128
          inet6 addr: fe80::8e89:a5ff:fe21:4718/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:239855 errors:0 dropped:0 overruns:0 frame:0
          TX packets:116612 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:289196059 (289.1 MB)  TX bytes:15068124 (15.0 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:8215 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8215 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:859431 (859.4 KB)  TX bytes:859431 (859.4 KB)
```

Figure.4. The output displayed on running “ifconfig -a”

What is the IPv4 address of your computer?

What is the MAC address/HW address of your NIC card?

c) sudo ifconfig eth0 down

The above command will bring the ethernet interface down, meaning, the system would be disconnected from the network. Now, try to ping any network host. What is the observed output?

d) sudo ifconfig eth0 up

This command will call DHCP service which is involved in obtaining an IP address. Now, ping to any external system. What is the observed output?

e) ifplugstatus

The ifplugstatus command will tell you whether a cable is plugged into a network interface or not. It isn’t installed by default on Ubuntu. Use the following command to install it:

sudo apt-get install ifplugd

```
ipc1@ipc1-ThinkCentre-M71e:~$ sudo apt-get install ifplugd
[sudo] password for ipc1:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  waproamd
The following NEW packages will be installed:
  ifplugd
0 upgraded, 1 newly installed, 0 to remove and 457 not upgraded.
Need to get 64.0 kB of archives.
After this operation, 270 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu/ trusty/universe ifplugd i386 0.28-19ubuntu1 [64.0 kB]
Fetched 64.0 kB in 1s (45.9 kB/s)
Preconfiguring packages ...
Selecting previously unselected package ifplugd.
(Reading database ... 163571 files and directories currently installed.)
Preparing to unpack .../ifplugd_0.28-19ubuntu1_i386.deb ...
Unpacking ifplugd (0.28-19ubuntu1) ...
Processing triggers for man-db (2.6.7.1-1) ...
Processing triggers for ureadahead (0.100.0-16) ...
ureadahead will be reprofiled on next reboot
Setting up ifplugd (0.28-19ubuntu1) ...
Processing triggers for ureadahead (0.100.0-16) ...
```

Figure.5. Installing "ifplugd" in Ubuntu system The

output of ifplugstatus command when the cable is plugged.

```
ipc1@ipc1-ThinkCentre-M71e:~$ ifplugstatus
lo: link beat detected
eth0: link beat detected
```

Figure.6. The output of "ifplugstatus" command

2) Finding all the intermediate network systems

   a) traceroute <address>

It isn't installed by default on Ubuntu. Use the following command to

install it: sudo apt-get install traceroute

traceroute is a command used to display the intermediate nodes through which a packet flows from a source location to a destination location. A program capable of doing the same in Microsoft Windows is **tracert**.

```
ipc1@ipc1-ThinkCentre-M71e:~$ traceroute bits-pilani.ac.in
traceroute to bits-pilani.ac.in (202.78.175.200), 30 hops max, 60 byte packets
 1  172.16.4.2 (172.16.4.2)  0.488 ms  0.502 ms  0.601 ms
 2  172.16.0.30 (172.16.0.30)  0.171 ms  0.175 ms  0.169 ms
 3  static-69.6.93.111.tataidc.co.in (111.93.6.69)  49.636 ms  50.148 ms  50.147 ms
 4  14.141.87.141.static-hyderabad.tcl.net.in (14.141.87.141)  49.587 ms  49.593 ms 115.113.207.153.stati
c-hyderabad.vsnl.net.in (115.113.207.153)  49.581 ms
 5  172.29.250.34 (172.29.250.34)  55.857 ms  57.240 ms  55.687 ms
 6  172.25.75.246 (172.25.75.246)  59.170 ms  58.021 ms  57.996 ms
 7  115.113.254.18.static-delhi.vsnl.net.in (115.113.254.18)  59.012 ms  35.659 ms  34.937 ms
 8  202.78.175.46 (202.78.175.46)  35.295 ms  32.466 ms  32.301 ms
 9  202.78.168.30 (202.78.168.30)  39.161 ms  39.136 ms  36.193 ms
10  202.78.173.58 (202.78.173.58)  52.669 ms  50.606 ms  52.411 ms
11  ns1.bits-pilani.ac.in (202.78.175.200)  53.657 ms  53.632 ms  53.627 ms
```

Figure.7. The output of “traceroute” command

Observe the latency for every hop, IP address of the First hop router, and First hop of your ISP, and Total number ISPs which your search explored.

1) DNS tools

Find the IP addresses of the following machines: bits-hyderabad.ac.in
swd.bits-hyderabad.ac.in
sites.bits-hyderabad.ac.in

a) nslookup <address>

nslookup is a program to query Internet domain name servers. nslookup has two modes: interactive and noninteractive. Interactive mode allows the user to query name servers for information about various hosts and domains or to print a list of hosts in a domain. Non-interactive mode is used to print just the name and requested information for a host or domain.

```
drnagaprasanthi@ubuntu:~$ nslookup lbrce.ac.in
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   lbrce.ac.in
Address: 160.153.91.69
```

Figure.8. The output of “nslookup” command

b) host <address>

host is a simple utility for performing DNS lookups. It is normally used to convert names to IP addresses and vice versa. When no arguments or options are given, host prints a short summary of its command line arguments and options.

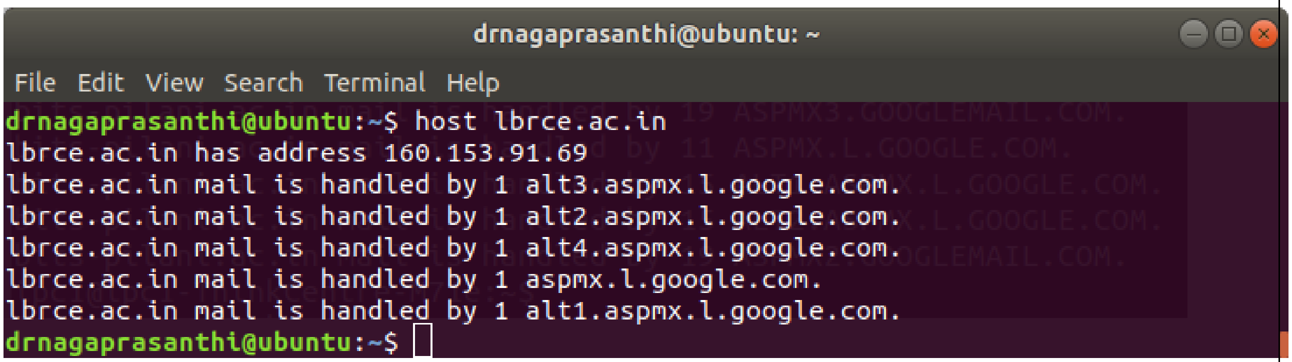


Figure.9. The output of “host” command

c) dig <address>

domain information groper or **dig** is a flexible tool for interrogating DNS name servers. It performs DNS lookups and displays the answers that are returned from the name server(s) that were queried.

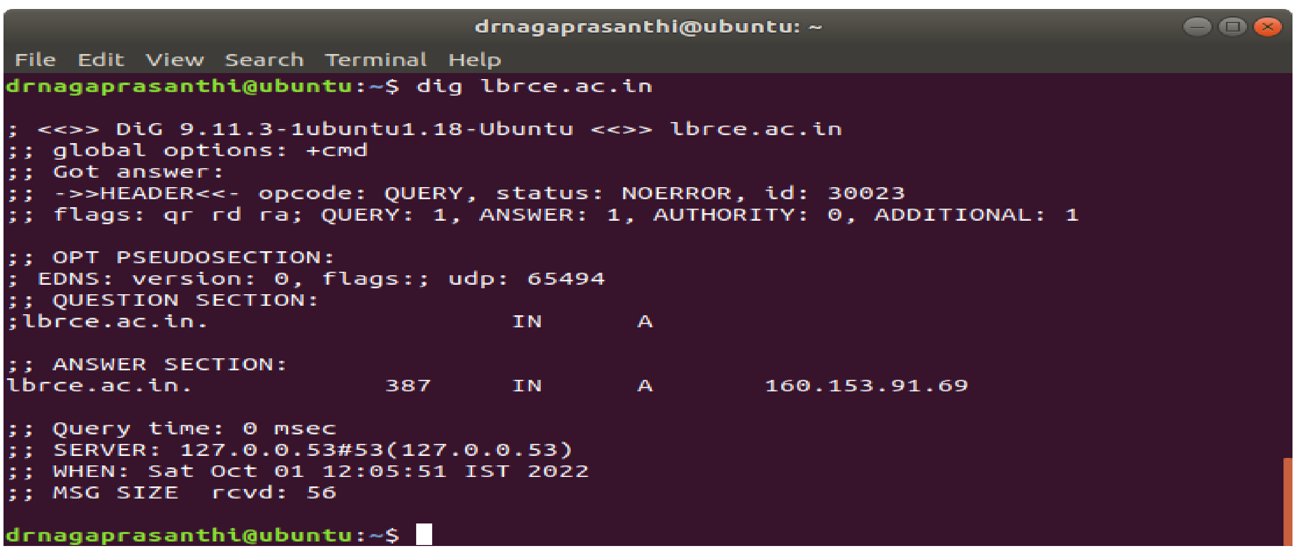


Figure.10. The output of “dig” command

d) DNS Configuration file

All DNS tools makes use of system DNS configuration file located at /etc directory (/etc/resolv.conf). The contents of the file should appear like the below screenshot.

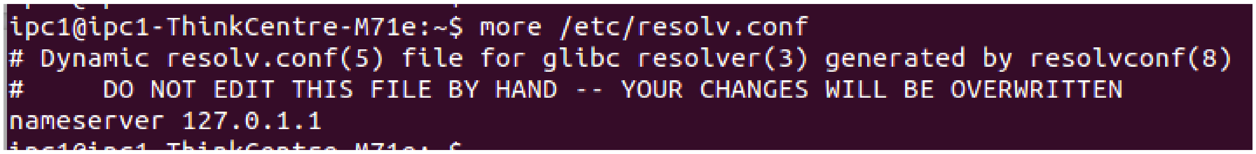


Figure.11. The configuration file "/etc/resolv.conf"

2) Copying files from/to a remote host

a) scp

scp allows files to be copied to, from, or between different hosts. For example, "scp remote_username@remote_host:/home/remote_username/file.txt /home/your_username" will copy the file "file.txt" (located at /home/remote_username) of the remote host "remote_host" to your local directory (/home/your_username).

To download the tar file (CNLab1.tar) from host 172.16.4.100 to your system, the command is sudo scp ipc1@172.16.4.100:/home/ipc1/CNLab1.tar /home/ipc1

Figure.12. Ubuntu Terminal Output

To upload a tar file (2011A7PS111H.tar) from your local directory to a remote directory, the command is

sudo scp /home/ipc1/2011A7PS111H.tar ipc1@172.16.4.100:/home/ipc1

Figure.13. Ubuntu Terminal Output

Now, try to download the folder (/home/ipc1/Lab1) from host 172.16.4.100 to your local PC. Also, try to upload your own folder to a remote host. (The folder and all its contents including sub-folders should be copied).

3) To test the speed of internet connection

Run the following commands for installing speedtest-cli package.

sudo apt-get install python-pip
sudo pip install speedtest-cli

After installation, type “speedtest” in the Terminal and press enter. The output should be similar to the figure below.

```
ipc1@ipc1-ThinkCentre-M71e:~$ speedtest
Retrieving speedtest.net configuration...
Retrieving speedtest.net server list...
Testing from Tata Teleservices ISP (111.93.5.194)...
Selecting best server based on latency...
Hosted by Pioneer Elabs Limited. (Hyderabad) [1.05 km]: 45.016 ms
Testing download speed........................................
Download: 24.94 Mbits/s
Testing upload speed..................................................
Upload: 81.91 Mbits/s
```

Figure.14. Ubuntu Terminal Output

References:

1. Linux ‘man pages’ available at http://linux.die.net/man/
2. http://www.computerhope.com/

http://www.webupd8.org/2014/02/how-to-test-internet-speed-via-command.html

Experiment-2

AIM:- To learn about network layer tools and analyze captures for congestion.

To view routing table entries:

A routing table, or routing information base (RIB), is a data table stored in a router or a networked computer that lists the routes to particular network destinations, and in some cases, metrics (distances) associated with those routes. The routing table contains information about the topology of the network immediately around it. The construction of routing tables is the primary goal of routing protocols. Static routes are entries made in a

routing table by non-automatic means, which are fixed rather than being the result of some network topology "discovery" procedure.

"route" command is used to print, add, delete, edit routes in kernel's IP routing table. Its primary use is to set up static routes to specific hosts or networks via an interface after it has been configured with the ifconfig program. When the **add** or **del** options are used, **route** modifies the routing tables. Without these options, **route** displays the current contents of the routing tables.

Tools used: **route , ip route**

Experiment 1:

1. Print routing table of your system. Use manual pages to capture observations that follow.
    a. The command is "route". Type "route" in Ubuntu terminal.

```
gokul@gokul-VirtualBox:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        *               255.255.255.0   U     1      0        0 eth0
link-local      *               255.255.0.0     U     1000   0        0 eth0
```

Figure.1.1 "route" output

a) What is the IP address of the default gateway (in your IPC system)?
b) What does the flags 'U' and 'G' represent?
c) What is the metric value for default gateway?

b. Option "-n" displays all symbolic references in numeric values.

```
gokul@gokul-VirtualBox:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     1      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 eth0
```

Figure.1.2 “route” numeric output

a) Which route will be taken by a packet with destination address as 172.16.5.128?

c. For faster processing, the routing table is stored in kernel cache. To retrieve table from cache, use option “-C”. By default, “route” command shows the table stored in FIB.

```
gokul@gokul-VirtualBox:~$ route -C
Kernel IP routing cache
Source          Destination     Gateway         Flags Metric Ref    Use Iface
```

Figure.1.3 “route” output from kernel cache

```
gokul@gokul-VirtualBox:~$ route -F
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        *               255.255.255.0   U     1      0        0 eth0
link-local      *               255.255.0.0     U     1000   0        0 eth0
```

Figure.1.4 “route” output from FIB

d. To display all the information in the routing table, use option “ee”.

```
gokul@gokul-VirtualBox:~$ route -nee
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface    MSS   Window irtt
0.0.0.0         10.0.2.2        0.0.0.0         UG    0      0        0 eth0     0     0      0
10.0.2.0        0.0.0.0         255.255.255.0   U     1      0        0 eth0     0     0      0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 eth0     0     0      0
```

Figure.1.5 “route” long listing

a) What are MSS and Window?

e. “netstat” command can also be used to display routing table.

```
gokul@gokul-VirtualBox:~$ netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         10.0.2.2        0.0.0.0         UG        0 0          0 eth0
10.0.2.0        *               255.255.255.0   U         0 0          0 eth0
link-local      *               255.255.0.0     U         0 0          0 eth0
```

Figure.1.6 “netstat” for displaying routing table

2. Add a route to the kernel route table

a. In order to add/delete routes, you should have root privileges. In the below figure, a loopback address is added to the route.

```
root@gokul-VirtualBox:~# route add -net 127.0.0.0 netmask 255.0.0.0 dev lo
root@gokul-VirtualBox:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     1      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 eth0
```

Figure.1.7 Adding a loopback address

a) what happens when you add options of "window 6000 mss 1440 irtt 300" to the route command in the above figure (note: to 'add' the route again, you must 'delete' it first as explained in step 3)

b. In the below figure, a route to IP address "192.56.76.0" is added. The next hop is interface eth0.

```
root@gokul-VirtualBox:~# route add -net 192.56.76.0 netmask 255.255.255.0 dev eth0
root@gokul-VirtualBox:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        *               255.255.255.0   U     1      0        0 eth0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
link-local      *               255.255.0.0     U     1000   0        0 eth0
192.56.76.0     *               255.255.255.0   U     0      0        0 eth0
```

Figure.1.8 Adding "192.56.76.0" as IP address

3. Delete a route in the kernel route table

a. Delete route for IP address "192.56.76.0" created in step 2b.

```
root@gokul-VirtualBox:~# route del -net 192.56.76.0/24
root@gokul-VirtualBox:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        *               255.255.255.0   U     1      0        0 eth0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
link-local      *               255.255.0.0     U     1000   0        0 eth0
```

Figure.1.9 Delete "192.56.76.0" address from routing table

b. Delete loopback address route.

```
root@gokul-VirtualBox:~# route del -net 127.0.0.0/8
root@gokul-VirtualBox:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        *               255.255.255.0   U     1      0        0 eth0
link-local      *               255.255.0.0     U     1000   0        0 eth0
```

Figure.1.10 Delete loopback address

4. Add/Remove a default gateway

In the below experiment, we will delete the default gateway and re-create it.

a. Delete the route for default gateway by the following the below figure.

```
root@gokul-VirtualBox:~# route del default
root@gokul-VirtualBox:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.2.0        *               255.255.255.0   U     1      0        0 eth0
link-local      *               255.255.0.0     U     1000   0        0 eth0
```

Figure.1.11 Delete default gateway

b. Use your browser and connect to gmail.com (or any other web-site). There will be a connection error because default gateway is unavailable. The above can also be tested using a "ping" command as shown below.

```
root@gokul-VirtualBox:~# ping google.com
ping: unknown host google.com
root@gokul-VirtualBox:~# ping yahoo.com
ping: unknown host yahoo.com
```

Figure.1.12 ping not working

Add the default gateway back to its original value (172.16.4.1   for IPC1 lab).

```
root@gokul-VirtualBox:~# route add default gw 10.0.2.2 eth0
root@gokul-VirtualBox:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     1      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 eth0
root@gokul-VirtualBox:~# ping gmail.com
PING gmail.com (74.125.236.149) 56(84) bytes of data.
64 bytes from bom03s02-in-f21.1e100.net (74.125.236.149): icmp_req=1 ttl=56 time=66.7 ms
64 bytes from bom03s02-in-f21.1e100.net (74.125.236.149): icmp_req=2 ttl=56 time=56.9 ms
64 bytes from bom03s02-in-f21.1e100.net (74.125.236.149): icmp_req=3 ttl=56 time=56.7 ms
^C
--- gmail.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 56.741/60.122/66.709/4.662 ms
```

Figure.1.13 Adding the default gateway

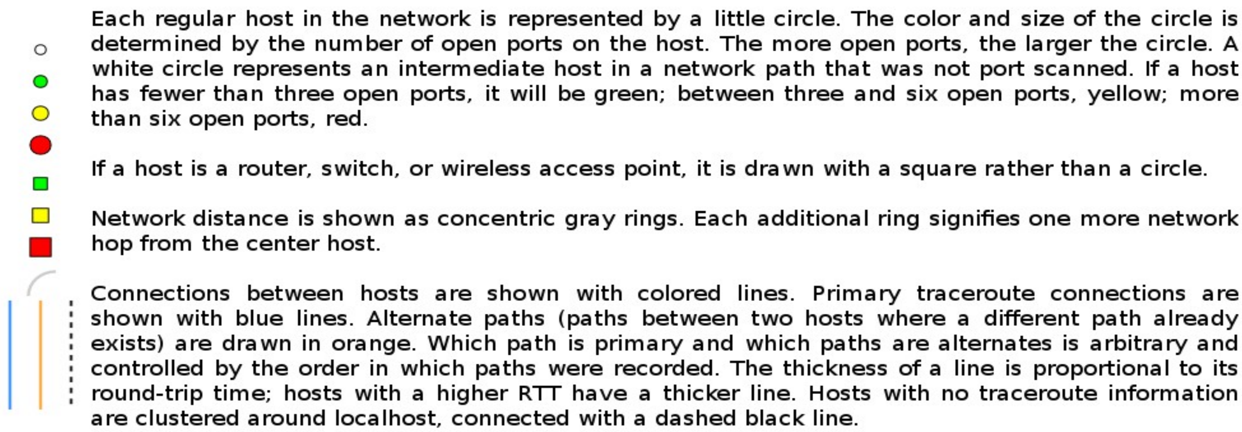
d. Use your browser and connect to gmail.com (or any other web-site). It should work fine.

Analyse an internal network using Zenmap/nmap:

Nmap (*Network Mapper*) is a security scanner used to discover hosts and services on a computer network, thus creating a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host and then analyzes the responses.

The software provides a number of features for probing computer networks, including host discovery and service and operating system detection. These features are extensible by scripts that provide more advanced service detection,vulnerability detection, and other features. Nmap is also capable of adapting to network conditions including latency and congestion during a scan.

Zenmap is the official Nmap Security Scanner GUI. It is a multi-platform (Linux, Windows, Mac OS X, BSD, etc.) free and open source application which aims to make Nmap easy for beginners to use while providing advanced features for experienced Nmap users. Frequently used scans can be saved as profiles to make them easy to run repeatedly. A command creator allows interactive creation of Nmap command lines. Scan results can be saved and viewed later. Saved scan results can be compared with one another to see how they differ. The results of recent scans are stored in a searchable database. The topology view in the Zenmap uses many symbols and color conventions.



Each regular host in the network is represented by a little circle. The color and size of the circle is determined by the number of open ports on the host. The more open ports, the larger the circle. A white circle represents an intermediate host in a network path that was not port scanned. If a host has fewer than three open ports, it will be green; between three and six open ports, yellow; more than six open ports, red.

If a host is a router, switch, or wireless access point, it is drawn with a square rather than a circle.

Network distance is shown as concentric gray rings. Each additional ring signifies one more network hop from the center host.

Connections between hosts are shown with colored lines. Primary traceroute connections are shown with blue lines. Alternate paths (paths between two hosts where a different path already exists) are drawn in orange. Which path is primary and which paths are alternates is arbitrary and controlled by the order in which paths were recorded. The thickness of a line is proportional to its round-trip time; hosts with a higher RTT have a thicker line. Hosts with no traceroute information are clustered around localhost, connected with a dashed black line.

Tool used: Zenmap

Experiment 2:

1. Port Scanning: Performs a port-scan to check for open ports on the specified IP address range
   Open zenmap and give the following details.
   Target: **172.16.5.0/27**
   Command*:* nmap 172.16*.5.0/27*
   Click "Scan" button on the top right corner.

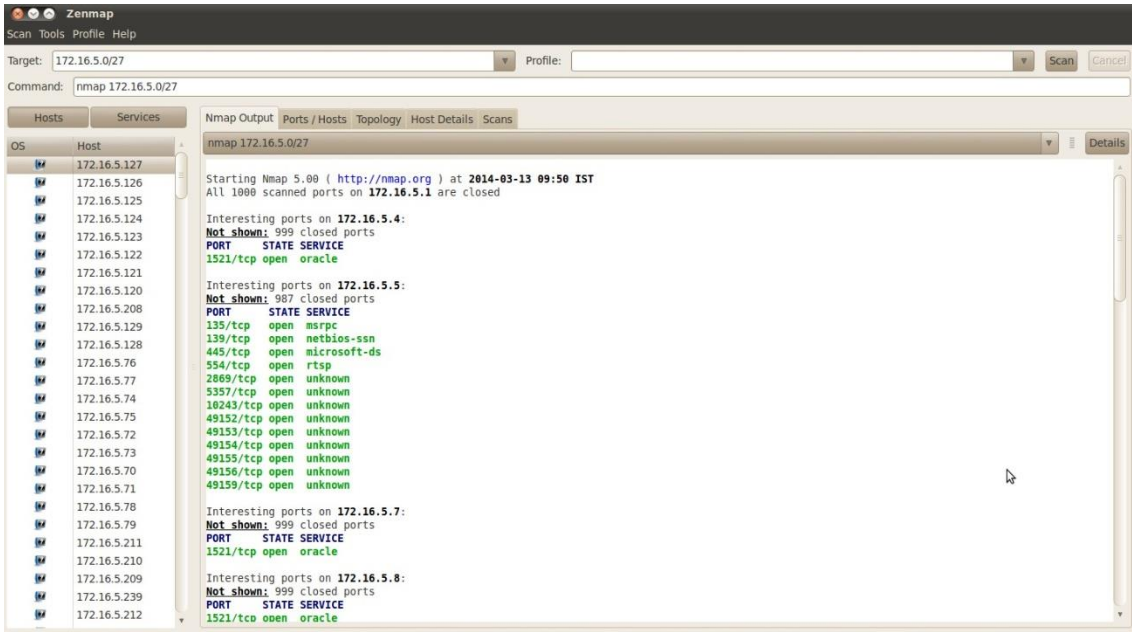


Figure 2.1: port scan

a) Can you identify what ports are open on your neighbors' system?

The below figure shows the network topology for the above scan (Click on the **Topology** tab).

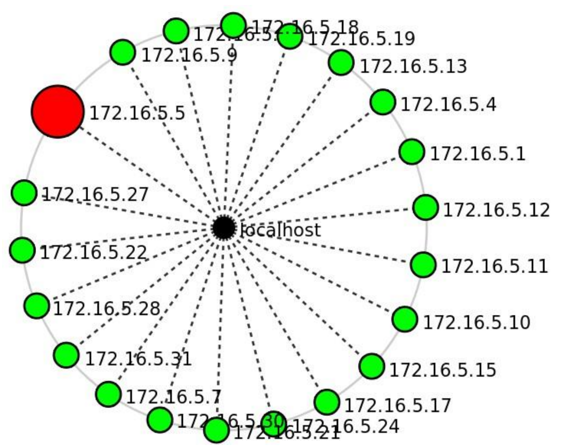


Figure 2.2: Port scan Topology

a) What is your observation from the above topology?

2. Ping scan: Performs a Ping scan for the specified IP range. It can be used to figure out which machines are up and are responding to pings.
Give the following details in zenmap:
Target: 172.16.5.0/24
Command: choose the **ping scan** from the profile dropdown
Click "Scan" button on the top right corner.

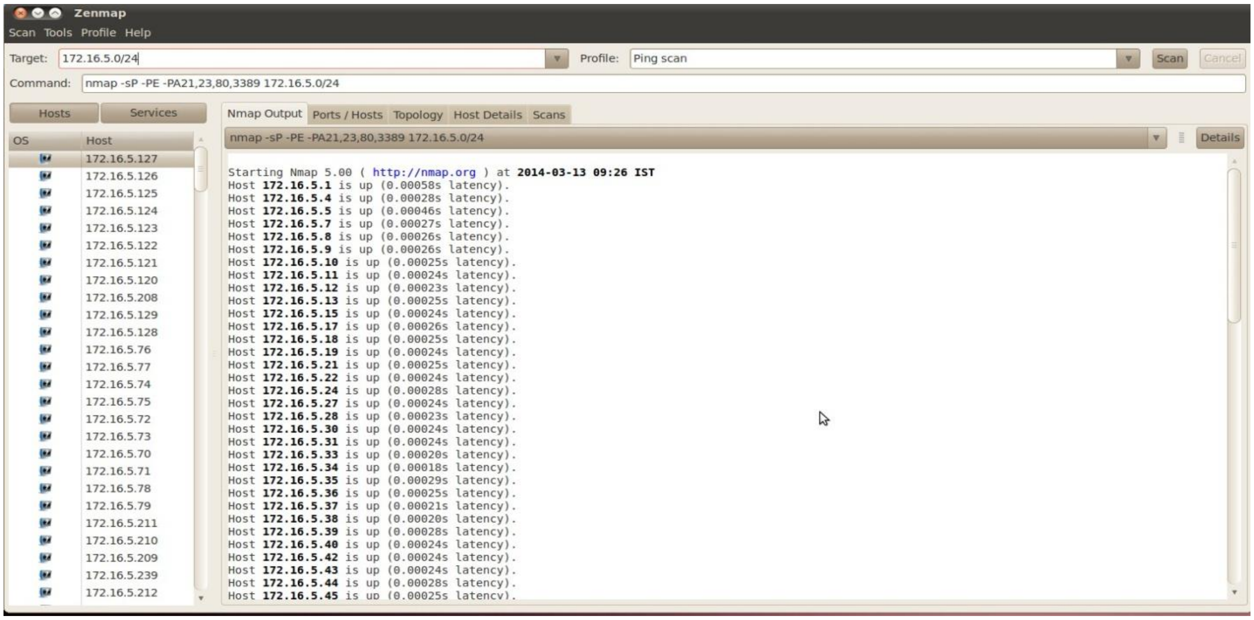


Figure 2.3: Ping scan

3. Traceroute scan: Performs traceroute operation for specified IP addresses
   (experiment with external IP's - say 173.194.36.16/28)
   Give the following details in zenmap:
   Target: 172.16.5.0/24
   Command: choose the **Quick traceroute** from the profile dropdown
   Click "Scan" button on the top right corner.

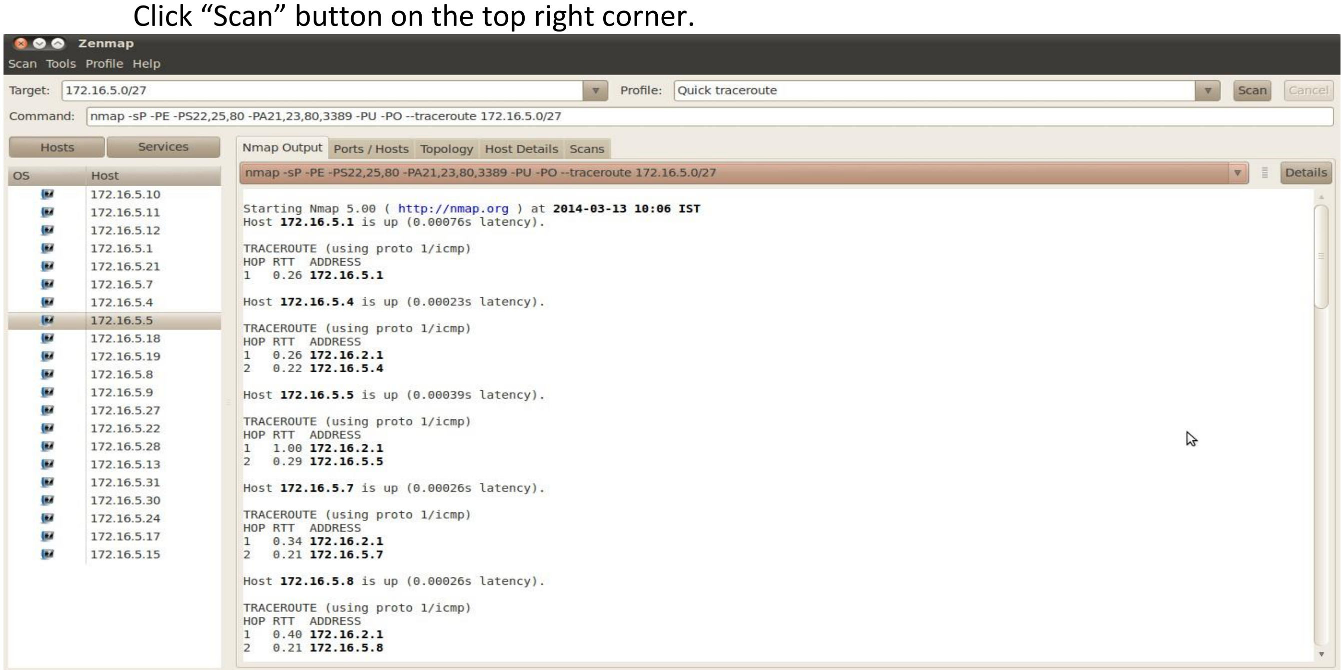


Figure 2.4: Quick Traceroute

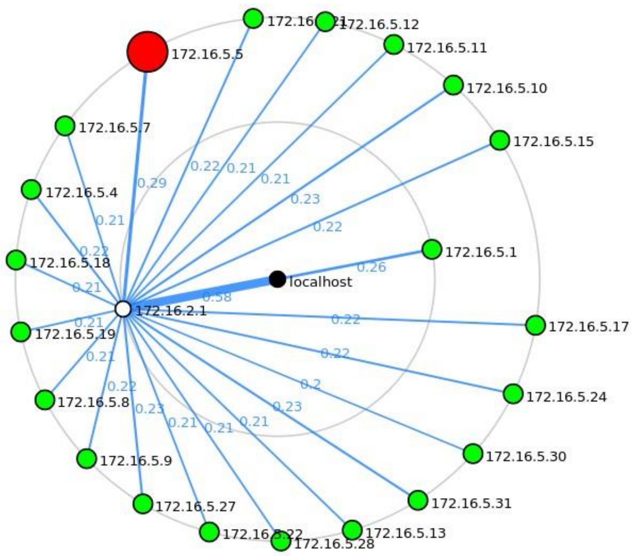


Figure 2.5: Quick Traceroute Topology

4. Intense Scan: Enables OS detection, os version, script scanning and traceroute. This is considered as an "intrusive scan". Give the following details in zenmap:

   Target: 172.16.5.0/24
   Command: choose the **intense scan** from the profile dropdown
   Click "Scan" button on the top right corner.

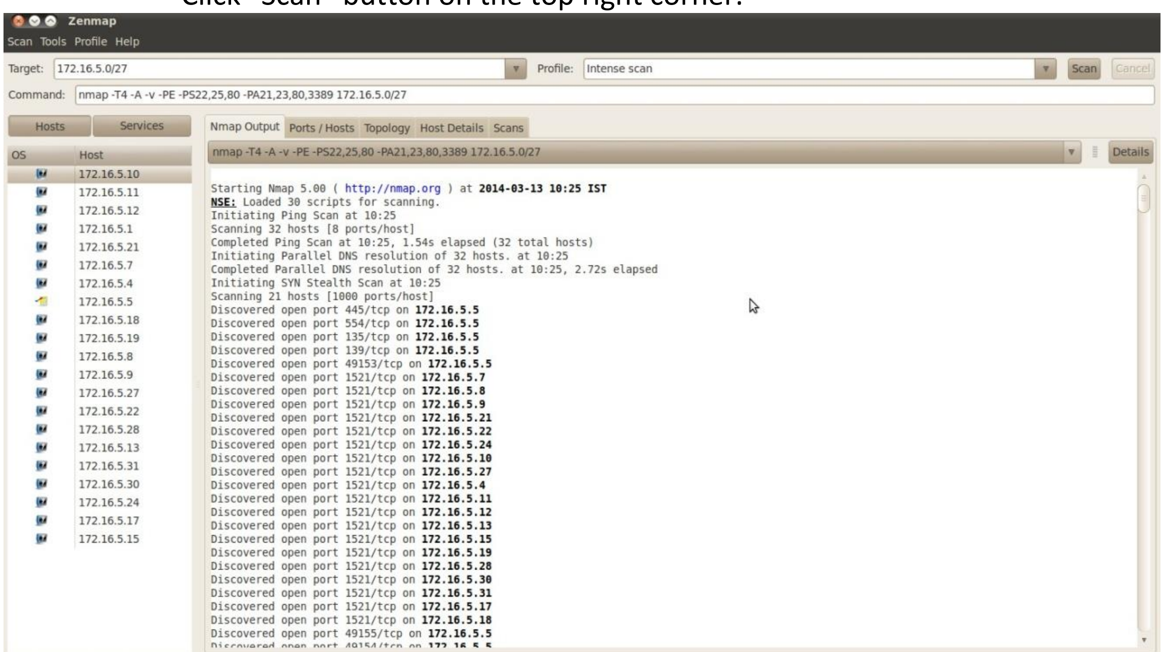


Figure 2.6: Intense scan

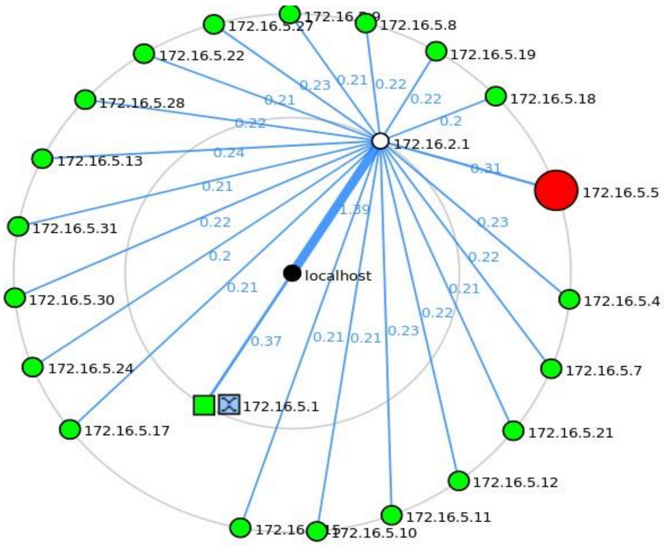


Figure 2.7: Intense scan Topology

Basic rule setting in iptables :- iptables is a user space application program that allows a system administrator to

configure the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules it stores. Different kernel modules and programs are currently used for different protocols; *iptables* applies to IPv4, *ip6tables* to IPv6, *arptables* to ARP, and *ebtables* to Ethernet frames. iptables requires elevatedprivileges to operate and must be executed by user root, otherwise it fails to function. On most Linux systems, iptables is installed as /usr/sbin/iptables and documented in its man pages which can be opened using man iptables when installed. It may also be found in /sbin/iptables, but since iptables is more like a service rather than an "essential binary", the preferred location remains /usr/sbin.

Tool used:iptables

Experiment 3:

1. To display firewall rule-set using iptables
    a. Use “-L” option in “iptables” command. By default, “filter” table is displayed.

```
root@gokul-VirtualBox:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figure 3.1 Listing firewall rules

b. Explicitly the table can be specified using “-t” option.

```
root@gokul-VirtualBox:~# iptables -L -t filter
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figure 3.2 Listing firewall rules (using filter table)

2. To block gmail server, so that, your browser cannot open gmail.com.
   a. Do a nslookup to find the IP addresses of gmail.com

```
root@gokul-VirtualBox:~# nslookup gmail.com
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:   gmail.com
Address: 173.194.36.21
Name:   gmail.com
Address: 173.194.36.22
```

Figure 3.3 nslookup for gmail.com

There are two IP addresses associated with gmail.com. Block both the IP addresses.

   b. Add a firewall rule for outgoing traffic. All TCP packets are not allowed to reach internet.

```
root@gokul-VirtualBox:~# iptables -I OUTPUT 1 -t filter -d 173.194.36.21 -p tcp -j REJECT
```

Figure 3.4 Adding a firewall rule

```
root@gokul-VirtualBox:~# iptables -L -t filter -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
REJECT     tcp  --  0.0.0.0/0            173.194.36.21        reject-with icmp-port-unreachable
```

Figure 3.5 iptables showing added firewall rule

   c. Similarly, add a firewall rule for rest of the IP addresses (Gmail server).

d. Open the browser and connect to gmail.com. What is the observation?
e. Do a ping to one of the IP addresses of Gmail.com. What is the observation?
f. Delete the above rules using "-D" option.

```
root@gokul-VirtualBox:~# iptables -D OUTPUT 1 -t filter
```

Figure 3.6 Delete a firewall rule

The above command will remove the first rule.
In a similar fashion, remove all the rules in the OUTPUT chain.
There is an option "-F" (flush) to remove rule-sets ("*iptables -F*" will remove all the rules).

3. To block all the outgoing packets to gmail.com

a. Add the firewall rule specified in the figure below.

```
root@gokul-VirtualBox:~# iptables -I OUTPUT 1 -t filter -d 173.194.36.21 -j REJECT
root@gokul-VirtualBox:~# iptables -L -t filter -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
REJECT     all  --  0.0.0.0/0            173.194.36.21        reject-with icmp-port-unreachable
```

Figure 3.7 Adding a firewall rule

b. Similarly, add rules for other IP addresses associated with gmail.com.
c. Do a ping for one of the IP addresses.

```
root@gokul-VirtualBox:~# ping 173.194.36.21 -c 5
PING 173.194.36.21 (173.194.36.21) 56(84) bytes of data.
From 10.0.2.15 icmp_seq=1 Destination Port Unreachable
From 10.0.2.15 icmp_seq=1 Destination Port Unreachable
From 10.0.2.15 icmp_seq=1 Destination Port Unreachable
From 10.0.2.15 icmp_seq=1 Destination Port Unreachable
From 10.0.2.15 icmp_seq=1 Destination Port Unreachable

--- 173.194.36.21 ping statistics ---
0 packets transmitted, 0 received, +5 errors
```

Figure 3.8 Testing with ping command

d. Delete the above rules using "-D" option.

```
root@gokul-VirtualBox:~# iptables -D OUTPUT 1 -t filter
```

Figure 3.9 Delete the firewall rule

The above command will remove the first rule.
In a similar fashion, remove all the rules in the OUTPUT chain.

Experiment-3

Aim: To learn about queue management techniques, and global routing in ns3

Study the performance of **DropTail** and RED queue management techniques:

Tail Drop, or Drop Tail, is a very simple queue management algorithm used by Internet routers, e.g., in the network schedulers, and network switches to decide when to drop packets. In contrast to the more complex algorithms like RED and WRED, in Tail Drop the traffic is not differentiated. Each packet is treated identically. With tail drop, when the queue is filled to its maximum capacity, the newly arriving packets are dropped until the queue has enough room to accept incoming traffic**.**
The name arises from the effect of the policy on incoming datagrams. Once a queue has been filled, the router begins discarding all additional datagrams, thus dropping the tail of the sequence of datagrams. The loss of datagrams causes the TCP sender to enter slow start, which reduces throughput in that TCP session until the sender begins to receive acknowledgements again and increases its congestion window. A more severe problem occurs when datagrams from multiple TCP connections are dropped, causing global synchronization; i.e., all the involved TCP senders enter slow start. This happens because, instead of discarding many segments from one connection, the router would tend to discard one segment from each connection.
Random early detection (RED), also known as random early discard or random early drop is an queueing discipline for a network scheduler suited for congestion avoidance. RED monitors the average queue size and drops (or marks when used in conjunction with ECN) packets based on statistical probabilities. If the buffer is almost empty, all incoming packets are accepted. As the queue grows, the probability for dropping an incoming packet grows too. When the buffer is full, the probability has reached 1 and all incoming packets are dropped.

Experiment 1: Compare the performance of DropTail and RED queue techniques
1. Copy Red_vs_nlred.cc file from examples directory and put all .ccfiles into the scratch folder in ns3. 2. The dropTail_vs_red.cc code simulates the following network topology

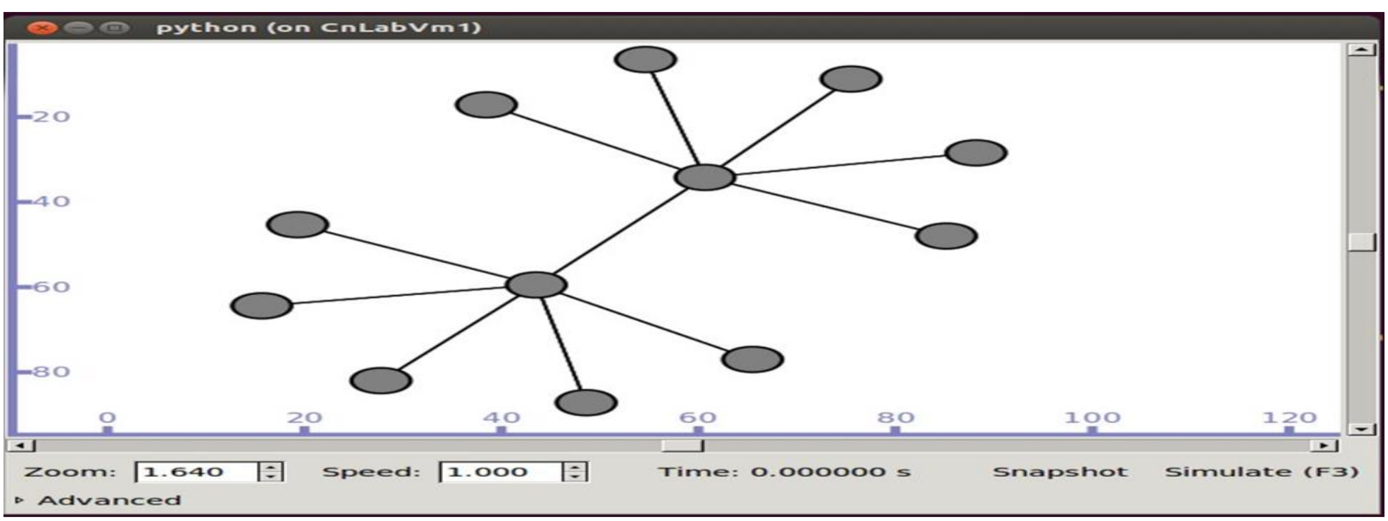
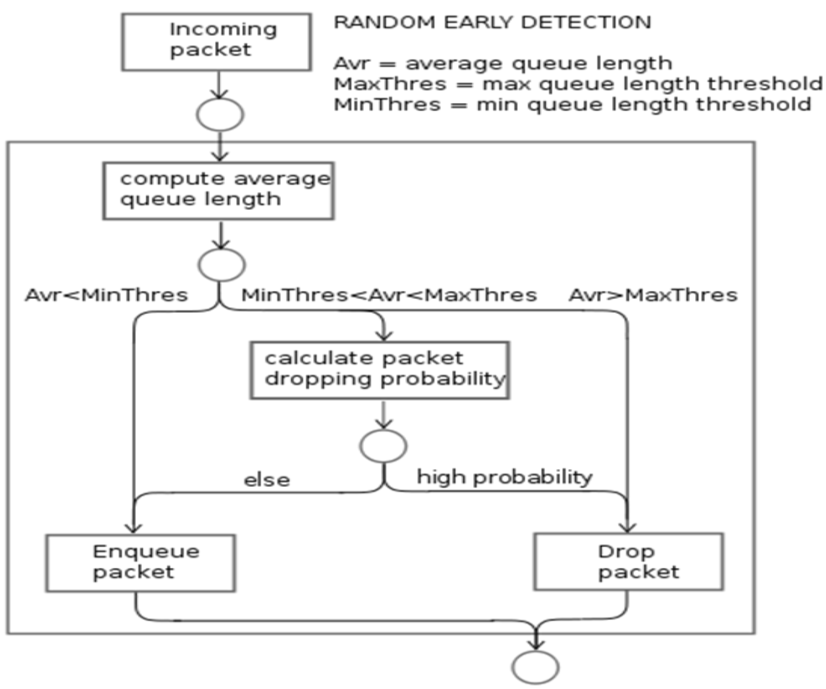


Figure.1. Topology

3. The bandwidth and delay of the bottleneck link is 1 Mbps and 50 ms. The data rate at the source nodes are higher than 1 Mbps. Since all traffic passes through a single route, there is a congestion in the network. This leads to drop in packets.
4. Copy the file to ns3_home_folder/scratch/ directory.
5. Open a terminal and navigate to ns3_home_folder.
6. Compile ns3 programs using the below command. ./waf
7. Run droptail_vs_red executable using the following command. ./waf run dropTail_vs_red vis
8. Simulator window will be opened on running the above command. Click "Simulate" button.
9. Wait for the simulation to complete. Once the simulation is completed, close the window.
10. The terminal will show the total no. of bytes received successfully at different destinations.
11. Create a new copy of droptail_vs_red.cc and change the queue to RED.
12. Run the experiment from step.4. to step.10.
13. Compare and contrast DropTail and RED queue techniques.
    a. What is the total no. of bytes received in Droptail queue technique?
    b. What is the total no. of bytes received in RED queue technique?
    c. What is the inference?

Experiment.2: Performance of RED for different link bandwidths and queue lengths



For a given network, the following parameters play a critical role in network congestion.
Traffic characteristics in source/destination hosts
1. Packet priority = Low, Medium, High
2. Traffic Type: Data, Voice, Video
3. Application Data Size: Distribution (Constant, Exponential, etc.), Application Data Size (1472 bytes, 512 bytes, etc.)

4. Inter Arrival Time: Distribution (Constant, Exponential, etc.), Mean Inter Arrival Time (micro seconds)

Link Properties
1. Distance (km)
2. Bit Error Rate (BER)
3. Physical Medium (CAT510 Mbps, E2, etc.)

Router properties:
1. Buffer size (KB): 8, 16, 32, etc.
2. Scheduling Type: FIFO, Priority
3. Queue Technique: DropTail, RED

Steps in the experiment:

1. The code simulates the following network topology.

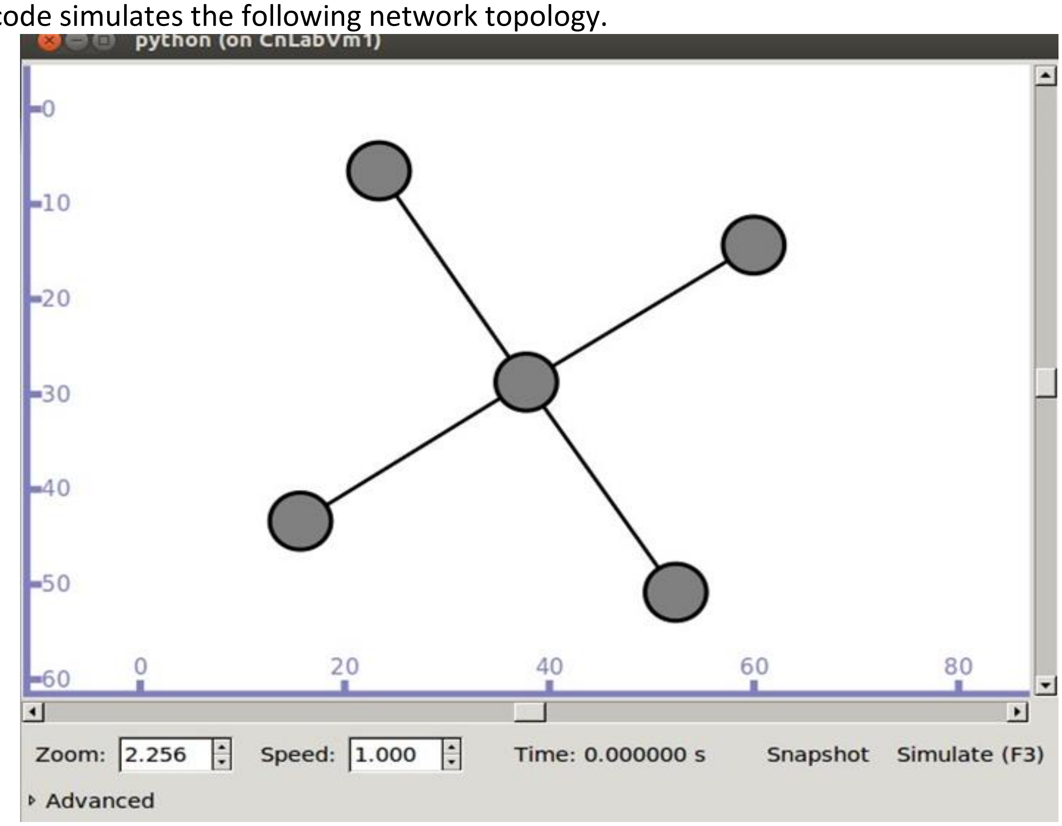


Figure. 2. Topology

2. The bandwidth and delay of the bottleneck link is 10 Mbps and 10 ms. The traffic at source and destinations are 8 Mbps, 5 Mbps, and 7 Mbps. Since, there are many sources passing through a single route (as shown in the above figure), there is a huge drop in packets.
3. Copy the file to ns3_home_folder/scratch/ directory.
4. Open a terminal and navigate to ns3_home_folder.
5. Compile ns3 programs using the below command. ./waf
6. Run redqueue executable using the following command ./waf run RedQueueStats vis

7. Simulator window will be opened on running the above command. Click “Simulate” button. 8. Wait for the simulation to complete. Once the simulation is completed, close the window.
9. The terminal will show the total no. of dropped packets. The following information is displayed.
   a. Packets drop after crossing avg. threshold level (High prob.)
   b. Packets drop after crossing max. threshold level (QueueAvg > MaxQueue)
   c. Packets drop after crossing the queue length (Queue is full)
10.Change the bandwidth of the bottleneck link to 15 Mbps. Compile the code and run the experiment.
   a. What is your observation for step.9
11.Change the bandwidth of the bottleneck link to 20 Mbps. Compile the code and run the experiment.
   a. What is your observation for step.9
12.From the steps 9, 10, and 11
   a. What is the inference?
   b. What is the minimum and maximum threshold value?
13.Change the bandwidth of the bottleneck link to 2 Mbps. Default values of MinThreshold , MaxThreshold and QueueLimit are 5, 15 and 25. Change MinThreshold less than 5 and MaxThreshold less than 15. Compile the code and run the experiment.
   a. What is your observation for step.9

Analyse the effect of broken links on routing table:

Routing is the process of selecting best paths in a network. In the past, the term routing was also used to mean forwarding network traffic among networks. However this latter function is much better described as simply forwarding. Routing is performed for many kinds of networks, including the telephone network (circuit switching), electronic data networks (such as the Internet), and transportation networks.

In packet switching networks, routing directs packet forwarding (the transit of logically addressed network packets from their source toward their ultimate destination) through intermediate nodes. Intermediate nodes are typically network hardware devices such as routers, bridges, gateways, firewalls, or switches. General purpose computers can also forward packets and perform routing, though they are not specialized hardware and may suffer from limited performance. The routing process usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the router's memory, is very important for efficient routing

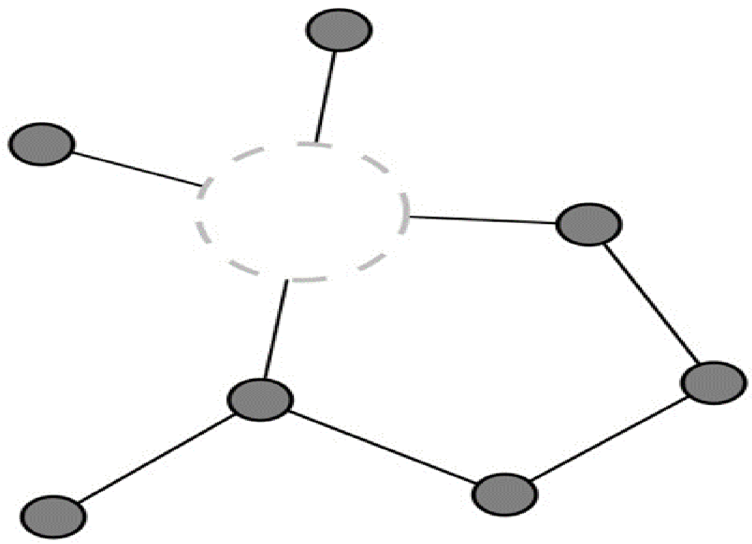
When applying link state algorithms, a graphical map of the network is the fundamental data used for each node. To produce its map, each node floods the entire network with information about the other nodes it can connect to. Each node then independently assembles this information into a map. Using this map, each router independently determines the least cost path from itself to every other node using a standard shortest paths algorithm such as Dijkstra's algorithm.

Experiment.3: Dynamic Global Routing

We will examine global routing in a mixed environment with Point to Point links and CSMA/CD channel.

Copy the dynamic_global_routing.cc file from examples into scratch folder.

The following topology is created:



1. Run the simulation from ns3 home folder
   a. ./waf
   b. ./waf run globalrouting (use 'vis' to enable visualization)
2. Data is transferred from n1 to n6. Identify n1 and n6 correctly from the visualization.
3. At presimulation time, global routes configured. Look for the following line in the code: aa a.Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
4. The shortest path from n1 to n6 is via the direct pointtopoint link. This will be the default choice.
5. At time 1s, CBR traffic flow from n1 to n6 is started
6. At time 2s,the n1 pointtopoint interface goes down. Through what path will the packets get diverted?
   a. Under what other circumstances (apart from the interface going down) could the path of the packets dynamically change?
7. At time 4s, the n1/n6 interface is reenabled to up. Now what will be the path taken by the packets between n1n6?
8. At time 6s, the n6n1 pointtopoint Ipv4 interface is set to down (note, this keeps the pointtopoint link "up" from n1's perspective). Observe the change of path of the packets (NOTE: observe the visualization as well as the corresponding pcap files)
9. At time 8s, the interface comes up. The older path is restored.
10. At time 10s, the first flow is stopped.
11. At time 11s, a new flow started, but to n6's other IP address (the one on the n1/n6 p2p link)
12. At time 12s, the n1 interface down between n1 and n6 is put down. Packets will be diverted to the alternate path
13. At time 14s, the n1/n6 interface is reenabled. This will change routing back to n1n6 since the

interface up notification will cause a new local interface route, at higher priority than global routing
14.At time 20s, the second flow stopped and simulation ends.

Experiment-4

Aim: To learn about broadcasting, multicasting, and bridging in a Local AreaNetwork using ns-3.

First remove old .cc files from your scratch folder. Copy first.cc and second.cc from examples->Tutorial into scratch folder.

Carrier Sense Multiple Access (CSMA)

Carrier Sense Multiple Access/Collision Detect (CSMA/CD) is the protocol for carrier transmission access in Ethernet networks. On Ethernet, any device can try to send a frame at any time. Each device senses whether the line is idle and therefore available to be used. If it is, the device begins to transmit its first frame. If another device has tried to send at the same time, a collision is said to occur and the frames are discarded. Each device then waits a random amount of time and retries until successful in getting its transmission sent.

Experiment 1: One Point-to-Point link with one CSMA channel with four nodes

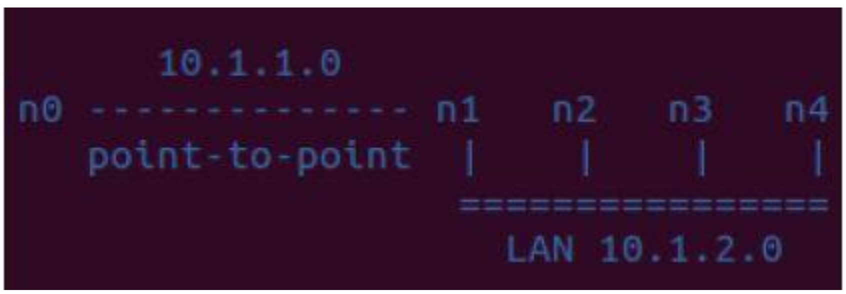


Figure.1. Default LAN Topology

In *p oint2point-csma.cc* file you have the topology as given above. This program builds a one point-to-point channel from *n 0* to one LAN segment over csma channel. But when you run *point2point-csma. cc* program, ns3does not visualize it in the same way. The visualization by ns-3looks like the figure below.

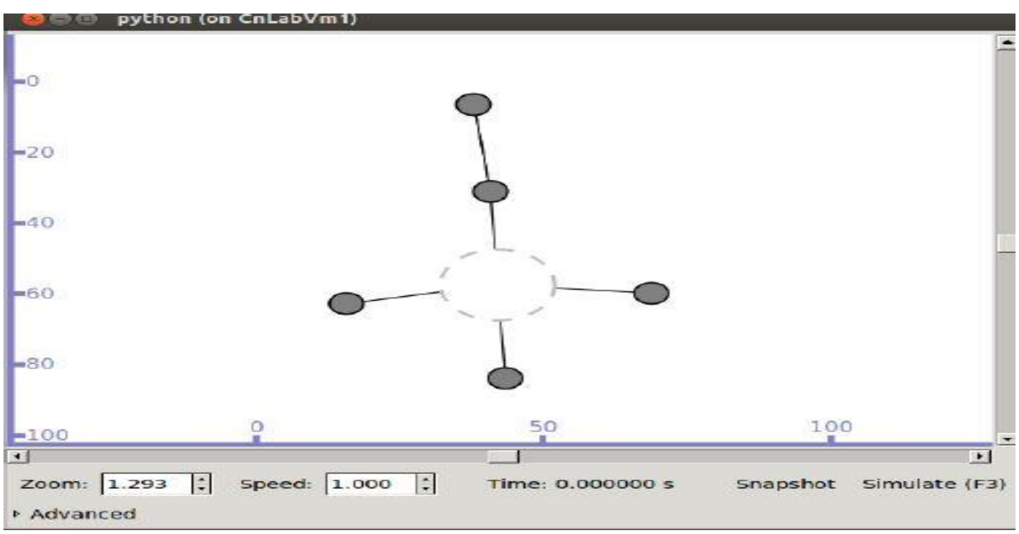


Figure.2. Simulation of LAN Topology

In the figure above, the dotted circle in the center is the CSMA channel. The top most node is *n0*.

Pcap captures for the simulation are enabled by default.

1. Do you see ARP communication in the Point-to-Point link?
2. What about the CSMA LAN?

Write down your observations.

For the experiment above, extend it to create the topology given below:

```
                 10.1.3.0           10.1.1.0
 n7   n6   n5 -------------- n0 ---------------- n1   n2   n3   n4
 |    |    |  point-to-point    point-to-point   |    |    |    |
 ==========                                      ================
LAN 10.1.4.0                                       LAN 10.1.2.0
```

Figure.3. Topology2

Experiment 2: IP broadcasting over two CSMA channels

Open the IpBroadcastCSMA.cc file. It uses the following topology:

```
Network topology
    ======================
         |          |      |
  n3    n0     n1    n2     n4===========
  |     |      |                  |
 ===============                  n5
```

Figure.4. Default Broadcast Topology

In the figure below, *n0* is the node in the middle of the two CSMA channels. *n0* originates UDP broadcast to specified LAN segment. In the IpBroadcastCSMA.cc file, line 101 has OnOffHelper which takes the IP address as a parameter. The code given to you has 1 0.1.3.255 as the IP address, which is the broadcast IP of the CSMA channel at the bottom in the given figure. If you want *n 0* to broadcast to both CSMA channels, comment out line 101 and uncomment line 103. This will make the broadcast IP as 255.255.255.255 .

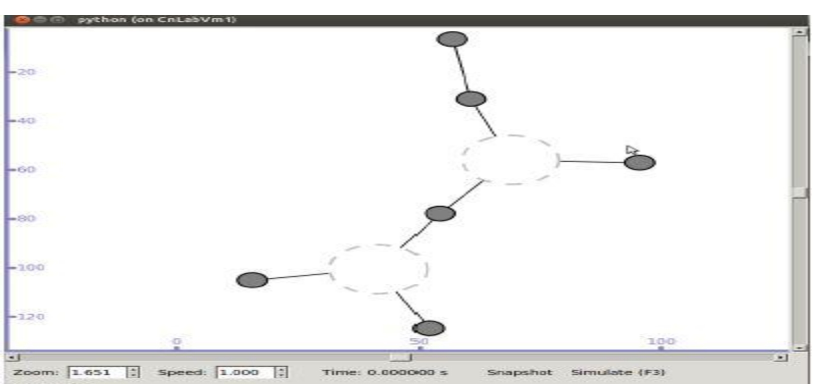


Figure.5. Simulation of Broadcast Topology

Write down your observations.

Experiment 3 : IP multicasting over two CSMA channels

Open IpMulticastCSMA.cc file. It uses the following topology.

Figure.6. Default Multicast Topology

As given in line 101 of the code, *n0* is the multicast source

Ipv4Address multicastSource(“10.1.1.1”)

This topology is similar to the previous one. *n2*is the node in the middle of the two CSMA channels.

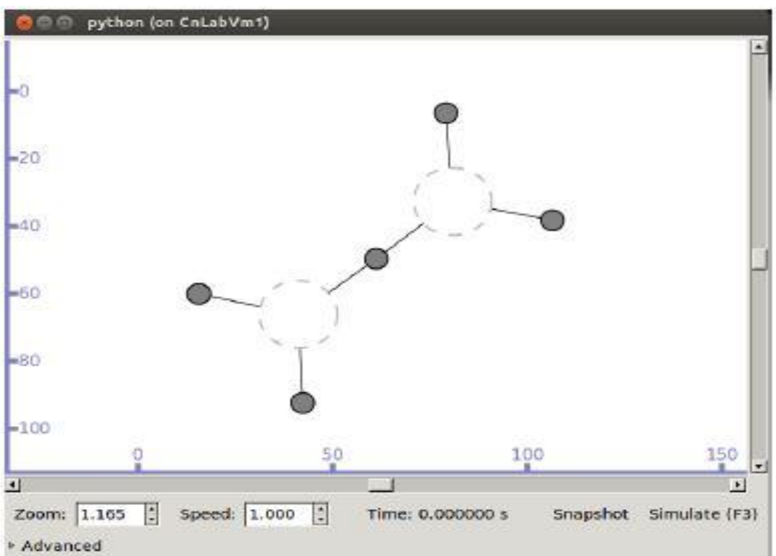


Figure.7. Simulation of Multicast Topology

Change the multicast source node to *n3* . Send multicast data to nodes *n0, n1* , and *n4* .

Write down your observations.

Experiment 4: Bridging over CSMA and with one intermediate router
Open BridgingOneHop.cc file. It uses the following topology.

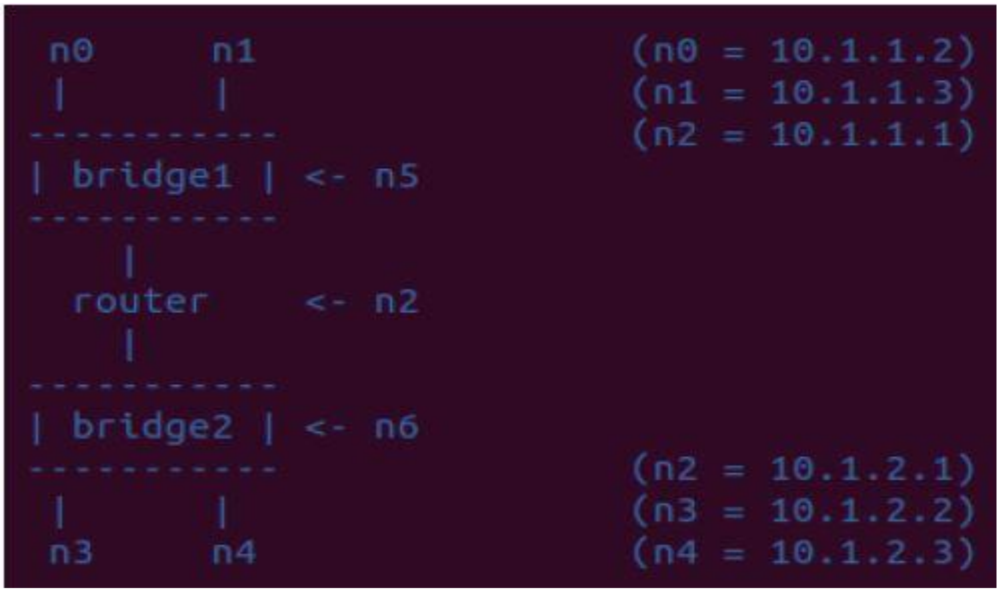


Figure.8. Default Bridging Network

In the figure below, *n2* is the router node in the middle. Application data is transmitted from *n0* to *n1* and from *n3* to *n0* . The Data rate for n0>n1 transmission is 500Kb/s (see line 165 of code), and Data rate for n3>n0 transmission is 100Kb/s (see line 181 of code).
In the figure below, *n2* is the router node in the middle. Application data is transmitted from *n0* to *n1* and from *n3* to *n0* . The Data rate for n0>n1 transmission is 500Kb/s (see line 165 of code), and Data rate for n3>n0 transmission is 100Kb/s (see line 181 of code)

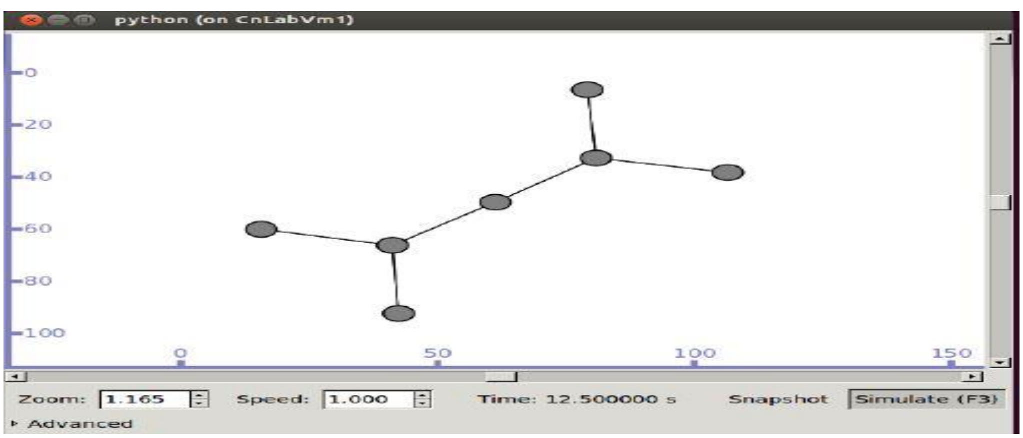


Figure.9. Simulation of Bridging Network
This shows two broadcast domains, each interconnected by a bridge with a router node interconnecting the Layer2 broadcast domains.

While running the simulation, observe the 'interface statistics' (right click on the node) for n0 and the bridging node n5. Can you observe 'IPv4 Routing Table' for the bridging nodes (n5 and n6)?
Write down your observations.

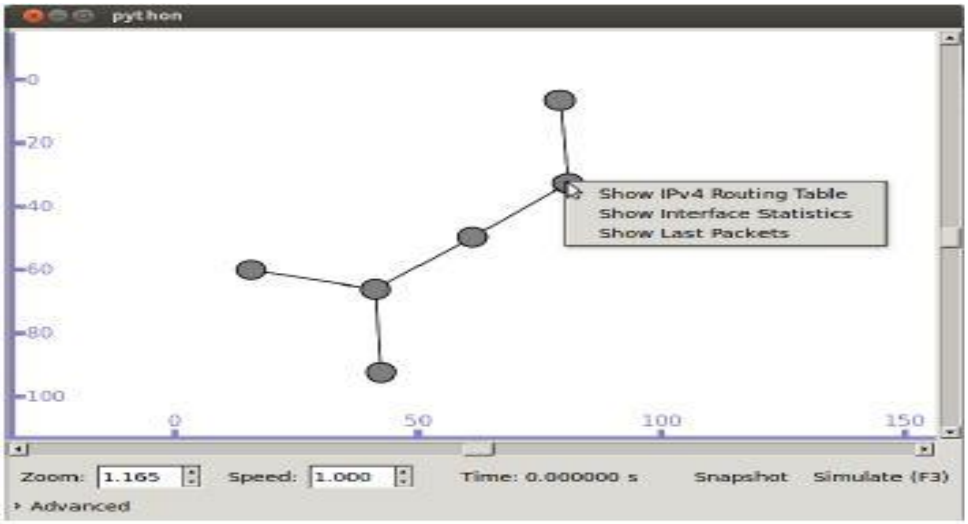


Figure.10. Observing Node statistics

At n5, we can observe the following statistics:

Statistics for node 5

Interface	Tx Packets	Tx Bytes	Tx pkt/1s	Tx bit/1s	Rx Packets	Rx Bytes	Rx pkt/1s	Rx bit/1s
(interface 0)	2	128	0.0	0.0	219	121214	0.0	0.0
(interface 1)	219	121214	0.0	0.0	1101	612876	0.0	0.0
(interface 2)	1100	612812	0.0	0.0	1	64	0.0	0.0
(interface 3)	0	0	0.0	0.0	0	0	0.0	0.0

Figure.11. Statistics for node 5

The Received bytes at interface 1 are maximum. Can you correctly identify interface 1 of node 5 on your visualization? Can you identify its interface 2 as well? Why does interface 3 has no Transmitted or Received bytes?
Write down your observations.

Experiment-5

Aim: To learn about Wi-fi and Mobile Ad-hoc topologies with ns-3.

IEEE 802.11 wireless LANs use a media access control protocol called Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA).

Wi-Fi systems are half duplex shared media configurations, where all stations transmit and receive on the same radio channel. The fundamental problem this creates in a radio system is that a station cannot hear while it is sending, and hence it is impossible to detect a collision. Because of this, the developers of the 802.11 specifications came up with a collision avoidance mechanism called the Distributed Control Function (DCF).

According to DCF, a Wi-Fi station will transmit only if it thinks the channel is clear. All transmissions are acknowledged, so if a station does not receive an acknowledgement, it assumes a collision occurred and retries after a random waiting interval.

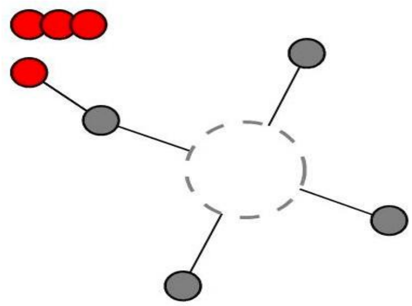
Experiment 1:

1. third.cc file creates the following topology:

```
Default Network Topology

  Wifi 10.1.3.0
                AP
 *    *    *    *
 |    |    |    |    10.1.1.0
n5   n6   n7   n0 -------------- n1   n2   n3   n4
                  point-to-point  |    |    |    |
                                 ================
                                   LAN 10.1.2.0
```

In ns-3, the visualization will appear similar to the figure below:



You may increase the number of Wi-fi devices by specifying it at run-time in the following manner:

```
./waf --run 'wifi-example --nWifi=10' --vis
```

(18 is the hard-coded upper limit)

You may observe the pcap files generated at different nodes. Packet captures are enabled in the line numbers 172-174 of the code:

```
pointToPoint.EnablePcapAll ("wifi-p2p"); phy.EnablePcap ("wifi-ap", apDevices.Get (0));
csma.EnablePcap ("wifi-csma", csmaDevices.Get (0), true);
```

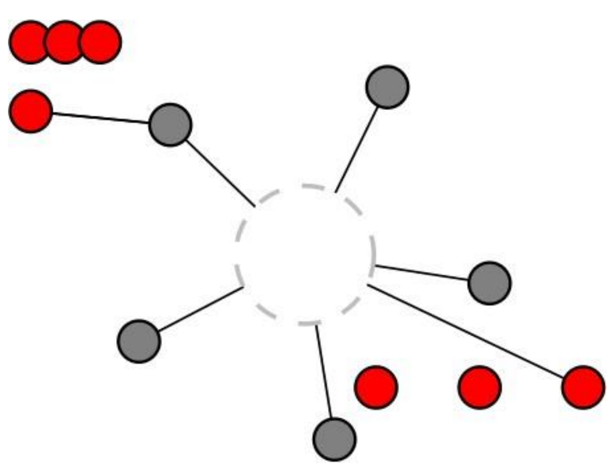
Since no data applications are enabled, you do not see any data in the pcap files apart from broadcast messages by the Wi-fi A.P. In pcap files generated at the A.P., can you observe the beacon frames and acknowledgements?

2. Create a UDP Echo Client-Server application for the above topology. The last node on the CSMA LAN should be your Echo Server. Configure any of the Wi-fi devices as your Client.

Comment out line number 152-166 to implement the above.
Observe UDP server port number and other client attributes.
After running the Echo Client-Server application, observe the fresh pcap files generated.

Experiment 2: Extending the previous topology

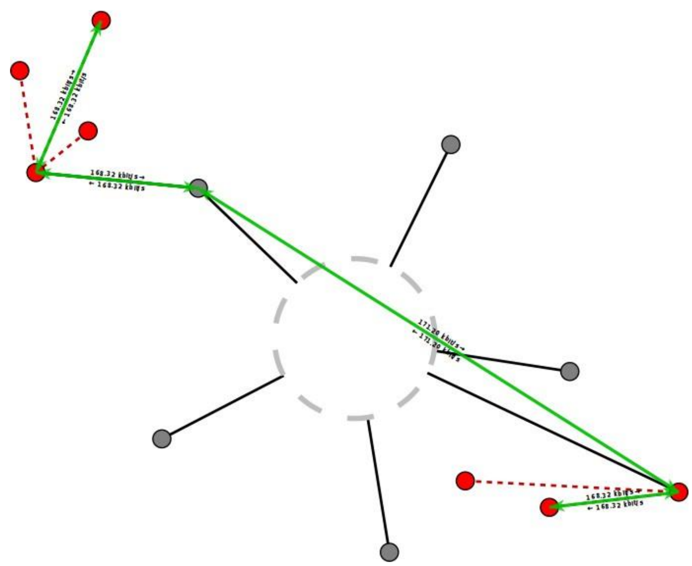
Extend the above topology so as to create a Wi-fi Access Point on one of the CSMA LAN nodes. Connect three Wi-fi nodes to this Access Point as shown in the figure below:



Some hints to help you extend the topology:

- Do not install the Internet stack twice on any node.
- Remember to give your new Wi-fi Access Point a SSID which is different from the previous one.
- The Wi-fi devices and their Access Point must be in the same IP subnet.
- In order to have your new Wi-fi nodes physically apart from the old nodes, you will need to set appropriate values for the SetPositionAllocator method of the MobilityHelper class.

3. Further, modify your UDP Echo Server-Client application by configuring the Server as a Wi-fi device on the new A.P.. (Hint: the present code will not suffice for this implementation because the Wi-fi devices have no 'IPv4Interface' associated with them)
4. The final topology will look like the figure below:
5.



6.
7.
8. Observe the fresh pcap files generated. You should be able to see data transfer between the two Wi-fi devices.

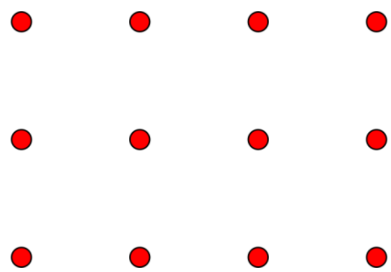
AODV (Ad-hoc On-Demand Distance Vector)

Ad hoc On-Demand Distance Vector (AODV) Routing is a routing protocol for mobile ad hoc networks (MANETs) and other wireless ad hoc networks.

The AODV Routing Protocol uses an on-demand approach for finding routes, that is, a route is established only when it is required by a source node for transmitting data packets. It employs destination sequence numbers to identify the most recent path. In AODV, the source node and the intermediate nodes store the next-hop information corresponding to each flow for data packet transmission. In an on-demand routing protocol, the source node floods the RouteRequest packet in the network when a route is not available for the desired destination. It may obtain multiple routes to different destinations from a single RouteRequest. The major difference between AODV and other on-demand routing protocols is that it uses a destination sequence number (DestSeqNum) to determine an up-to-date path to the destination. A node updates its path information only if the DestSeqNum of the current packet received is greater or equal than the last DestSeqNum stored at the node with smaller hopcount.

Experiment 3: Understanding the AODV routing protocol for Mobile adhoc networks

The My_aodv.cc file generates a topology of 12 mobile nodes, 4 in each line.



The nodes are separated by a distance ‘step’ specified in the code. The initial value given to ‘step’ is 100m. The step variable can be varied from 80 to 120. As the ‘step’ size increases, you will observe that neighboring nodes are not able to communicate to each other directly (the link between then breaks).

Thus the nodes will use AODV routing to figure out alternate routing paths. Observe the aodv.routes file, which gives the AODV routing table at each node.

```
ipc3@ipc3-ThinkCentre-A58:~/ns3_cnlab/ns-allinone-3.22/ns-3.22$ cat aodv.routes
Node: 0 Time: 8s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.2        10.0.0.2        10.0.0.1        UP      2.02            1
10.0.0.5        10.0.0.5        10.0.0.1        DOWN    13.00           1
10.0.0.12       10.0.0.2        10.0.0.1        IN_SEARCH       12.02           5
10.255.255.255  10.255.255.255  10.0.0.1        UP      9223372028.85   1
127.0.0.1       127.0.0.1       127.0.0.1       UP      9223372028.85   1

Node: 1 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.1        10.0.0.2        UP      2.02            1
10.0.0.3        10.0.0.3        10.0.0.2        UP      1.02            1
10.0.0.6        10.0.0.6        10.0.0.2        UP      12.02           1
10.0.0.12       10.0.0.6        10.0.0.2        DOWN    12.02           4
10.255.255.255  10.255.255.255  10.0.0.2        UP      9223372028.85   1
127.0.0.1       127.0.0.1       127.0.0.1       UP      9223372028.85   1

Node: 2 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.2        10.0.0.3        DOWN    12.82           2
10.0.0.2        10.0.0.2        10.0.0.3        UP      1.01            1
10.0.0.4        10.0.0.4        10.0.0.3        UP      11.82           1
10.0.0.7        10.0.0.7        10.0.0.3        DOWN    12.82           1
10.255.255.255  10.255.255.255  10.0.0.3        UP      9223372028.85   1
127.0.0.1       127.0.0.1       127.0.0.1       UP      9223372028.85   1

Node: 3 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.3        10.0.0.4        DOWN    12.04           5
10.0.0.3        10.0.0.3        10.0.0.4        UP      12.04           1
10.0.0.8        10.0.0.8        10.0.0.4        UP      9.04            1
10.255.255.255  10.255.255.255  10.0.0.4        UP      9223372028.85   1
```

References

1. http://www.nsnam.org/
2. http://en.wikipedia.org/wiki/Wi-Fi

http://en.wikipedia.org/wiki/AODV

Experiment-6

AIM:-To Introduce Socket Programming in TCP and UDP

TCP Socket Programming:

A socket is the mechanism that most popular operating systems provide to give programs access to the network. It allows messages to be sent and received between applications (unrelated processes) on different networked machines. The sockets mechanism has been created to be independent of any specific type of network.

A socket address is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension. Based on this address, internet sockets deliver incoming data packets to the appropriate application process or thread.

An Internet socket is characterized by a unique combination of the following:

1. Local socket address: Local IP address and port number

2. Remote socket address: Only for established TCP sockets. This is necessary since a TCP server may serve several clients concurrently. The server creates one socket for each client, and these sockets share the same local socket address from the point of view of the TCP server.

3. Protocol: A transport protocol (e.g., TCP, UDP, raw IP, or others). TCP port 53 and UDP port 53 are consequently different, distinct sockets.

Tools used: gedit, terminal

Experiment 1: Working of a TCP concurrent server

1. Create "tcp_client.c" and "tcp_server.c".
2. Compile server first (as shown below). gcc -o server tcp_server.c
3. Similarly, compile the client using the following command. gcc -o client tcp_client.c
4. Run the server using the below command. After running, the server would wait for an incoming connection (as shown in the Figure.1)

./server

```
suchetana@suchetana-HP-Pro-3330-MT:~$ cd ../suchetana/Dropbox/Network/Network\ Lab/Lab4_Code/tcp_concurrent/
suchetana@suchetana-HP-Pro-3330-MT:~/Dropbox/Network/Network Lab/Lab4_Code/tcp_concurrent$ gcc -o server tcp_server.c
suchetana@suchetana-HP-Pro-3330-MT:~/Dropbox/Network/Network Lab/Lab4_Code/tcp_concurrent$ gcc -o client tcp_client.c
suchetana@suchetana-HP-Pro-3330-MT:~/Dropbox/Network/Network Lab/Lab4_Code/tcp_concurrent$ ./server
Server running...waiting for connections.
```

Fig1. Server waiting for connections

5.On separate terminal window, run the client using
./client *<server IP or localhost>*

```
suchetana@suchetana-HP-Pro-3330-MT:~/Dropbox/Network/Network Lab/Lab4_Code/tcp_c
oncurrent$ ./client 127.0.0.1
_
```

Fig2. Client waiting for User input

6.Enter a character as input and it will be echoed back by the server (Refer Figure.3)

```
suchetana@suchetana-HP-Pro-3330-MT:~/Dropbox/Network/Network Lab/Lab4_Code/tcp_c
oncurrent$ ./client 127.0.0.1
h
String received from the server: h
i
String received from the server: i
```

Fig3. Client input-output

```
suchetana@suchetana-HP-Pro-3330-MT:~/Dropbox/Network/Network Lab/Lab4_Code/tcp_concurrent$ ./server
Server running...waiting for connections.
Received request...
Child created for dealing with client requests
Data received from and resent to the client:h
Data received from and resent to the client:i
```

Fig4. Server output

7.To stop the server and client, click “Ctrl +C” in their respective terminals. (To start the server again, wait for a couple of seconds).

Experiment 2: Modification of the TCP Client-Server programs

1. Download “tcp_client.c” and “tcp_server.c” from the CMS Website
2. Modify the filenames as “tcp_client_n.c” and “tcp_server_n.c”
3. Modify the program such that the client sends a string as a message to the server. Make sure the server echoes back the same string. (Hint: use buffer to handle the string exchanges by server-client and modify read and write functions).
4. Compile server first (as shown below). gcc -o server_ntcp_server_n.c
5. Similarly, compile the client using the following command. gcc -o client_ntcp_client_n.c
6. Run the server using the below command.

   ./server_n
7. The server will start, waiting for a client to connect. On a separate terminal window, run the client using

   ./client_n *<server IP or localhost>*

8. Type “hi I am client!” in the client terminal window. As you see the figure.6, the same message is echoed back from the server.

Fig5. Client sending string

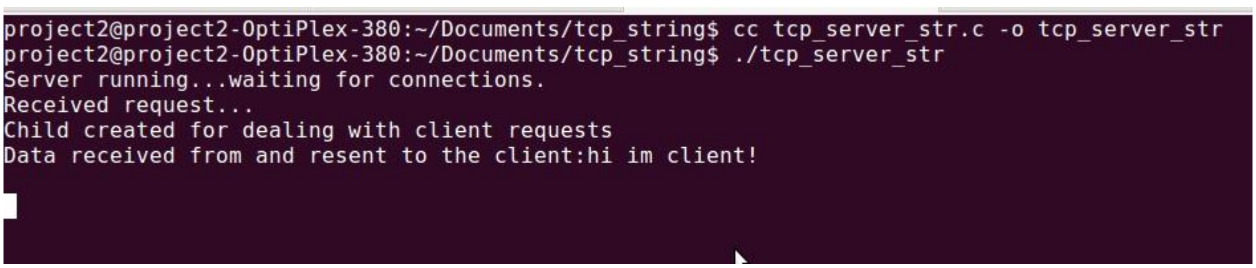


Fig6: Server receiving String

9. To stop the server and client, click “Ctrl +C” in their respective terminals. (To start the server again, wait for a couple of seconds).

Questions

Answer the following questions based on your understanding of the experiments.

1. Which field in the *socket* function specifies the type of transport layer protocol (like TCP, UDP, etc.)?
2. What is the IP address and port no. of the server?
3. What is the purpose of bind function?
4. Which of the functions mentioned below are blocking calls?
    a. socket
    b. connect
    c. bind
    d. listen
    e. accept
    f. send
    g. recv
    h. close
5. Which function in the client program involved in connection establishment?
6. Which function in the server program involved in connection establishment?
7. send and recv functions are analogous to writing to a file and reading from a file. (T/F)
8. What is a concurrent server?

UDP Socket Programming:

A datagram socket is a type of connectionless network socket, which is the sending or receiving point for packet delivery services. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may arrive in any order and might not arrive at the receiving computer.

A datagram socket provides a symmetric data exchange interface without requiring connection establishment. Each message carries the destination address.

Tools used: gedit, terminal

Experiment 3: Working of an UDP Client-Server program

1. Download “udp_client.c” and “udp _server.c” from the CMS Website.
2. The programs are partially complete. Complete the rest of the program, so that, it compiles and runs successfully (Implement echo server described in Experiment.2.)
3. Compile server first (as shown below). gcc -o udp_serverudp_server.c
4. Similarly, compile the client using the following command. gcc -o udp_clientudp_client.c
5. Run the server using the below command.

   ./udp_server
6. The server will start, waiting for a client to connect. On a separate terminal window, run the client using

   ./udp_client *<server IP or localhost>*
7. Enter a character as input and it will be echoed back by the server (Refer Figure.7)

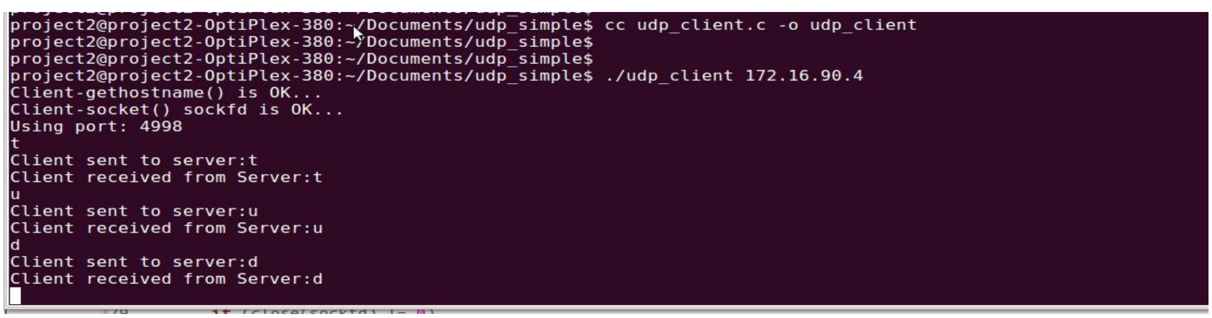


Fig7: Client sending and receiving a character

8. To stop the server and client, click “Ctrl +C” in their respective terminals. (To start the server again, wait for a couple of seconds).

Experiment 4: Modification of the UDP Client-Server programs

1. Download “udp_client.c” and “udp_server.c” from the CMS Website
2. Modify the filenames as “udp_client_n.c” and “udp_server_n.c”

3. Modify the program such that the client sends a string as a message to the server. Make sure the server echoes back the same string. (Hint: use buffer to handle the string exchanges by server-client and modify read and write functions).
4. Compile server first (as shown below). gcc -o udp_server_nudp_server_n.c
5. Similarly, compile the client using the following command. gcc -o udp_client_nudp_client_n.c
6. Run the server using the below command.

   ./udp_server_n
7. The server will start, waiting for a client to connect. On a separate terminal window, run the client using

   ./udp_client_n *<server IP or localhost>*
8. Type "hello" in the client terminal window. As you see the Figure.8, the same message is echoed back from the server.

```
project2@project2-OptiPlex-380:~/Documents/UDP_string$ ./udp_client 172.16.90.4
Client-gethostname() is OK...
Client-socket() sockfd is OK...
Using port: 45678
hello
Client sent to server:hello

String received from the server: hello
```

Fig8: Client sending and receiving a string

9. To stop the server and client, click "Ctrl +C" in their respective terminals. (To start the server again, wait for a couple of seconds).

<u>Questions</u>

Answer the following questions based on your understanding of the experiment.

1. Order the sequence of operations in an UDP socket communication.

Client functions	Server functions
Close	Close
Socket	Socket
Sendto	Sendto
Recvfrom	Recvfrom
	Bind

2. What is the difference between UDP and TCP echo servers?

References

➢ Unix Manual: http://man7.org/linux/man-pages/man2/socket.2.html

Experiment-7

Aim: Observations of Transmission Control Protocol (TCP) Connection states, Flags and Flow control.

TCP Connection States:

TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (*connection establishment*) before entering the data transfer phase. After *data transmission* is completed, the *connection termination* closes established virtual circuits and releases all allocated resources.

A TCP connection is managed by an operating system through a programming interface that represents the local end-point for communications, the Internet socket. During the

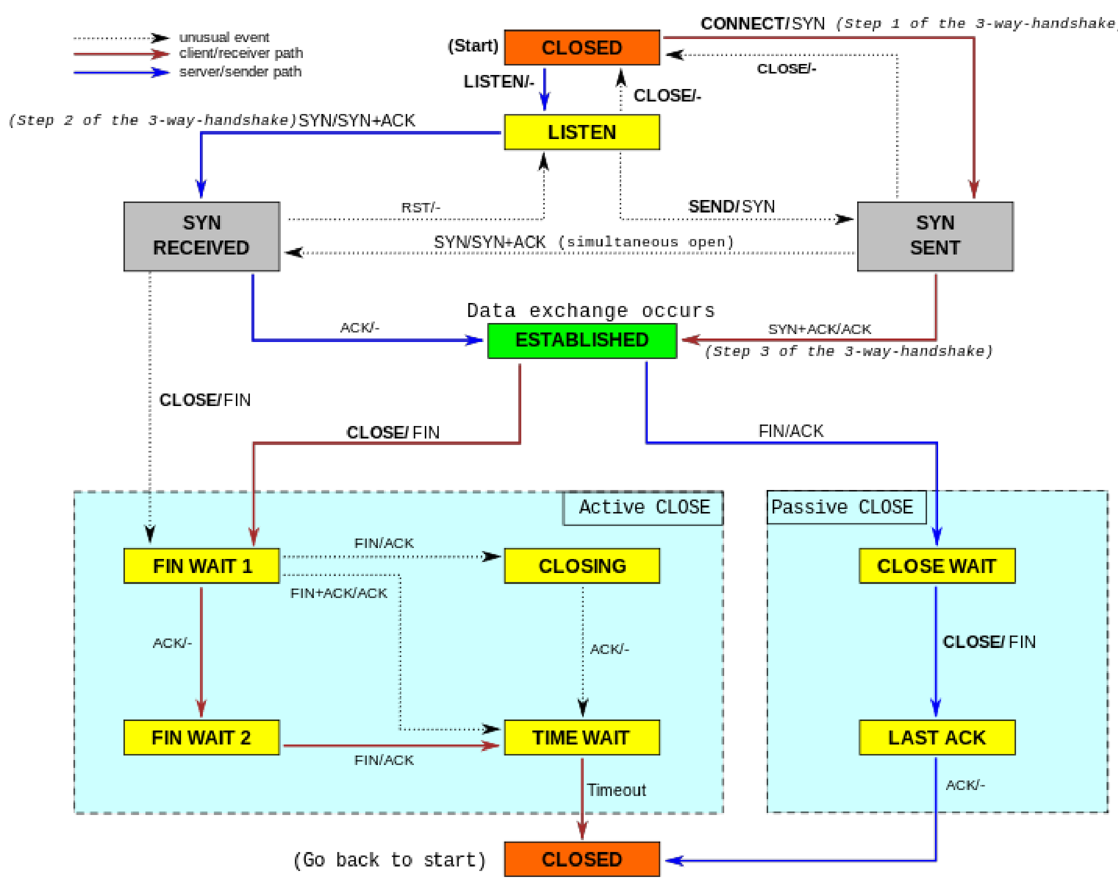


Fig.1: TCP State Diagram

lifetime of a TCP connection the local end-point undergoes a series of state changes:

Tools used: Wireshark, netstat.

Experiment 1: Observation of TCP connection states

1. Download “tcp_client.c” and “tcp_server.c” from the CMS Website

2. Compile server first (as shown below).

   gcc -o tcp_server tcp_server.c

3. Similarly, compile the client using the following command.

   gcc -o tcp_client tcp_client.c

4. Run the server using the below command.

   ./tcp_server

5. The server will start, waiting for a client to connect. On a separate terminal window, run the client using

   ./tcp_client <IP address of your neighbour>

6. Enter a character as input and it will be echoed back by the server. Use netstat to check the TCP connection state in client PC and server PC separately.
   a. What is the connection state in the client machine?
   b. What is the connection state in the server machine?

7. Stop client and server programs. Immediately, start the server again.
   a. What is the observation?
8. Connection Establishment states (LISTEN, SYN_SENT, SYN_RCVD)
   a. LISTEN state

      Start the server in your neighbor PC.

      i. What is the TCP connection state (observed using netstat)?

```
project2@project2-OptiPlex-380:~/Documents/sps2014/trainings$ cd
project2@project2-OptiPlex-380:~$ sudo netstat -actn |grep 7777
[sudo] password for project2:
tcp        0      0 0.0.0.0:7777            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:7777            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:7777            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:7777            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:7777            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:7777            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:7777            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:7777            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:7777            0.0.0.0:*               LISTEN
```

Fig. 2: LISTEN state on Server side

9. Connection Termination states
   a. FIN_WAIT2 state and CLOSE_WAIT state

      Server or Client does close (CTRL+C)

      i. What is the connection state in the client machine?

ii. What is the connection state in the server machine?

```
tcp        0      0 172.16.90.4:7777        172.16.90.5:51458       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51457       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51458       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51457       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51458       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51457       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51458       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51457       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51458       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51457       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51458       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51457       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51458       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51457       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51458       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51457       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51458       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51457       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51458       FIN_WAIT2
tcp        0      0 172.16.90.4:7777        172.16.90.5:51457       FIN_WAIT2
```

Fig. 3: FIN_WAIT2 state on Server side (172.16.90.4)

```
tcp        1      0 172.16.90.5:51458       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51457       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51458       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51457       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51458       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51457       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51458       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51457       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51458       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51457       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51458       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51457       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51458       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51457       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51458       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51457       172.16.90.4:7777        CLOSE_WAIT
tcp        1      0 172.16.90.5:51458       172.16.90.4:7777        CLOSE_WAIT
```

Fig. 4: CLOSE_WAIT state on Client side (172.16.90.5)

a).TIME_WAIT

Connect to the Server with two (or more clients). Stop the server program (normal close). Server goes to FIN_WAIT2. Then, terminate your client programs.

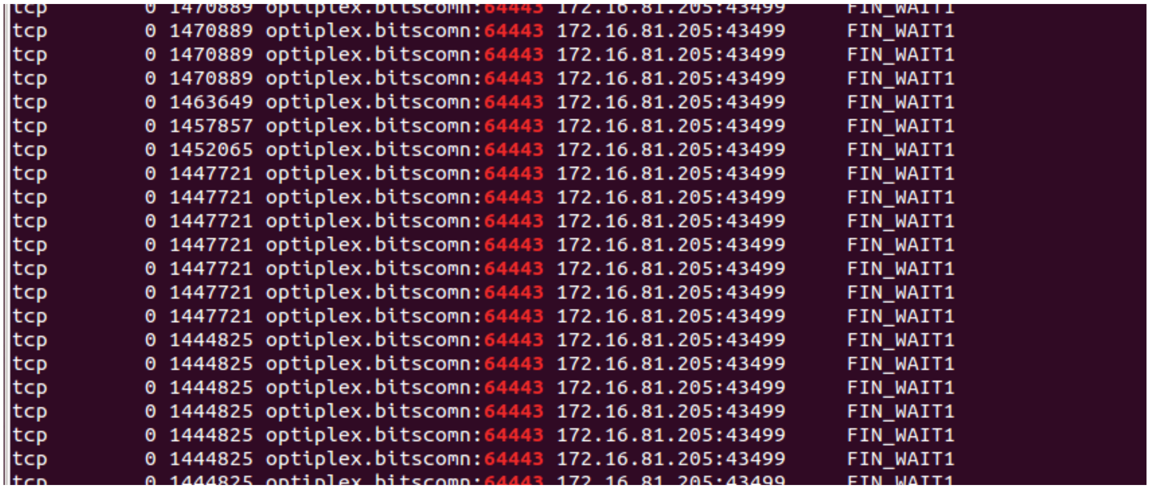
i. Now, what is the connection state in the Server machine?

```
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
tcp        0      0 172.16.90.5:47062       172.16.90.5:3030        TIME_WAIT
```

Fig. 5: TIME_WAIT state at Server

b. FIN_WAIT1

Terminate the server while the client is still receiving data from the server (HINT: sending a single character will not give such behavior. You will need to send lot more data). FIN_WAIT1 state will observed at server.



2. Fig. 6: FIN_WAIT1 at Server

3. Reset Connection

   Try sending data to Server when it has been terminated. RST packet is transmitted to client.

   a. Find the RST packet in Wireshark (Refer Figure 7 below)

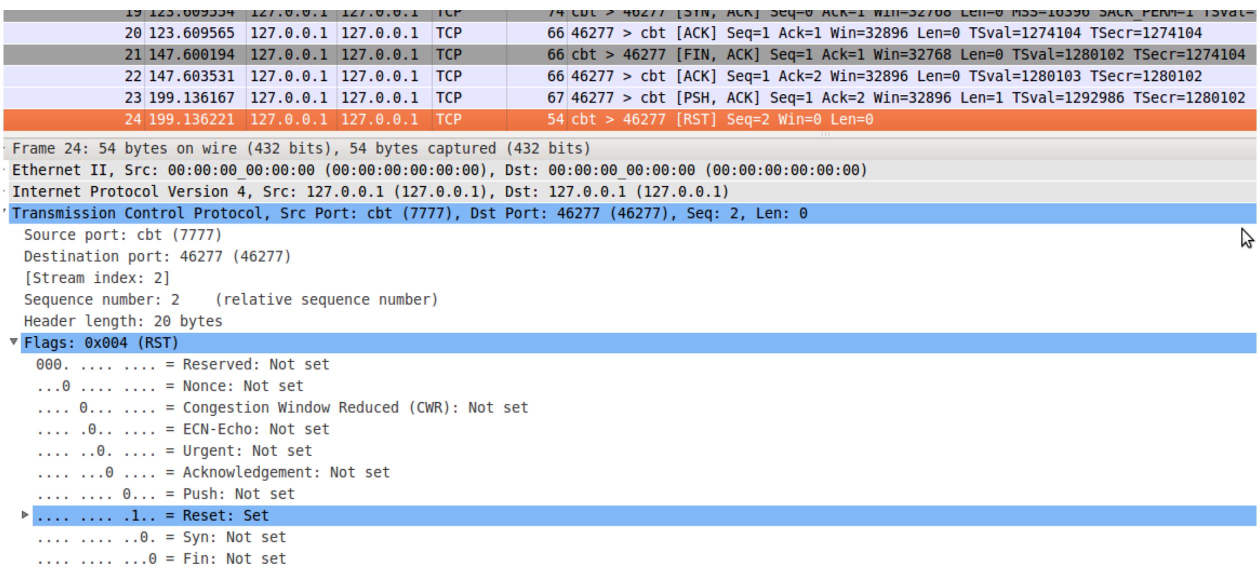


Fig. 7: Packet from Server (on port 7777) to client (on port 46277) with RST flag set

4. The Use of PUSH Flag

   a. Observe the data packets. Since, the amount of data is too low (1 byte) the TCP uses PUSH protocol to send the data. Figure 8 shows the PUSH flag being set in the TCP packet (TCP Flags)

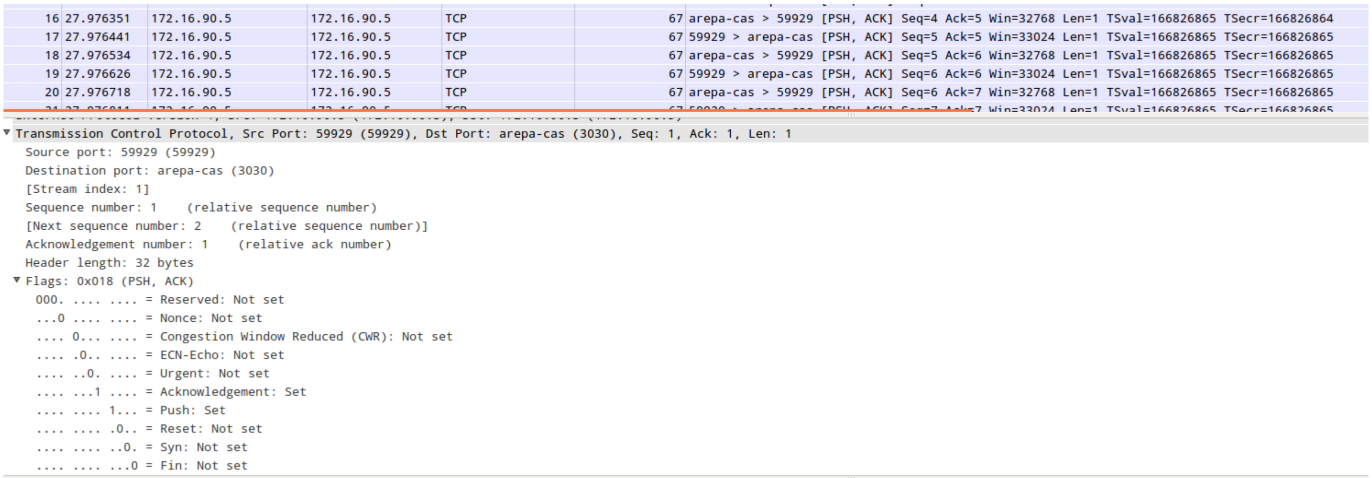


Fig. 8: Use of PSH to send data

TCP Header Fields:

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite (IP), and is so common that the entire suite is often called TCP/IP. TCP provides reliable, ordered, error-checked delivery of a stream of octets between programs running on computers connected to a local area network, intranet or the public Internet. It resides at the transport layer.

Tool used: Wireshark

Experiment 2: Observation of fields in a TCP packet header

1. Close all the browsers.
2. Run Wireshark in non-promiscuous mode with root privileges.
3. Open the website http://lbrce.ac.in/ in a browser window.
4. Stop Wireshark and observe the packets.

   How to differentiate a control packet and a data packet?

5. Connection Establishment packets
   a. Find the SYN packet in the TCP flow. The below diagram (Figure 9, see packet number 6) shows the TCP SYN flag set as in a Wireshark window.

      Who sends the SYN packet?

What is the HLEN value for the SYN message?

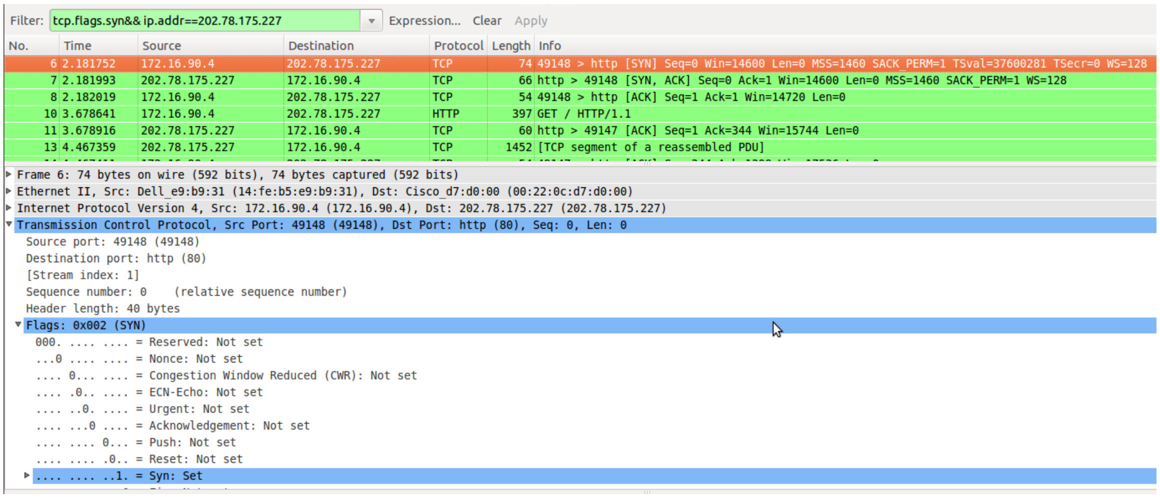


Fig. 9: SYN/ SYN-ACK /ACK packets

a. Find the SYN-ACK packet in the TCP flow. The above diagram (Figure 9, see packet number 7) shows the SYN-ACK packet.

   Who sends the SYN-ACK packet?

Find the first ACK packet from the server (the website). The below diagram (Figure 10) shows the TCP flags as in a Wireshark window.

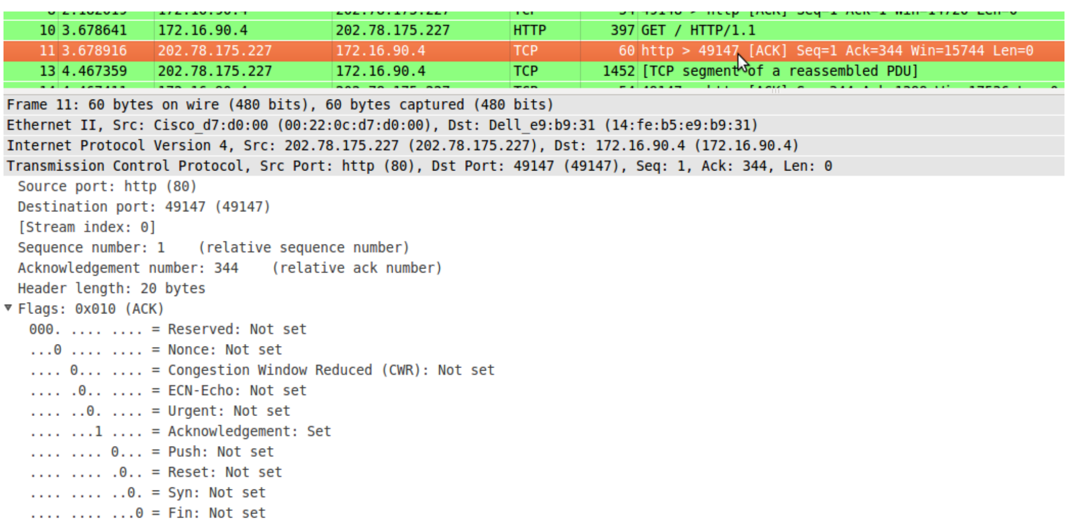


Fig. 10: Data ACKed by Server

a. Check if the last ACK in connection establishment (SYN-SYN ACK-ACK) is piggy backed with data packets?

6. Connection Termination packets
   a. Find the FIN packet in the TCP flow.
   b. Find the ACK or FIN-ACK packet from the server (the website).
   c. The below diagram (Figure. 11) shows the TCP FIN flags as in a Wireshark window.

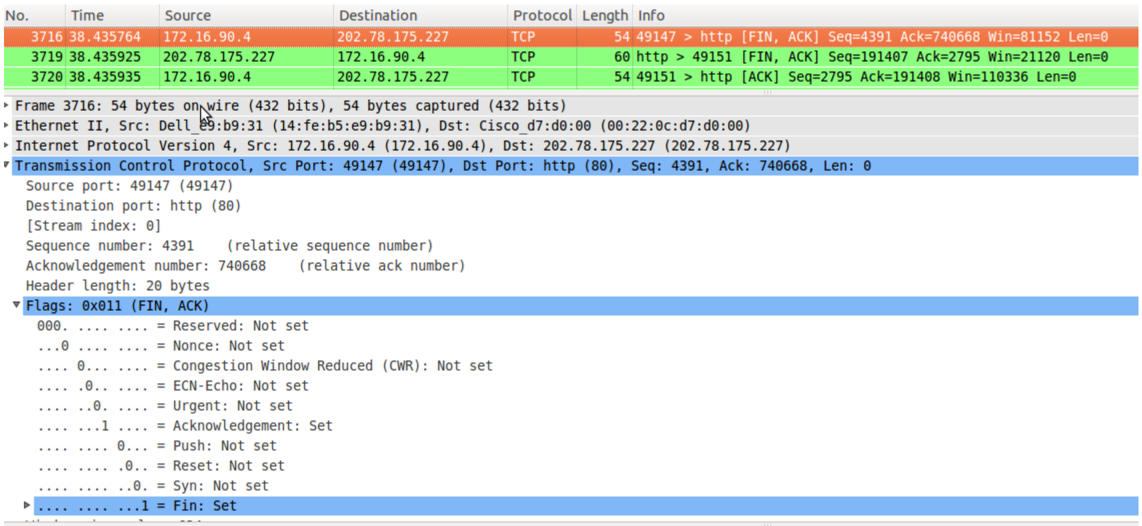


Fig. 11: Connection termination with FIN flags

1. Urgent flag
    a. In the Wireshark capture of your Client-Server module, find the packet(s) with Urgent flag set. (Figure 12)

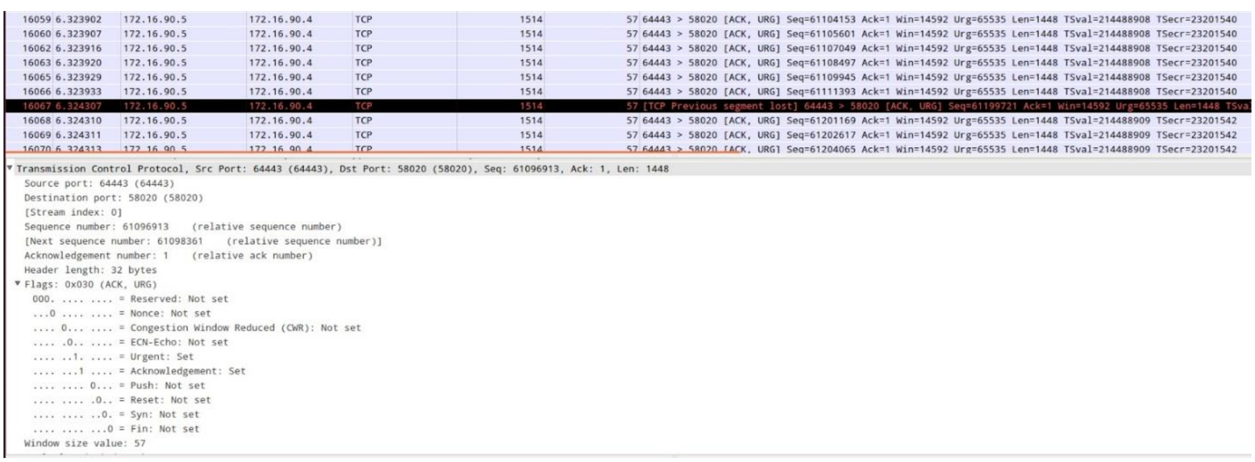


Fig. 12: URG flag set

2. Source port/Destination Port

The below figure (Figure. 13) shows the source port and destination port of a packet that travels from the web browser to the server.

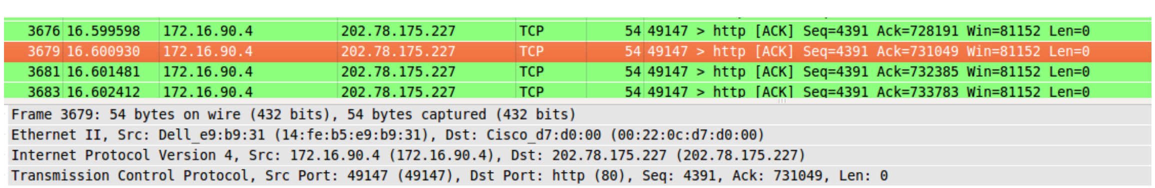


Fig. 13: Source port= 49147, Destination port= 80

a. Does the destination port number change for all packets from a client to a server?
b. For a given connection, will the source port number change for all packets from a client to a server?
c. What is the source and destination port number of a packet that traverses from the server to the client machine?

3. Sequence Number/Acknowledgement Number

Observe the sequence number and acknowledgement number in connection establishment packets.

a. Write down the sequence number and acknowledgement number for the following packets.

- SYN packet
- SYN-ACK packet
- ACK packet

b.What is the value of Window Size in each of the packets in question (a)

1. Round Trip Time Measurement

a. Go to the SYN packet in the TCP flow. Find the "timestamps" field in TCP Options (Refer Figure 14 below). The value of TSecr should be 0.

```
▼ Checksum: 0x0c59 [validation disabled]
    [Good Checksum: False]
    [Bad Checksum: False]
▼ Options: (20 bytes)
    Maximum segment size: 1460 bytes
    TCP SACK Permitted Option: True
  ▼ Timestamps: TSval 189415552, TSecr 0
      Kind: Timestamp (8)
      Length: 10
      Timestamp value: 189415552
      Timestamp echo reply: 0
    No-Operation (NOP)
```

Fig. 14: TSecr value = 0

b.Go to the SYN-ACK packet in the TCP flow. The "timestamps" field should be similar to the below figure (Figure. 15). What is the value of TSecr?

```
  Maximum segment size: 1460 bytes
  TCP SACK Permitted Option: True
▼ Timestamps: TSval 38123463, TSecr 189415552
    Kind: Timestamp (8)
    Length: 10
    Timestamp value: 38123463
    Timestamp echo reply: 189415552
  No-Operation (NOP)
▼ Window scale: 7 (multiply by 128)
    Kind: Window Scale (3)
    Length: 3
    Shift count: 7
    [Multiplier: 128]
```

Fig. 15: TSecr value for SYN-ACK

The round trip time is calculated based on the receipt of SYN-ACK packet from the server. When the client receives the ACK packet, it subtracts the received TSecr from the current clock (OS Clock) to obtain the round trip clock difference.

Now, RTT = Round Trip Clock Difference * Clock Period

2. Flow Control

    a. In your Client-Server pcap, can you identify TCP out-of-order packets and TCP Dup ACKs?

11	0.013956	172.16.81.205	172.16.90.5	TCP	78	[TCP Dup ACK 10#1] 43499 > 64443 [ACK] Seq=1 Ack=4345 Win=37888 Len=0 TSva
12	0.013989	172.16.90.5	172.16.81.205	TCP	1514	[TCP Out-Of-Order] 64443 > 43499 [ACK] Seq=4345 Ack=1 Win=14592 Len=1448 T
13	0.013998	172.16.81.205	172.16.90.5	TCP	78	43499 > 64443 [ACK] Seq=1 Ack=5793 Win=40784 Len=0 TSval=1173637 TSecr=212
14	0.014011	172.16.90.5	172.16.81.205	TCP	1514	[TCP Out-Of-Order] 64443 > 43499 [ACK] Seq=5793 Ack=1 Win=14592 Len=1448 T
15	0.016163	172.16.81.205	172.16.90.5	TCP	78	43499 > 64443 [ACK] Seq=1 Ack=7241 Win=43680 Len=0 TSval=1173637 TSecr=212
16	0.016195	172.16.90.5	172.16.81.205	TCP	1514	64443 > 43499 [ACK] Seq=18929 Ack=1 Win=14592 Len=1448 TSval=212554321 TSe
17	0.025121	172.16.81.205	172.16.90.5	TCP	78	43499 > 64443 [ACK] Seq=1 Ack=8689 Win=46576 Len=0 TSval=1173638 TSecr=212
18	0.025154	172.16.90.5	172.16.81.205	TCP	1514	64443 > 43499 [ACK] Seq=20377 Ack=1 Win=14592 Len=1448 TSval=212554323 TSe
19	0.026084	172.16.81.205	172.16.90.5	TCP	66	43499 > 64443 [ACK] Seq=1 Ack=10241 Win=49472 Len=0 TSval=1173638 TSecr=21
20	0.026116	172.16.90.5	172.16.81.205	TCP	1514	64443 > 43499 [ACK] Seq=21825 Ack=1 Win=14592 Len=1448 TSval=212554324 TSe
21	0.027797	172.16.81.205	172.16.90.5	TCP	66	43499 > 64443 [ACK] Seq=1 Ack=11689 Win=52368 Len=0 TSval=1173638 TSecr=21
22	0.027824	172.16.90.5	172.16.81.205	TCP	1514	64443 > 43499 [ACK] Seq=23273 Ack=1 Win=14592 Len=1448 TSval=212554324 TSe
23	0.028983	172.16.81.205	172.16.90.5	TCP	66	43499 > 64443 [ACK] Seq=1 Ack=13137 Win=55264 Len=0 TSval=1173639 TSecr=21
24	0.029010	172.16.90.5	172.16.81.205	TCP	1514	64443 > 43499 [ACK] Seq=24721 Ack=1 Win=14592 Len=1448 TSval=212554324 TSe
25	0.040483	172.16.81.205	172.16.90.5	TCP	66	43499 > 64443 [ACK] Seq=1 Ack=14585 Win=58160 Len=0 TSval=1173639 TSecr=21
26	0.040514	172.16.90.5	172.16.81.205	TCP	1514	64443 > 43499 [ACK] Seq=26169 Ack=1 Win=14592 Len=1448 TSval=212554327 TSe
27	0.040554	172.16.81.205	172.16.90.5	TCP	66	43499 > 64443 [ACK] Seq=1 Ack=16033 Win=61056 Len=0 TSval=1173640 TSecr=21
28	0.040569	172.16.90.5	172.16.81.205	TCP	1514	64443 > 43499 [ACK] Seq=27617 Ack=1 Win=14592 Len=1448 TSval=212554327 TSe
29	0.040785	172.16.81.205	172.16.90.5	TCP	66	43499 > 64443 [ACK] Seq=1 Ack=17481 Win=63952 Len=0 TSval=1173640 TSecr=21
30	0.040816	172.16.90.5	172.16.81.205	TCP	1514	64443 > 43499 [ACK] Seq=29065 Ack=1 Win=14592 Len=1448 TSval=212554327 TSe
31	0.040880	172.16.81.205	172.16.90.5	TCP	66	43499 > 64443 [ACK] Seq=1 Ack=18929 Win=66848 Len=0 TSval=1173641 TSecr=21
32	0.040901	172.16.90.5	172.16.81.205	TCP	7306	64443 > 43499 [ACK] Seq=30513 Ack=1 Win=14592 Len=7240 TSval=212554327 TSe
33	0.041716	172.16.81.205	172.16.90.5	TCP	78	[TCP Dup ACK 31#1] 43499 > 64443 [ACK] Seq=1 Ack=18929 Win=66848 Len=0 TSv
34	0.042800	172.16.81.205	172.16.90.5	TCP	78	[TCP Dup ACK 31#2] 43499 > 64443 [ACK] Seq=1 Ack=18929 Win=66848 Len=0 TSv

Fig. 16: TCP DUP ACKs and TCP out of order packets

b. Window Size scaling

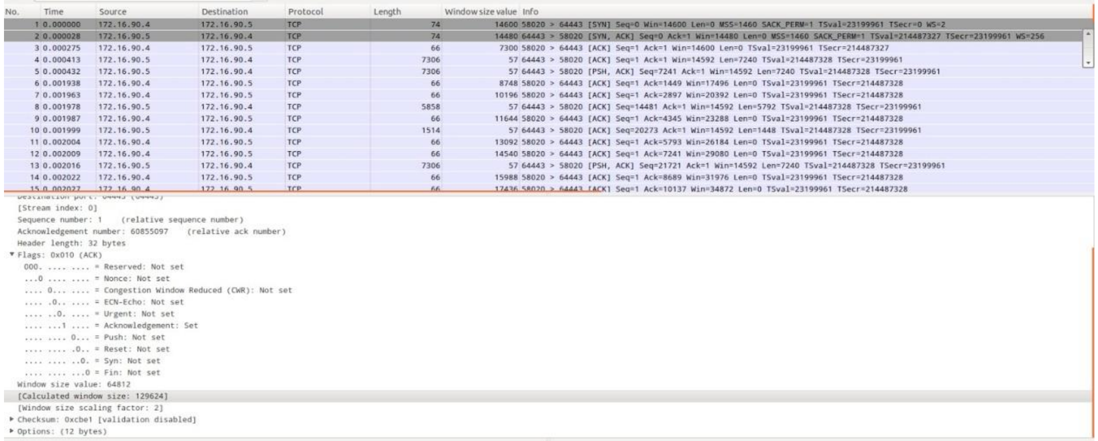


Fig. 17: Scaling window sizes

References

- ➢ Unix Manual: http://man7.org/linux/man-pages/man2/socket.2.html
- ➢ Wireshark User's Guide: www.wireshark.org/docs/wsug_html_chunked/
- ➢ Wireshark Wiki Help: wiki.wireshark.org/

Experiment-8

Aim: To learn Transmission Control Protocol (TCP) Flow Control, Error Control, and Congestion Control.

TCP Flow Control:

Automatic Repeat reQuest (ARQ), also known as Automatic Repeat Query, is an error-control method for data transmission that uses acknowledgements (messages sent by the receiver indicating that it has correctly received a data frame or packet) and timeouts (specified periods of time allowed to elapse before an acknowledgment is to be received) to achieve reliable data transmission over an unreliable service. If the sender does not receive an acknowledgment before the timeout, it usually re-transmits the frame/packet until the sender receives an acknowledgment or exceeds a predefined number of re-transmissions.

Selective Repeat is part of the automatic repeat-request (ARQ). With selective repeat, the sender sends a number of frames specified by a window size even without the need to wait for individual ACK from the receiver as in Go-back N ARQ. However, the receiver sends ACK for each frame individually, which is not like cumulative ACK as used with go-back-n. The receiver accepts out-of-order frames and buffers them. The sender individually retransmits frames that have timed out.

Tools used: Wireshark

Experiment 1: Demonstration of TCP Flow Control techniques

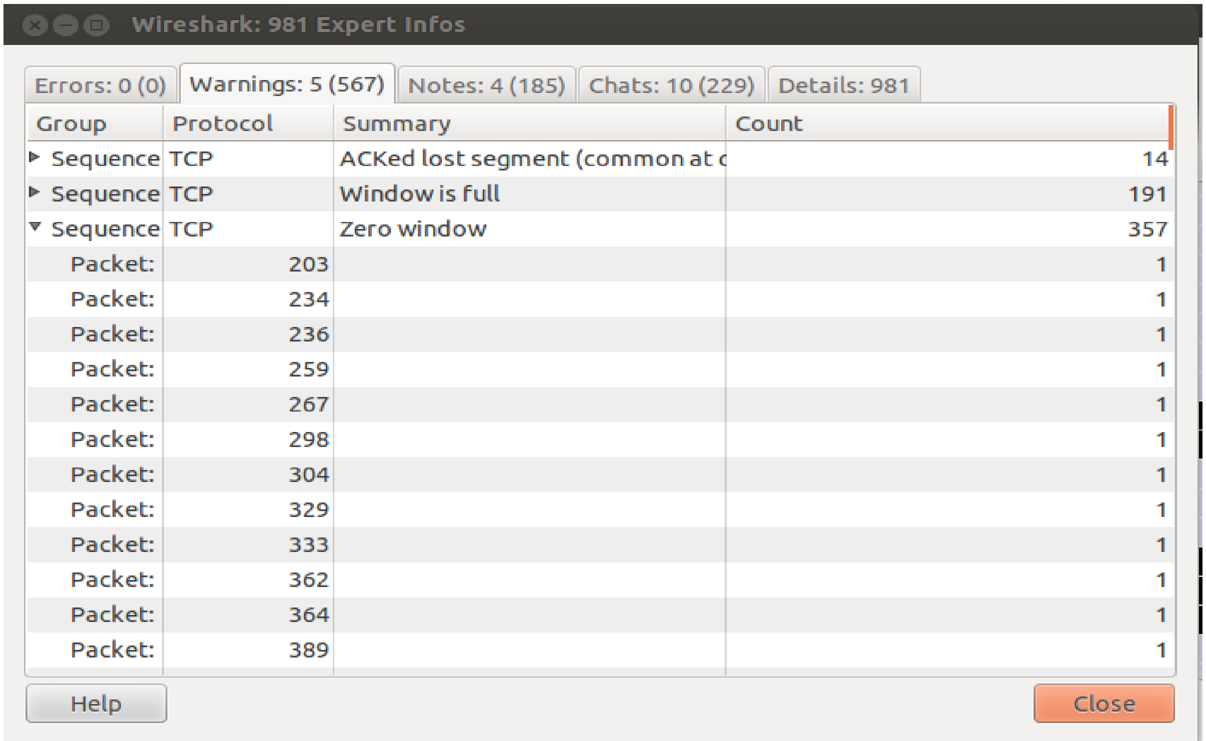
1. Open capture.pcap file in Wireshark.
2. Each and every ACK packet should have ACK flag bit set. Check if it is true for all the ACK packets.
3. A SACK reports a block of bytes that is out of order. Find a couple of packets with SACK option set? Write down the sequence number of the packets carrying SACK options?
4. Identifying zero window size packets (figure above)

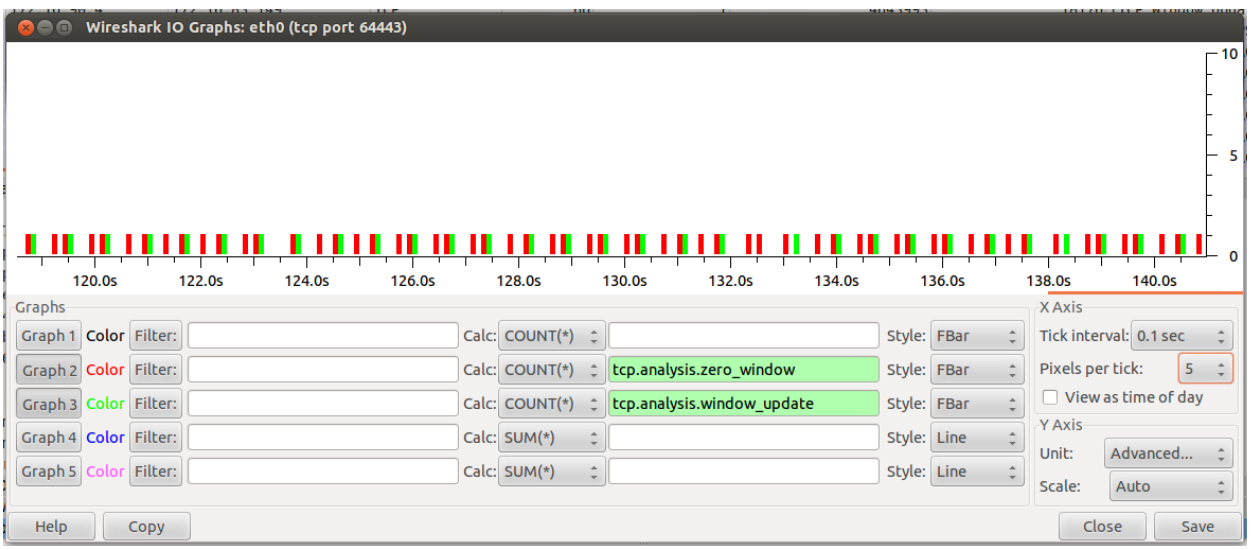
Go to Analyze->Expert Info composite. Under Warnings tab you should be able to see packet numbers which had zero Window.

No.	Time	Source	Destination	Protocol	Length	Sequence number	Acknowledgement number	Calculated window size	Info
6778	90.508766	172.16.83.149	172.16.90.4	TCP	1514	7588969	1	14592	64443 > 46155 [ACK] Se
6779	90.508815	172.16.90.4	172.16.83.149	TCP	78	1	7573041	42368	[TCP Dup ACK 6757#11]
6780	90.511452	172.16.83.149	172.16.90.4	TCP	1514	7590417	1	14592	64443 > 46155 [ACK] Se
6781	90.511477	172.16.90.4	172.16.83.149	TCP	78	1	7573041	42368	[TCP Dup ACK 6757#12]
6782	90.567280	172.16.83.149	172.16.90.4	TCP	1514	7565801	1	14592	[TCP Retransmission] 6
6783	90.567328	172.16.90.4	172.16.83.149	TCP	86	1	7573041	42368	[TCP Dup ACK 6757#13]

```
▸ Ethernet II, Src: Dell_e9:b9:31 (14:fe:b5:e9:b9:31), Dst: Cisco_d7:d0:00 (00:22:0c:d7:d0:00)
▸ Internet Protocol Version 4, Src: 172.16.90.4 (172.16.90.4), Dst: 172.16.83.149 (172.16.83.149)
▾ Transmission Control Protocol, Src Port: 46155 (46155), Dst Port: 64443 (64443), Seq: 1, Ack: 7573041, Len: 0
    Source port: 46155 (46155)
    Destination port: 64443 (64443)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    Acknowledgement number: 7573041    (relative ack number)
    Header length: 44 bytes
  ▸ Flags: 0x010 (ACK)
    Window size value: 331
    [Calculated window size: 42368]
    [Window size scaling factor: 128]
  ▸ Checksum: 0x05ed [validation disabled]
  ▾ Options: (24 bytes)
      No-Operation (NOP)
      No-Operation (NOP)
    ▸ Timestamps: TSval 148428888, TSecr 1491306
      No-Operation (NOP)
      No-Operation (NOP)
    ▾ SACK: 7574489-7591865
        left edge = 7574489 (relative)
        right edge = 7591865 (relative)
```

No.	Time	Source	Destination	Protocol	Info	Calculated window size
167	4.173004	172.16.90.4	172.16.83.149	TCP	46142 > 64443 [ACK] Seq=1 Ack=124529 Win=42368 Len=0 TSval=1	42368
170	4.177490	172.16.90.4	172.16.83.149	TCP	46142 > 64443 [ACK] Seq=1 Ack=127425 Win=42368 Len=0 TSval=1	42368
174	4.187524	172.16.90.4	172.16.83.149	TCP	46142 > 64443 [ACK] Seq=1 Ack=137561 Win=36096 Len=0 TSval=1	36096
182	4.216286	172.16.90.4	172.16.83.149	TCP	46142 > 64443 [ACK] Seq=1 Ack=167969 Win=15104 Len=0 TSval=1	15104
193	4.246365	172.16.90.4	172.16.83.149	TCP	46142 > 64443 [ACK] Seq=1 Ack=182449 Win=5248 Len=0 TSval=14	5248
197	4.276434	172.16.90.4	172.16.83.149	TCP	46142 > 64443 [ACK] Seq=1 Ack=186793 Win=2176 Len=0 TSval=14	2176
199	4.323487	172.16.90.4	172.16.83.149	TCP	46142 > 64443 [ACK] Seq=1 Ack=188241 Win=768 Len=0 TSval=147	768
203	4.731225	172.16.90.4	172.16.83.149	TCP	[TCP ZeroWindow] 46142 > 64443 [ACK] Seq=1 Ack=189009 Win=0	0
204	4.757390	172.16.90.4	172.16.83.149	TCP	[TCP Window Update] 46142 > 64443 [ACK] Seq=1 Ack=189009 Win	18176
215	4.787472	172.16.90.4	172.16.83.149	TCP	46142 > 64443 [ACK] Seq=1 Ack=202721 Win=8192 Len=0 TSval=14	8192
221	4.817538	172.16.90.4	172.16.83.149	TCP	46142 > 64443 [ACK] Seq=1 Ack=209961 Win=3072 Len=0 TSval=14	3072
224	4.847617	172.16.90.4	172.16.83.149	TCP	46142 > 64443 [ACK] Seq=1 Ack=212857 Win=256 Len=0 TSval=147	256
234	5.155277	172.16.90.4	172.16.83.149	TCP	[TCP ZeroWindow] 46142 > 64443 [ACK] Seq=1 Ack=213113 Win=0	0
236	5.415411	172.16.90.4	172.16.83.149	TCP	[TCP ZeroWindow] 46142 > 64443 [ACK] Seq=1 Ack=213113 Win=0	0
237	5.508925	172.16.90.4	172.16.83.149	TCP	[TCP Window Update] 46142 > 64443 [ACK] Seq=1 Ack=213113 Win	18176


Wireshark: 981 Expert Infos
Errors: 0 (0)
Warnings: 5 (567)
Notes: 4 (185)
Chats: 10 (229)
Details: 981
Group
Protocol
Summary
Count
Sequence TCP ACKed lost segment (common at c 14
Sequence TCP Window is full 191
Sequence TCP Zero window 357
Packet: 203 1
Packet: 234 1
Packet: 236 1
Packet: 259 1
Packet: 267 1
Packet: 298 1
Packet: 304 1
Packet: 329 1
Packet: 333 1
Packet: 362 1
Packet: 364 1
Packet: 389 1
Help
Close

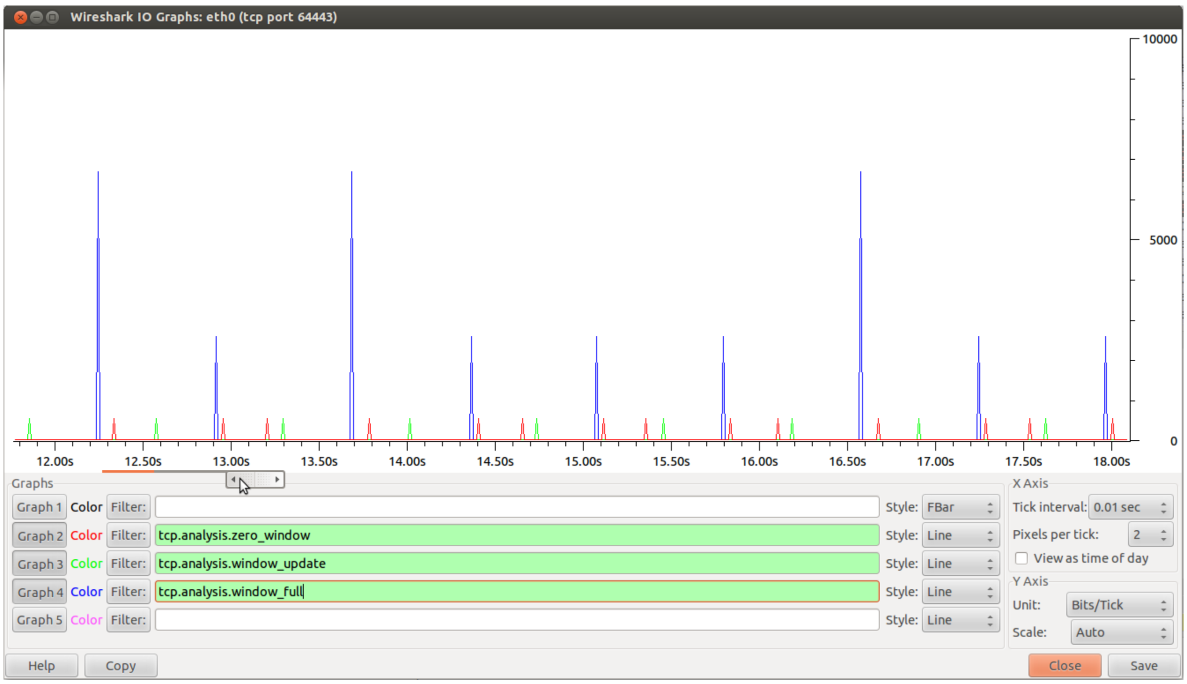

Wireshark IO Graphs: eth0 (tcp port 64443)
10
5
0
120.0s
122.0s
124.0s
126.0s
128.0s
130.0s
132.0s
134.0s
136.0s
138.0s
140.0s
Graphs
Graph 1 Color Filter: Calc: COUNT(*) Style: FBar
Graph 2 Color Filter: Calc: COUNT(*) tcp.analysis.zero_window Style: FBar
Graph 3 Color Filter: Calc: COUNT(*) tcp.analysis.window_update Style: FBar
Graph 4 Color Filter: Calc: SUM(*) Style: Line
Graph 5 Color Filter: Calc: SUM(*) Style: Line
X Axis
Tick interval: 0.1 sec
Pixels per tick: 5
View as time of day
Y Axis
Unit: Advanced...
Scale: Auto
Help
Copy
Close
Save

To identify zero window and window updates graphically:

Go to Statistics → IO graphs. For X axis, choose the Tick interval as 0.1 second and Pixels per tick as 5. For Y axis, choose Unit as Advanced. Plot graphs for tcp.analysis.zero_window and tcp.analysis.window_update as shown above. To generate the graph(s), click on the 'Graph *x*' (x=1,2,...) button at the left.

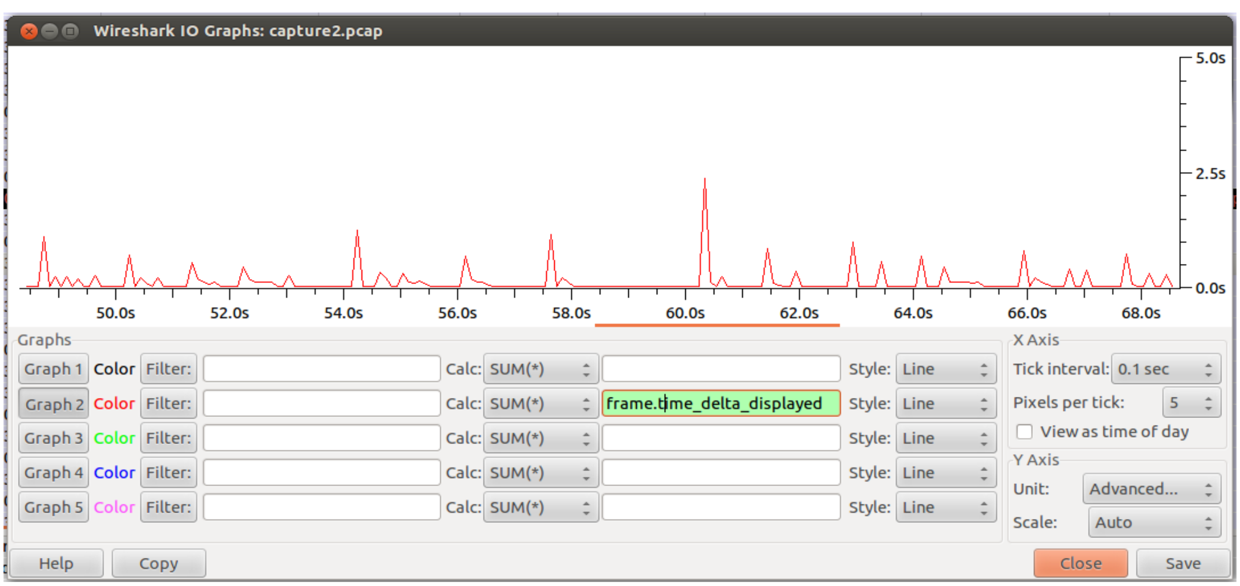
Similarly, you can generate a graph to observe TCP full window with zero window and window update.

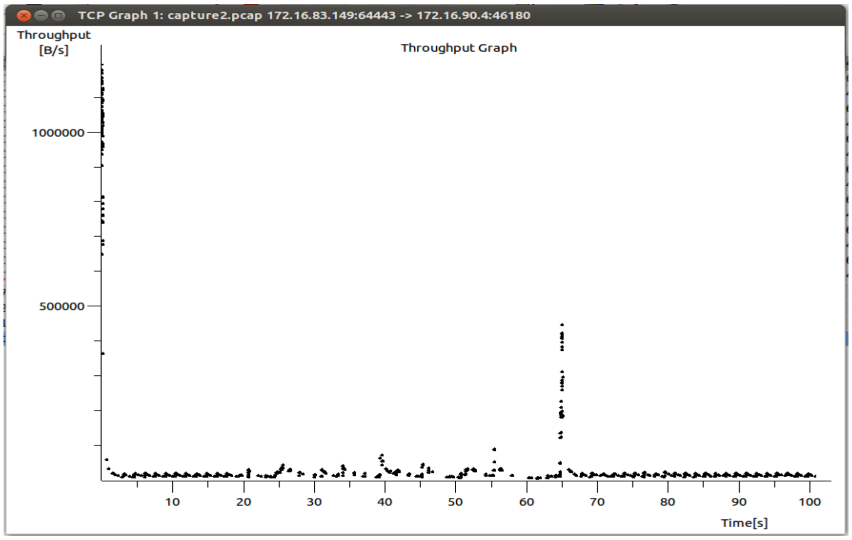
Modify arguments as shown below:



5. Response time IO Graphs (Inter-arrival time)

To observe time delay between packets, go to Statistics → IO graphs. For Y axis, choose the Unit as Advanced. Generate graph for time delay between packets as shown below. Large spike in the graph indicates large delays in time.



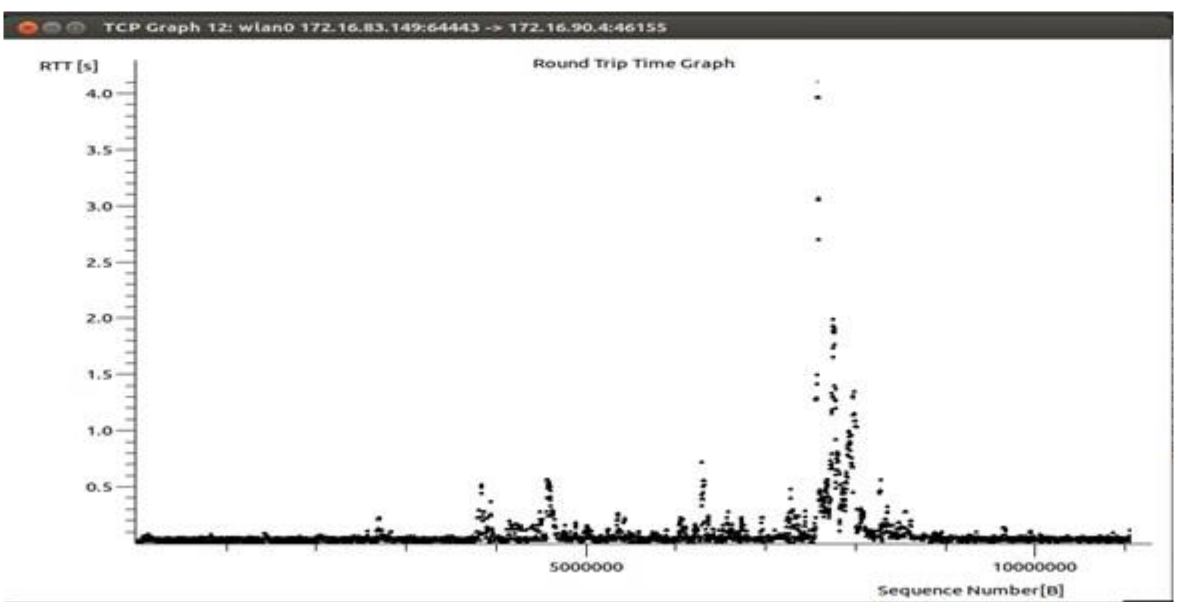


6. TCP Stream Graphs
    a. Round Trip Time Graph

Filter packets going from Server → Client:

Go to Statistics → TCP stream graphs →Round Trip Time graph

In the graph shown below, you can see a spike occurring for RTT. The explanation for this behavior is that the graph was obtained for a capture done with the Server on a laptop with a wi-fi connection. The Server was deliberately moved away from the wi-fi access point in a zone of weak network strength, which led to increased RTT values of the packets.



    a. Throughput Graph

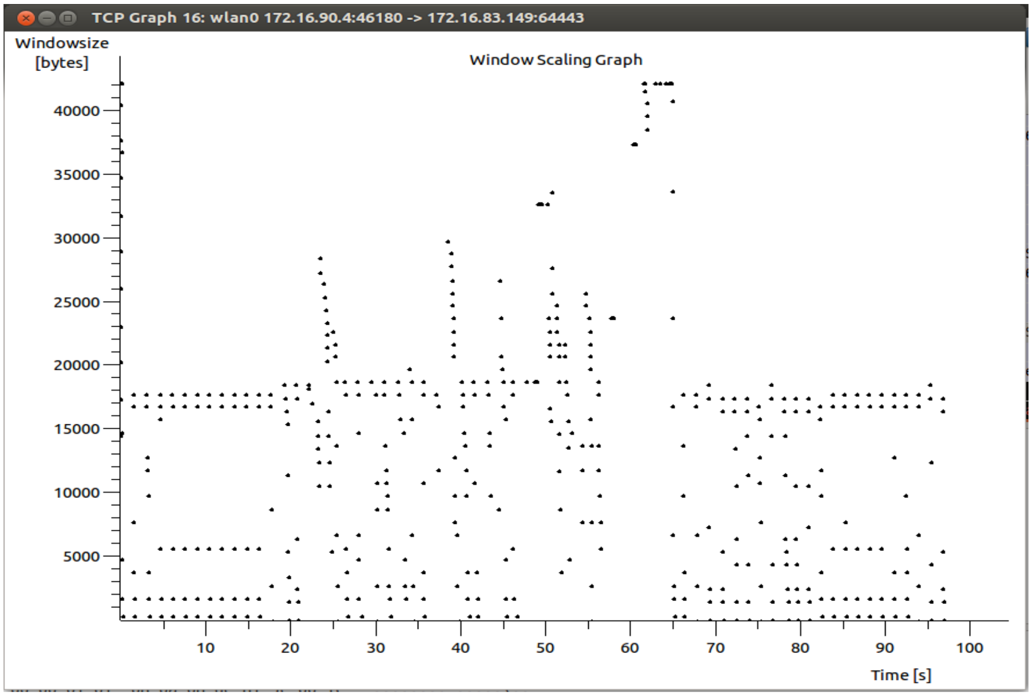
Filter packets going from Server-> Client:

Go to Statistics-> TCP stream graphs->Throughput graph

b. Window Scaling Graph

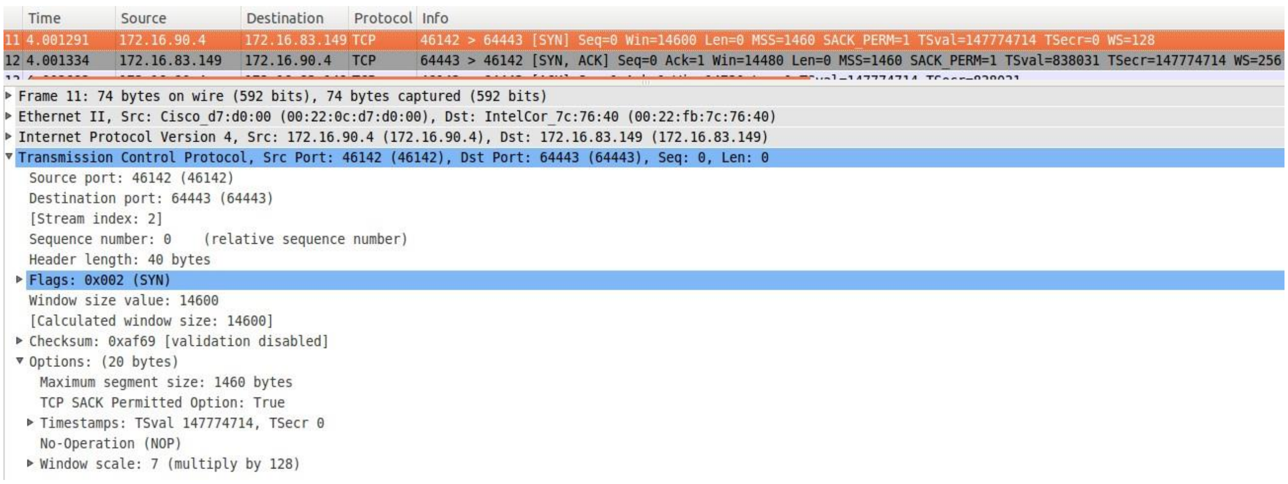
Filter packets going from Client->Server:

Go to Statistics-> TCP stream graphs->Window Scaling Graph



For better observations, you may zoom in the graph (Click the middle button on your mouse at the area which you want to zoom)

Window scale factors can be observed in the SYN packets sent from each side at the beginning of the TCP flow.



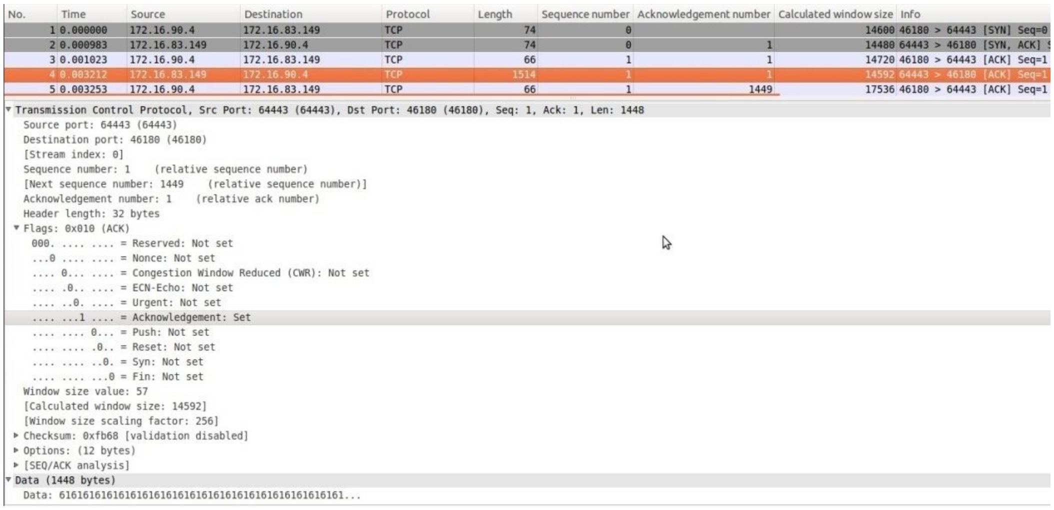
TCP Error Control:

There are six important rules that define the generation of an acknowledgement. The rules are given below.

Tool used: Wireshark

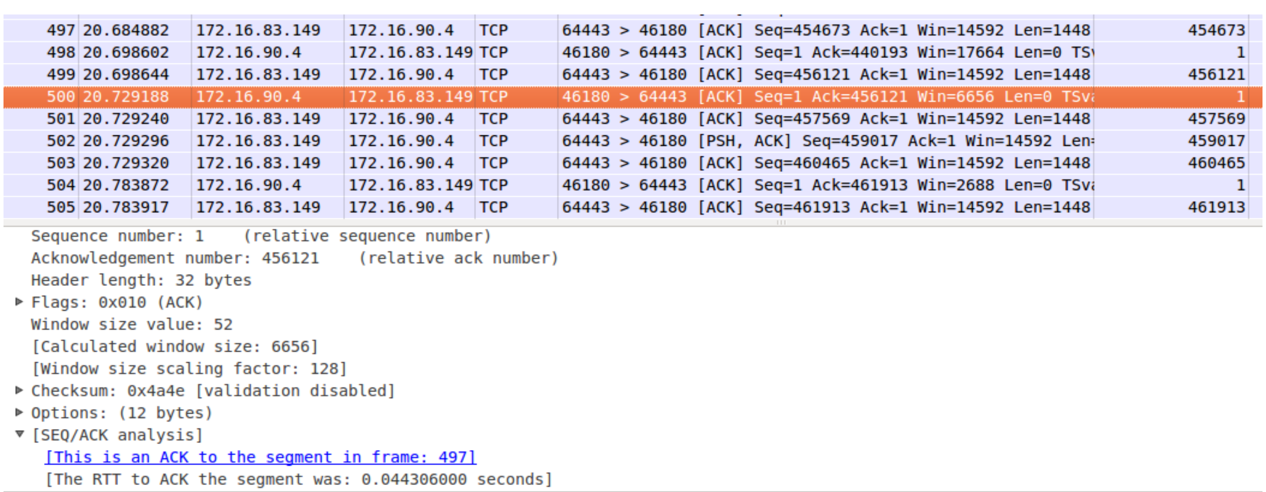
Experiment 2: Observation of Error Control in TCP protocol

a. Rule1: Normal TCP Operation
b. When end A sends a data segment to end B, it must include (piggyback) an acknowledgement that gives the next sequence number it expects to receive. This rule decreases the number of segments needed and therefore reduces traffic.
c. Can you see acknowledgement packets that are piggybacked?

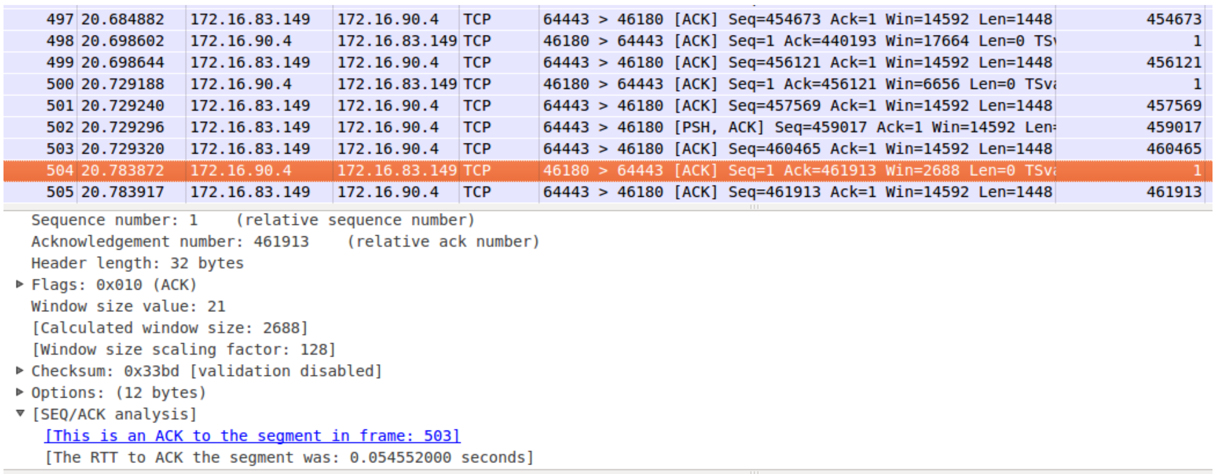


2. Rule2: Delayed ACK
   a. When the receiver has no data to send and it receives an in-order segment (with expected sequence number) and the previous segment has already been acknowledged, the receiver delays sending an ACK segment until another segment arrives or until a period of time (normally 500ms) has passed.
   b. We will observe this behavior along with rule 3.

3. Rule3: Preventing unnecessary retransmission of data segments.
   a. When a segment arrives with a sequence number that is expected by the receiver, and the previous in-order segment has not been acknowledged, the receiver immediately sends an ACK segment.
   b. Find the ACK packets in the pcap file which correspond to delayed ACK (rule 2) and rule 3. (See the two images below)



1. Rule4: Out-of-order sequence numbers

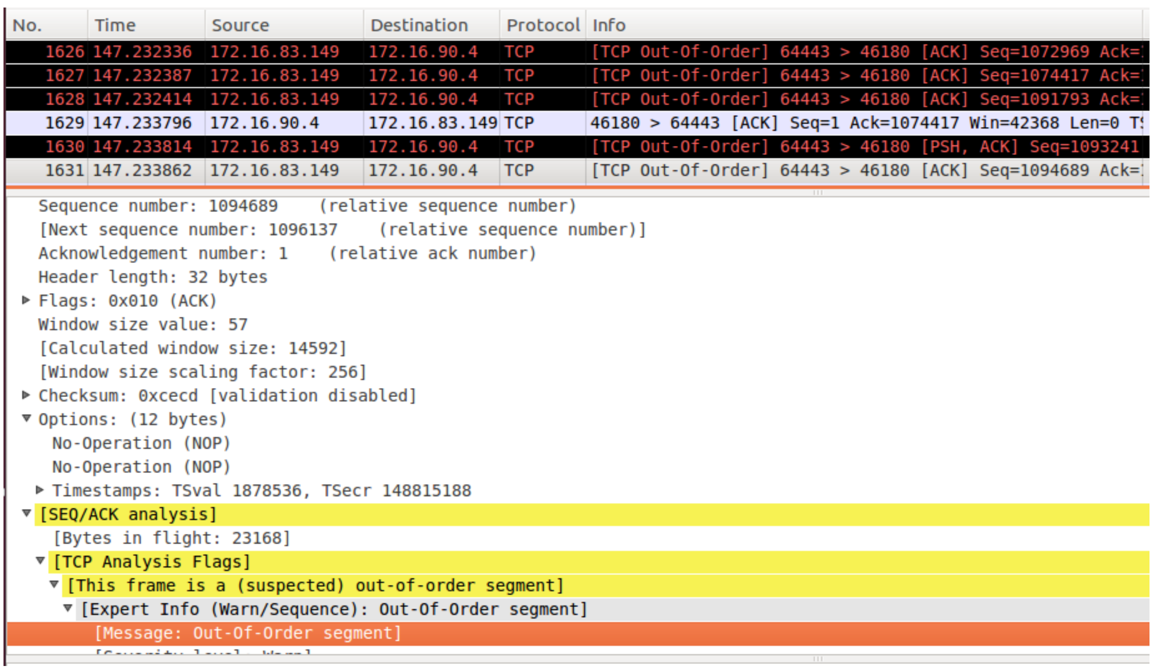


497	20.684882	172.16.83.149	172.16.90.4	TCP	64443 > 46180 [ACK] Seq=454673 Ack=1 Win=14592 Len=1448	454673
498	20.698602	172.16.90.4	172.16.83.149	TCP	46180 > 64443 [ACK] Seq=1 Ack=440193 Win=17664 Len=0 TS	1
499	20.698644	172.16.83.149	172.16.90.4	TCP	64443 > 46180 [ACK] Seq=456121 Ack=1 Win=14592 Len=1448	456121
500	20.729188	172.16.90.4	172.16.83.149	TCP	46180 > 64443 [ACK] Seq=1 Ack=456121 Win=6656 Len=0 TSv	1
501	20.729240	172.16.83.149	172.16.90.4	TCP	64443 > 46180 [ACK] Seq=457569 Ack=1 Win=14592 Len=1448	457569
502	20.729296	172.16.83.149	172.16.90.4	TCP	64443 > 46180 [PSH, ACK] Seq=459017 Ack=1 Win=14592 Len	459017
503	20.729320	172.16.83.149	172.16.90.4	TCP	64443 > 46180 [ACK] Seq=460465 Ack=1 Win=14592 Len=1448	460465
504	20.783872	172.16.90.4	172.16.83.149	TCP	46180 > 64443 [ACK] Seq=1 Ack=461913 Win=2688 Len=0 TSv	1
505	20.783917	172.16.83.149	172.16.90.4	TCP	64443 > 46180 [ACK] Seq=461913 Ack=1 Win=14592 Len=1448	461913

```
  Sequence number: 1    (relative sequence number)
  Acknowledgement number: 461913    (relative ack number)
  Header length: 32 bytes
▸ Flags: 0x010 (ACK)
  Window size value: 21
  [Calculated window size: 2688]
  [Window size scaling factor: 128]
▸ Checksum: 0x33bd [validation disabled]
▸ Options: (12 bytes)
▾ [SEQ/ACK analysis]
   [This is an ACK to the segment in frame: 503]
   [The RTT to ACK the segment was: 0.054552000 seconds]
```

a. When a segment arrives with an out-of-order sequence number that is higher than expected, the receiver immediately sends an ACK segment announcing the sequence number of the next expected segment.

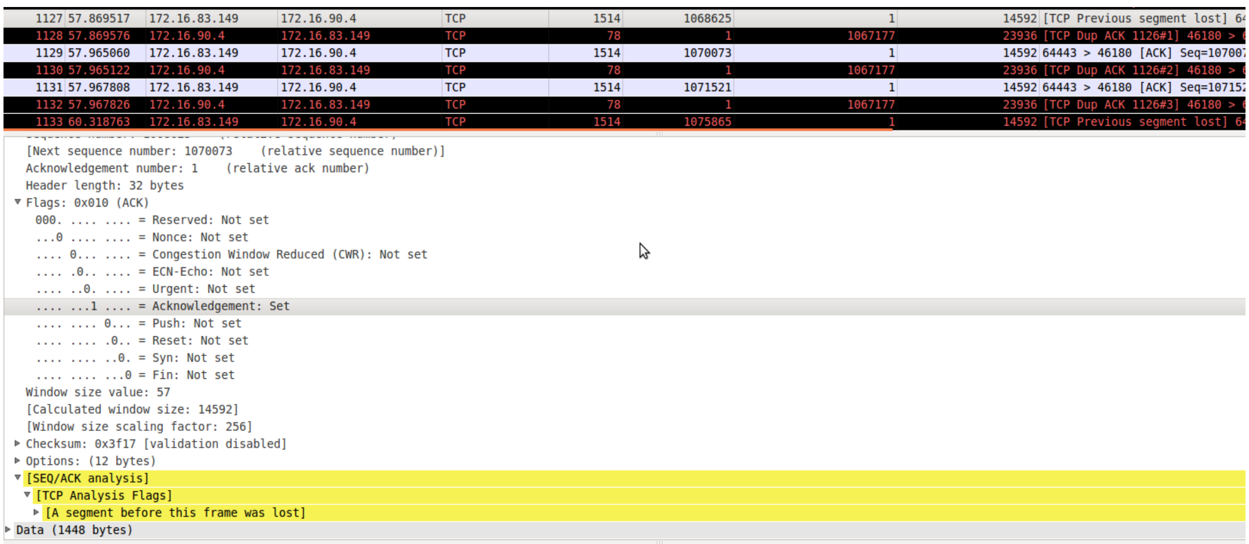
b. Find the out-of-order sequence numbered packets in the pcap file.



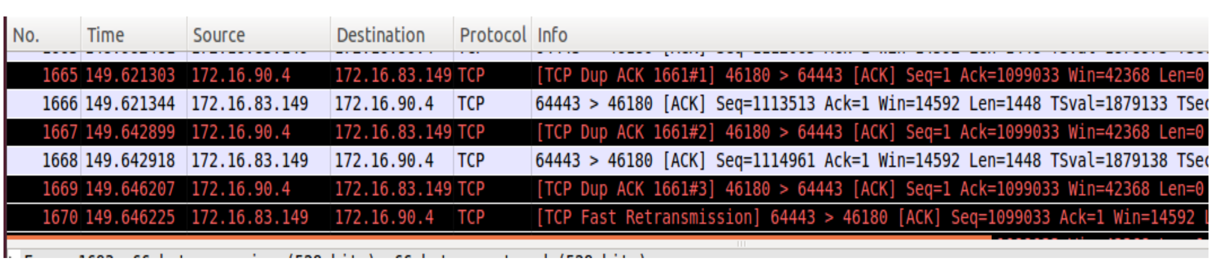
No.	Time	Source	Destination	Protocol	Info
1626	147.232336	172.16.83.149	172.16.90.4	TCP	[TCP Out-Of-Order] 64443 > 46180 [ACK] Seq=1072969 Ack=
1627	147.232387	172.16.83.149	172.16.90.4	TCP	[TCP Out-Of-Order] 64443 > 46180 [ACK] Seq=1074417 Ack=
1628	147.232414	172.16.83.149	172.16.90.4	TCP	[TCP Out-Of-Order] 64443 > 46180 [ACK] Seq=1091793 Ack=
1629	147.233796	172.16.90.4	172.16.83.149	TCP	46180 > 64443 [ACK] Seq=1 Ack=1074417 Win=42368 Len=0 T
1630	147.233814	172.16.83.149	172.16.90.4	TCP	[TCP Out-Of-Order] 64443 > 46180 [PSH, ACK] Seq=1093241
1631	147.233862	172.16.83.149	172.16.90.4	TCP	[TCP Out-Of-Order] 64443 > 46180 [ACK] Seq=1094689 Ack=

```
  Sequence number: 1094689    (relative sequence number)
  [Next sequence number: 1096137    (relative sequence number)]
  Acknowledgement number: 1    (relative ack number)
  Header length: 32 bytes
▸ Flags: 0x010 (ACK)
  Window size value: 57
  [Calculated window size: 14592]
  [Window size scaling factor: 256]
▸ Checksum: 0xcecd [validation disabled]
▾ Options: (12 bytes)
    No-Operation (NOP)
    No-Operation (NOP)
  ▸ Timestamps: TSval 1878536, TSecr 148815188
▾ [SEQ/ACK analysis]
    [Bytes in flight: 23168]
  ▾ [TCP Analysis Flags]
    ▾ [This frame is a (suspected) out-of-order segment]
      ▾ [Expert Info (Warn/Sequence): Out-Of-Order segment]
          [Message: Out-Of-Order segment]
```

2. Rule5: Missing segments
    a. When a missing segment arrives, the receiver sends an ACK segment to announce the next sequence number expected. This informs the receiver that segments reported missing has been received.
    b. Find the transmission of missing segments in the pcap file (see the **last packet** in the figure below).
    c. Can you see their corresponding ACKs in the pcap file?



1. Rule6: Duplicated segments
    a. If a duplicate segment arrives, the receiver discards the segment, but immediately sends an acknowledgment indicating the next in-order segment expected.
    b. Find the transmission of duplicated segments in the pcap file.
    c. Do you see TCP Fast Retransmit happening after Dup ACKs?



TCP Congestion Control:

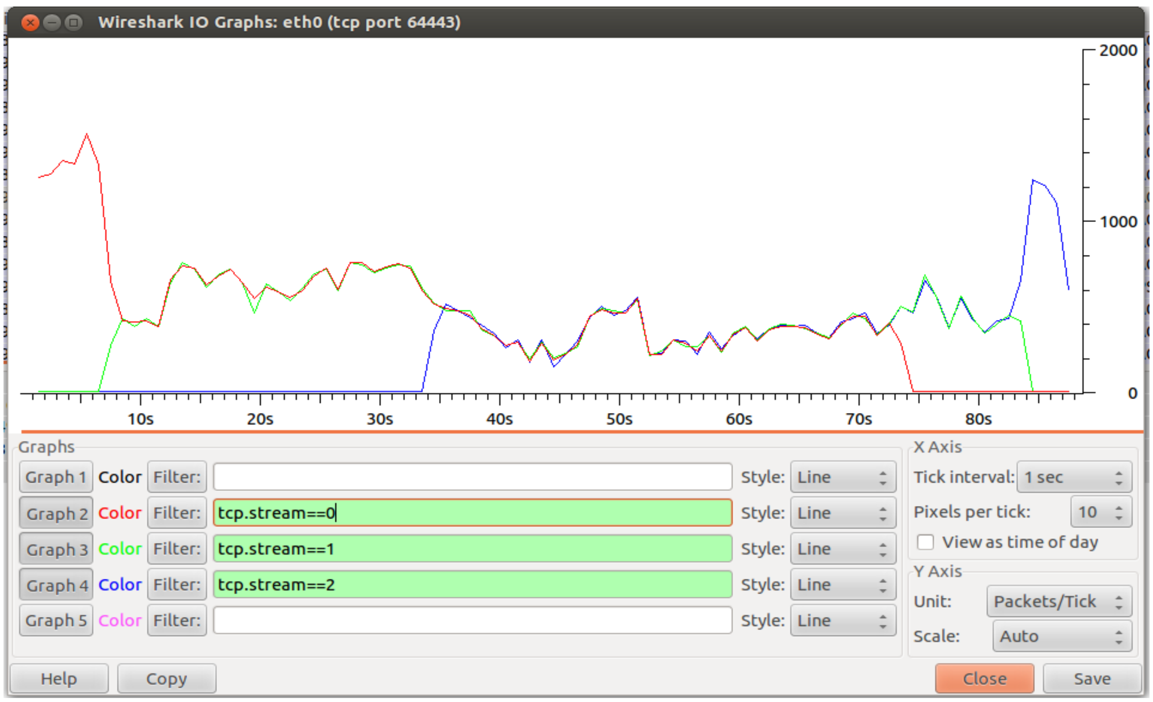
Congestion can occur when data arrives from a fast network to a slower network. Congestion can also occur when multiple input streams arrive at a router whose output capacity is less than the sum of the inputs. TCP is mainly used to avoid congestion in the network. To avoid the congestion, some of the packets are dropped in the network. This may lead to retransmission of data packets.

Tool used: wireshark, tracepath

Experiment 3: Demonstration of Congestion Control Techniques in TCP protocol

1. Congestion control

In our Client-Server module, congestion can be observed at the Server when multiple clients try to connect to it one after another. The figure below explains the fair share behavior of TCP. You can make these observations yourself by running a concurrent server and connecting multiple clients to it (with some small time difference- say a few seconds).



2. Find Path MTU using Tracepath tool

```
project2@project2-OptiPlex-380:~$ tracepath -n www.web.mit.edu
 1:  172.16.90.4                                          0.106ms pmtu 1500
 1:  172.16.90.1                                          0.731ms
 1:  172.16.90.1                                          0.688ms
 2:  172.16.0.30                                          0.619ms
 3:  111.93.6.69                                          2.784ms
 4:  115.113.207.153                                      2.604ms
 5:  172.31.16.193                                       14.087ms
 6:  121.240.1.202                                       49.918ms asymm  9
 7:  203.101.100.221                                     29.747ms asymm  9
 8:  125.22.194.10                                       36.582ms asymm 10
```

User-Datagram Protocol (UDP):

User-Datagram Protocol is a transport layer protocol used by many internet applications. In UDP, there is no handshaking between sending and receiving entities. For this reason, UDP is said to be connectionless.

Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets. In other words, all 16-bit words are summed using one's complement arithmetic. The sum is then one's complemented to yield the value of the UDP checksum field.

If the checksum calculation results in the value zero (all 16 bits 0) it should be sent as the one's complement (all 1s).

Tool used: Wireshark

Experiment 4: Observation of UDP Header fields

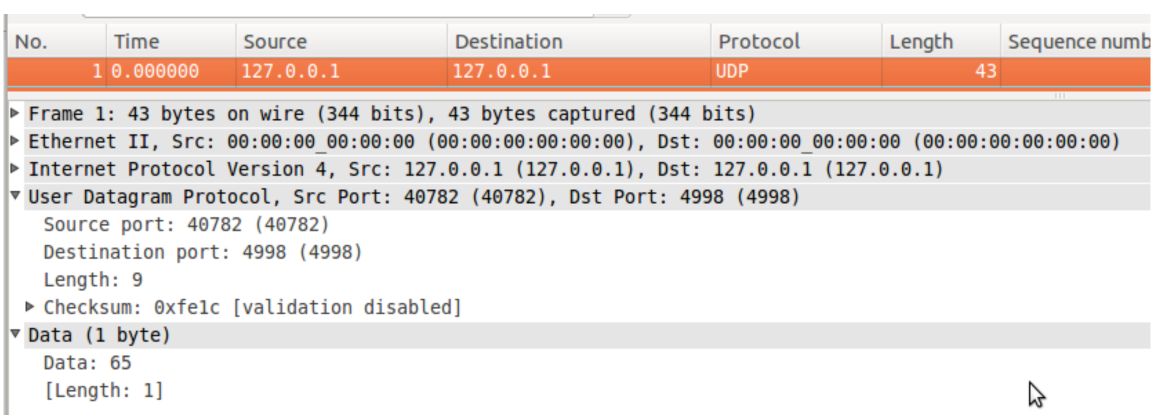
1. Close all the browsers.
2. Run Wireshark in non-promiscuous mode with root privileges.
3. Download "udp_client.c" and "udp_server.c" from the CMS Website
4. Compile server first (as shown below).

   gcc -o udp_server udp_server.c
5. Similarly, compile the client using the following

   command. gcc -o udp_client udp_client.c
6. Run the server using the below command.

   ./udp_server
7. The server will start, waiting for a client to connect. On a separate terminal window, run the client using

   ./udp_client 127.0.0.1
8. Payload size calculation
   a. Observe any UDP packet. The total length of UDP is given by the length of the



header fields (8 bytes) plus the length of the data (1 bytes in this case).

Answer the following questions based on your understanding of the above experiment.

1. What is the source port number?
2. What is the destination port number?

3. What is the total length of the user datagram?
4. What is the length of the data?

References

➢ Wireshark User's Guide: www.wireshark.org/docs/wsug_html_chunked/

Experiment-9

Aim: To give an Introduction to Wireshark &tcpdump, and observation of packets in a LAN network.

Packet Sniffer:

The basic tool for observing the messages exchanged between executing protocol entities is called a packet sniffer. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a copy of packets that are sent/received from/by application and protocols executing on your machine.

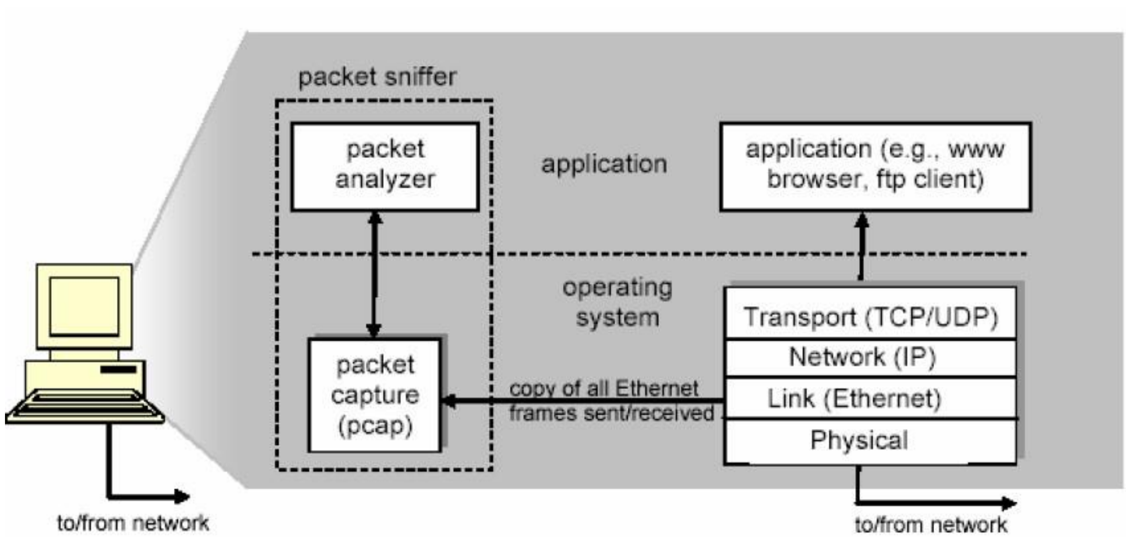


Figure1:PacketSnifferInternalStructure

We will be using the Wireshark packet sniffer [http://www.wireshark.org/] for these labs, allowing us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, Wireshark is a packet analyzer that uses a packet capture library in your computer). Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers. It's an ideal packet analyzer for our labs–it is stable, has a large user base and well-documented support includes a user guide.

(www.wireshark.org/docs/wsug_html_chunked/),manpages(www.wireshark.org/docs/man-pages/),anda detailed FAQ (www.wireshark.org/faq.html), rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface. It operates in computers using Ethernet, Token-Ring, FDDI, serial(PPPandSLIP),802.11wirelessLANs,andATMconnections(iftheOSonwhichit's running allows Wireshark to do so).

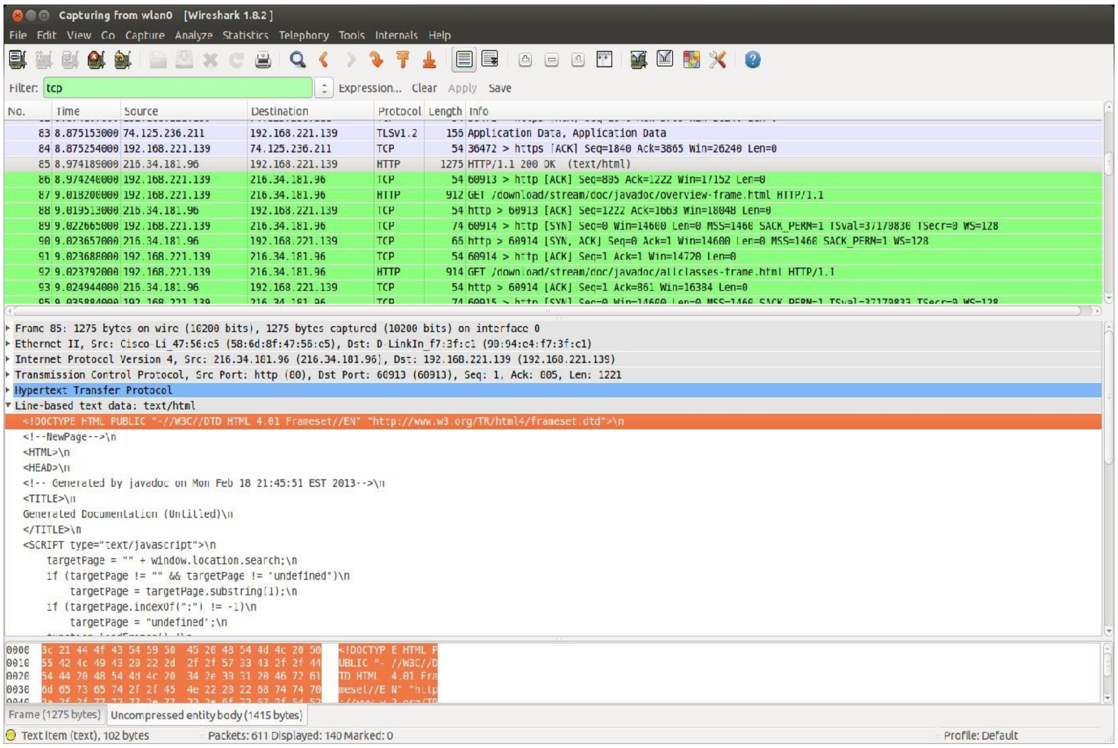


Figure2:Wireshark Snapshot Ubuntu

Experiment1: Introduction to Wireshark

1. Install Wireshark using the following command. If already installed, then please go to step.2.(One can also install from Ubuntu Software Center).

Sudo apt-get install wireshark

2. Oneneedsadministratorprivilegestoworkwith Wireshark.RunWiresharkwithsudo privileges (Type “sudo wireshark” in the Terminal).
3. Goto Capture->Optionsmenu.Check“eth0” interface and uncheck all other interfaces. Uncheck “Use promiscuous mode on all interfaces”.
4. Do packet capturing by clicking Capture->Start button. Now, the captured packets are shown in the center window. Stop capture (Capture->Stop button).
    - What is promiscuous mode of operation?
5. Filters–Therearedisplayfiltersandcapturefilters.Displayfilterscanbeusedonalready captured packets. Specify “tcp” in the display filter and press “Apply”.
    - What is the observation?
6. Capturefiltersisusedtofilteranynewincoming/outgoingpackets.Capturefilterscanbe specified in Capture->Options by typing in “Capture Filter” textbox.
7. Coloring rules–Depending on the protocol(IP,TCP,ARP,etc.)the color of the packet is different. These rules can be changed accordingly (View->Coloring Rules…).
8. Goto capture->interfaces. This will show all the interfaces available in the system.
    - How many interfaces does your system have?
    - Identify the IPaddress of “lo” interface.
9. Saving the output while capturing: After stopping the capture, do it from File->SaveAs.
    - Open try to open the pcap file in Wireshark.

Experiment 2: Introduction to tcpdump

Tcpdump is a common packet analyzer that runs under the command line. It allows the user to intercept and display TCP/IP and other packets being transmitted or received over a network to which the computer is attached. Distributed under theBSD license, tcpdump is free software. Tcpdump works on most Unix-like operating systems: Linux, Solaris, BSD, OS X, HP-UX and AIX among others. In those systems, tcpdump uses the libpcap library to capture packets. The port of tcpdump for Windows is called WinDump; it uses WinPcap, the Windows port of libpcap.

```
abhishek@atop9kx:~$ sudo tcpdump -i wlan0 -c 5
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type EN10MB (Ethernet), capture size 65535 bytes
12:39:12.986911 IP atop9kx.local.50452 > 224.0.0.142.4551: UDP, length 64
12:39:12.987015 IP6 fe80::9294:e4ff:fef7:3fc1.52803 > ff02::142.4551: UDP, lengt
h 64
12:39:12.987694 IP atop9kx.local.34535 > ns1.bits-hyderabad.ac.in.domain: 40492+
 PTR? 142.0.0.224.in-addr.arpa. (42)
12:39:13.007276 IP 192.168.221.123.47027 > 224.0.0.142.4551: UDP, length 61
12:39:13.008780 IP6 fe80::28e:f2ff:fe8a:e0d7.35347 > ff02::142.4551: UDP, length
 61
5 packets captured
101 packets received by filter
66 packets dropped by kernel
abhishek@atop9kx:~$
```

Figure3:tcpdump snapshot

If tcpdump is not already installed, run the below command to install it.

Sudo apt-get update
Sudo apt-get install tcpdump

1. Run tcpdump (with sudo privileges). Captured packets are displayed in each line (with minimal information).

2. Explore the various options in tcpdump
   - –i=>used to specify the interface to listen on(example:-ieth0)
   - –c => used to limit the total number of packets captured (example: -c 100 will capture100 packets and will stop)
   - –p=>run in non-promiscuous mode
   - –A=>displays the packets in ASCII format(-XXtodisplayinHEXformat)
   - –D=>lists only the interfaces
   - –w=>capture and write to a file(example:-wsample.pcap)
   - tcp=>capture only TCP packets

- port<num>=>capture from a specific port no.
- src<IP add#> => capture from specified source address. Try to differentiate sent and received packets.
- dst<IP add#> => capture from specified destination address. Try to differentiate sent and received packets.

Experiment3: Observation of packets in a LAN network

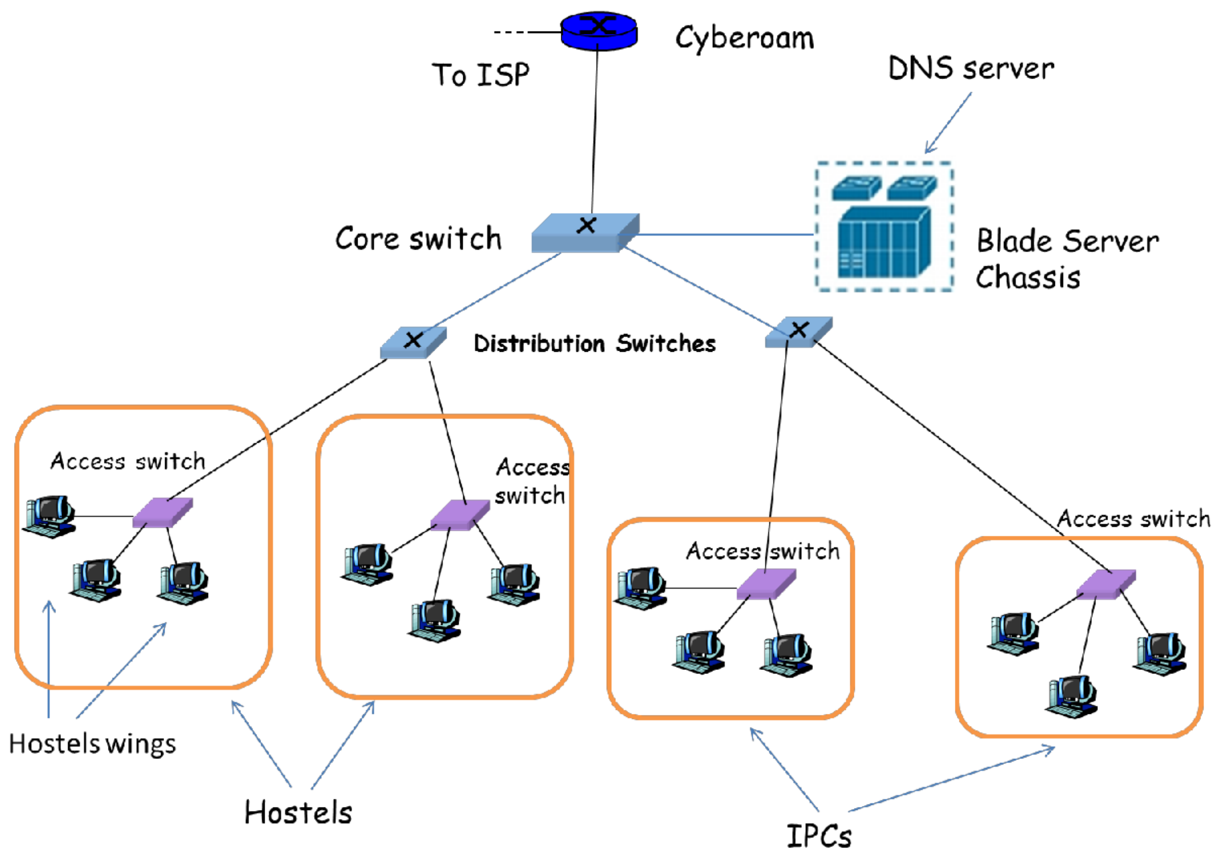
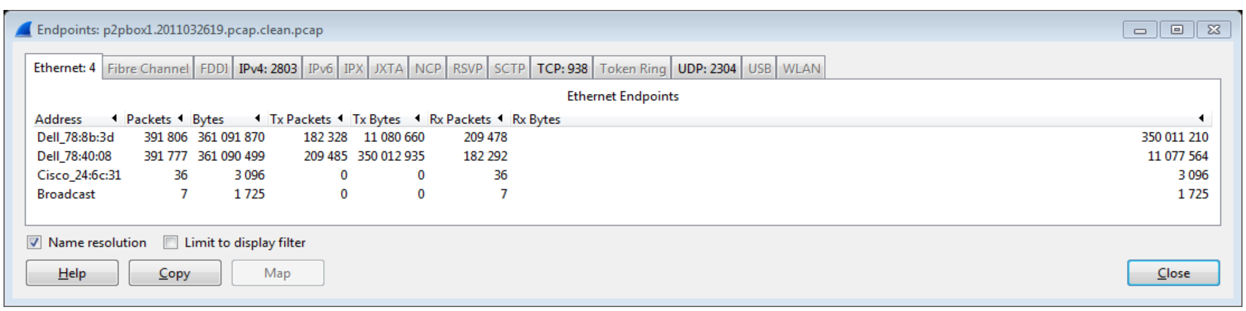


Figure4: A part of the BITS Hyderabad network

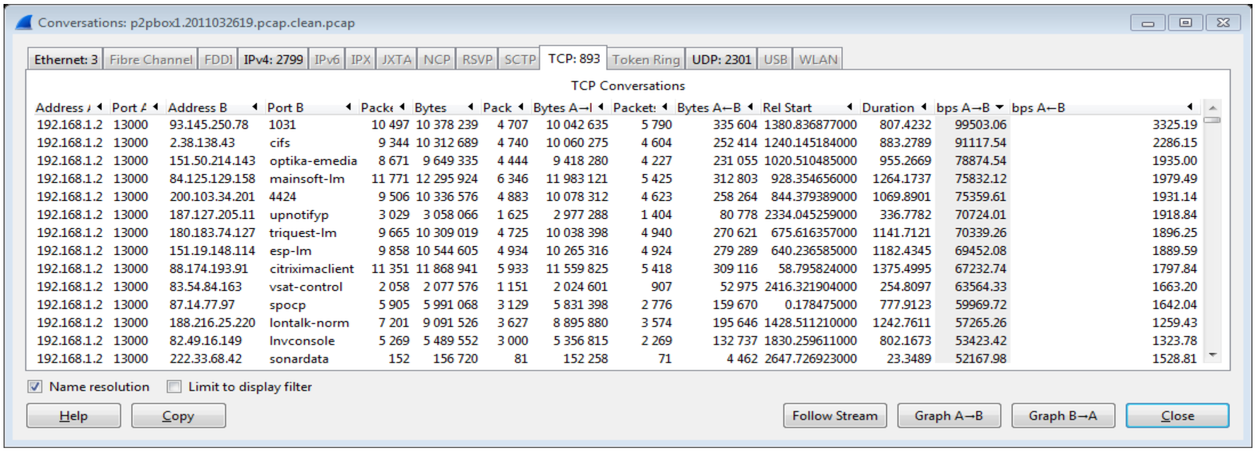
1. Close all browsers. Open a browser window and clear browser cache.
2. To view arp cache (On your Terminal: arp-a) (It displays MAC addresses, IP addresses, interface names). Depending on the subnet, the IP addresses of Cyberoam, default gateway, and DNS server will be shown.
3. Clear your system's ARP cache(On your Terminal: ipneigh flushall)
4. Put the **eth0** interface down using **ifconfig down**(as explained in previous lab session)
5. Launch tcpdump along with ifconfig up(with **sudo** privileges).
   - o sudo ifconfig eth0 up; sudo tcpdump -i eth0 c 1000 p w sample.pcap
   - o Open a couple of websites in your browser(ex.google.com, yahoo.com)
   - o Wait for tcpdump to stop.
6. Open sample.pcap file in Wireshark.
7. Observe on Wireshark how your system receives an IP address from the DHCP service. Identify the DHCP server's IP address.
8. ObservetheARPpacketsbeingsent.IdentifytheIPandMACaddressofyourdefaultgateway.
9. Observe the DNS query being made to resolve the IP address of the website you visited. Identify the DNS server which responded to you.
10. By observing the packets in Wireshark, identify your own IP address and the IP address of the website you visited.
11. Explore Statistics->Endpoints to identify entities involved in capture.
    - o Differentiate for the rnet,IP,TCP,UDP etc.,



12.ExploreStatistics->Conversations to cover flows(pair of endpoints)

- o Observer different tabs(Ethernet,IP,TCP,UDPetc.).
- o SortondifferentcolumnsinTCP–e.g.Duration,Packets,AddressA,RelStartetc.

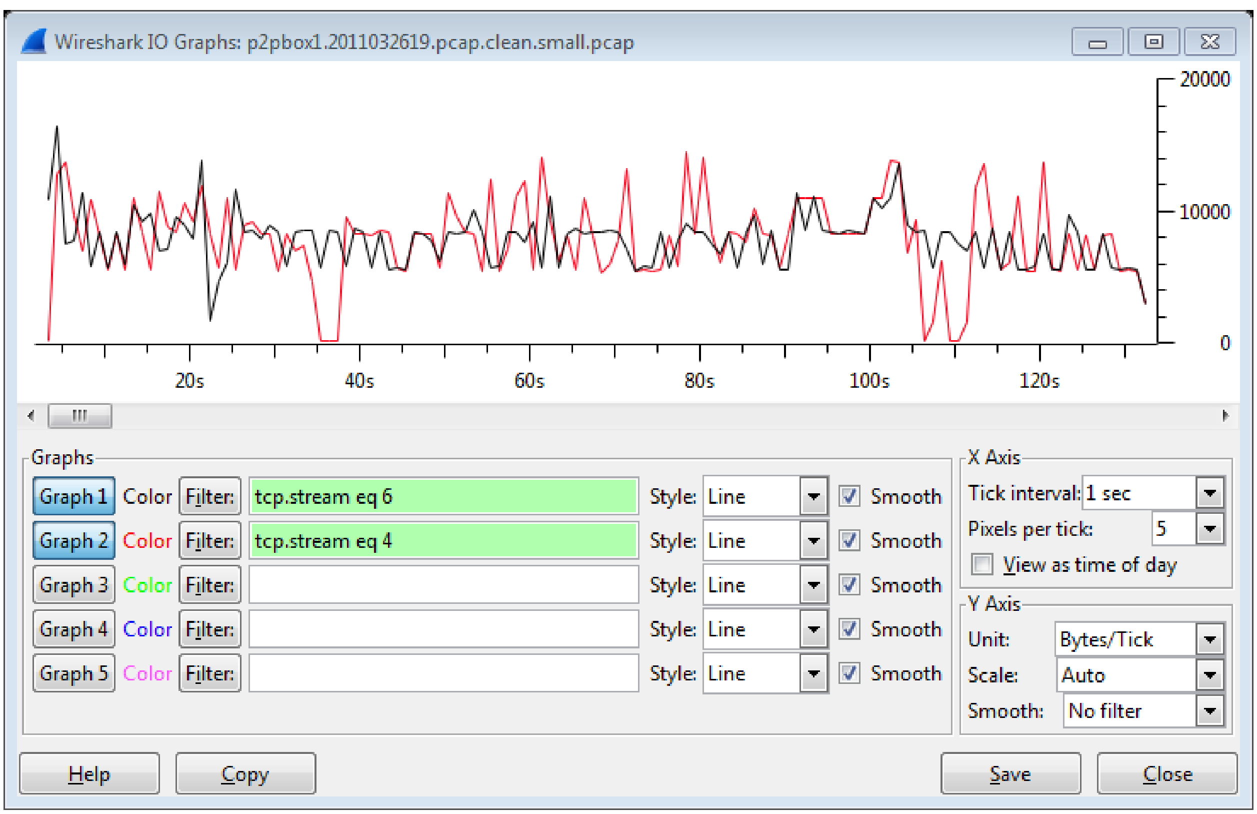
You may also experiment with "FollowStream" button on the popup dialog which adds a Display filter



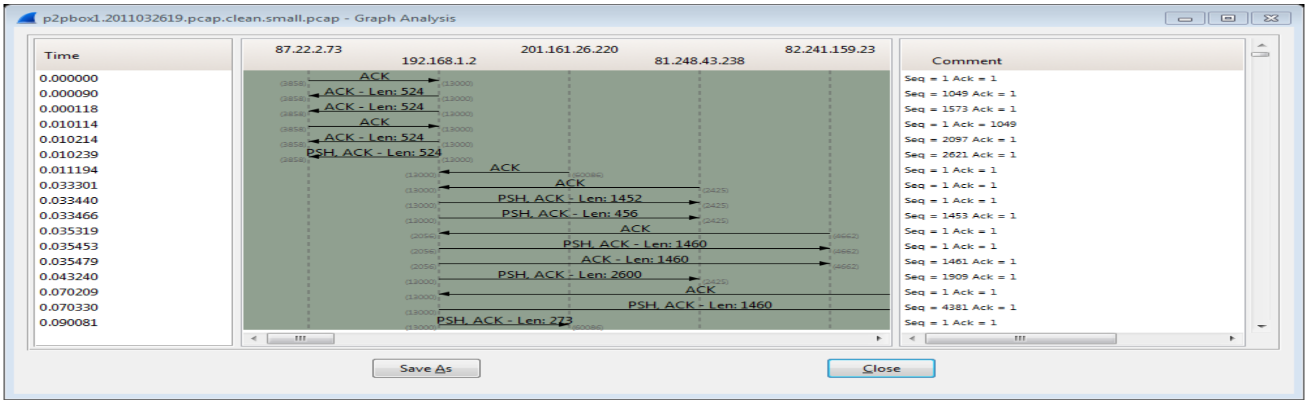
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	Bytes A←B	Rel Start	Duration	bps A→B	bps A←B
192.168.1.2	13000	93.145.250.78	1031	10 497	10 378 239	4 707	10 042 635	5 790	335 604	1380.836877000	807.4232	99503.06	3325.19
192.168.1.2	13000	2.38.138.43	cifs	9 344	10 312 689	4 740	10 060 275	4 604	252 414	1240.145184000	883.2789	91117.54	2286.15
192.168.1.2	13000	151.50.214.143	optika-emedia	8 671	9 649 335	4 444	9 418 280	4 227	231 055	1020.510485000	955.2669	78874.54	1935.00
192.168.1.2	13000	84.125.129.158	mainsoft-lm	11 771	12 295 924	6 346	11 983 121	5 425	312 803	928.354656000	1264.1737	75832.12	1979.49
192.168.1.2	13000	200.103.34.201	4424	9 506	10 336 576	4 883	10 078 312	4 623	258 264	844.379389000	1069.8901	75359.61	1931.14
192.168.1.2	13000	187.127.205.11	upnotifyp	3 029	3 058 066	1 625	2 977 288	1 404	80 778	2334.045259000	336.7782	70724.01	1918.84
192.168.1.2	13000	180.183.74.127	triquest-lm	9 665	10 309 019	4 725	10 038 398	4 940	270 621	675.616357000	1141.7121	70339.26	1896.25
192.168.1.2	13000	151.19.148.114	esp-lm	9 858	10 544 605	4 934	10 265 316	4 924	279 289	640.236585000	1182.4345	69452.08	1889.59
192.168.1.2	13000	88.174.193.91	citriximaclient	11 351	11 868 941	5 933	11 559 825	5 418	309 116	58.795824000	1375.4995	67232.74	1797.84
192.168.1.2	13000	83.54.84.163	vsat-control	2 058	2 077 576	1 151	2 024 601	907	52 975	2416.321904000	254.8097	63564.33	1663.20
192.168.1.2	13000	87.14.77.97	spocp	5 905	5 991 068	3 129	5 831 398	2 776	159 670	0.178475000	777.9123	59969.72	1642.04
192.168.1.2	13000	188.216.25.220	lontalk-norm	7 201	9 091 526	3 627	8 895 880	3 574	195 646	1428.511210000	1242.7611	57265.26	1259.43
192.168.1.2	13000	82.49.16.149	lnvconsole	5 269	5 489 552	3 000	5 356 815	2 269	132 737	1830.259611000	802.1673	53423.42	1323.78
192.168.1.2	13000	222.33.68.42	sonardata	152	156 720	81	152 258	71	4 462	2647.726923000	23.3489	52167.98	1528.81

13.Explore Statistics->IOGraph for complete communication, and after filtering for TCP communication.

- o Compare two TCP flows–e.g.stream6 and 4 below.
- o Observe the time slider below the graph.



14.ExploreStatistics->FlowGraphtounderstandsequenceofeventsforthefilteredcapture.



References

- ❖ WiresharkUser'sGuide:www.wireshark.org/docs/wsug_html_chunked/
- ❖ WiresharkWikiHelp:wiki.wireshark.org/
- ❖ Tcpdumpdocumentation:www.tcpdump.org/#documentation

Experimet-10

Aim: To analyze HTTP packets using Wireshark tool, and understand the records returned by a DNS server.

HTTP (Hypertext Transfer Protocol):

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. It is the foundation of data communication for the World Wide Web. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

HTTP functions as a request-response protocol in the client-server computing model. A web browser, for example, may be the client and an application running on a computer hosting a web site may be the server. The client submits an HTTP request message to the server. The server returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

Tools used: Wireshark

Experiment 1: Working of HTTP

1. Run Wireshark with sudo privileges. Start capturing in non-promiscuous mode.
2. Start up your web browser. Next, enter into your browser http://timesofindia.indiatimes.com.
3. Wait until the page is fully loaded. Now, stop the capture. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running? (Refer Figure.1.)

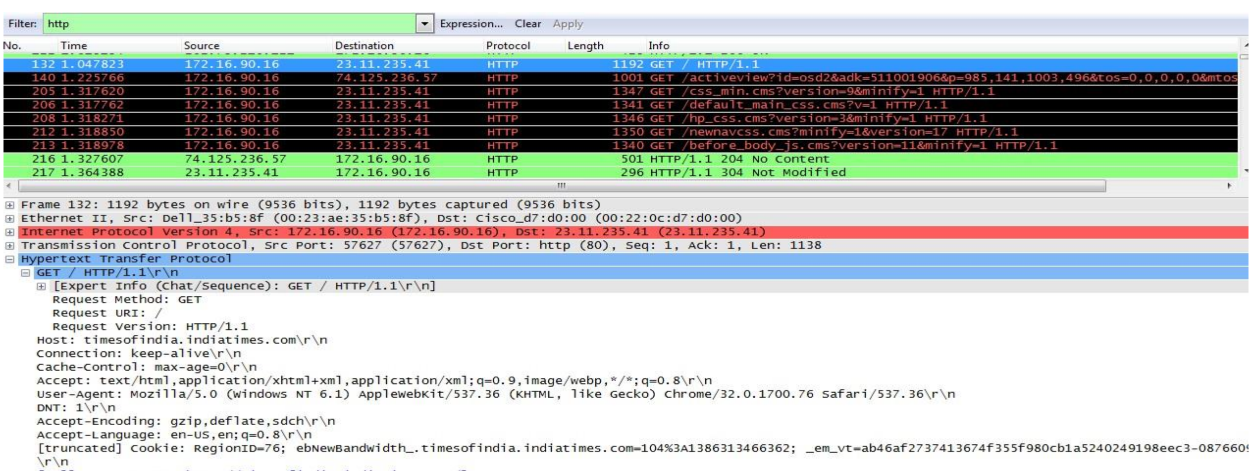


Figure.1. HTTP Request and Responses

4. Type "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window. A couple of the responses would have a response code of 304 (Not modified). What does the "Last-Modified" field imply? (Refer Figure.2.)

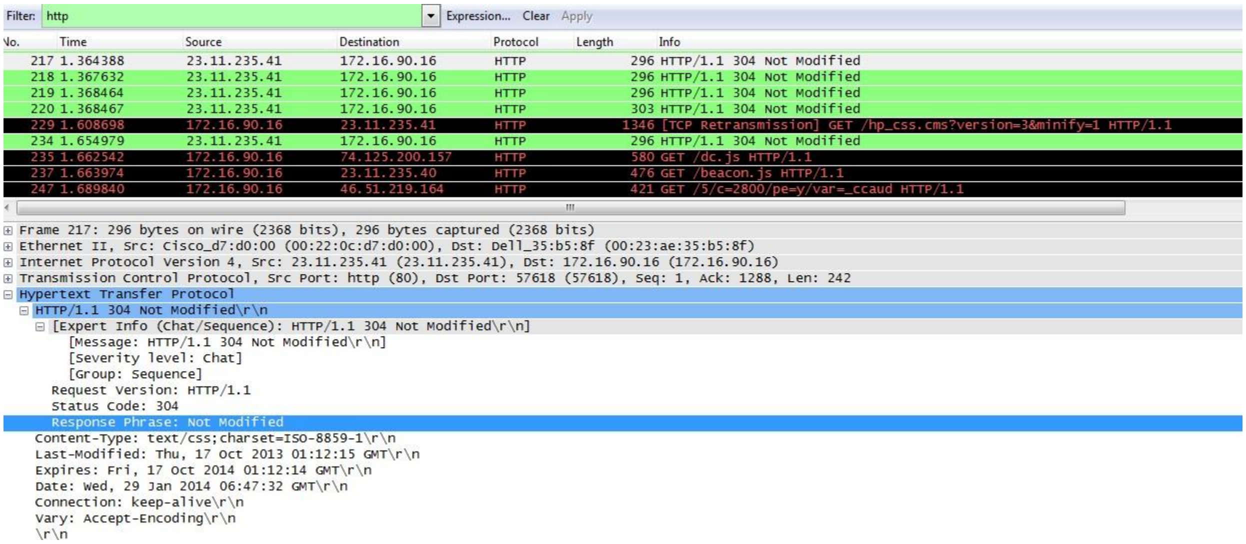


Figure.2. HTTP Not Modified page

5. Type "http.response.code==200" in the display-filter-specification window. Response code of 200 implies that HTTP request got processed successfully and HTTP response is sent to the browser window. You can explore different response codes like 304 (not modified), 404 (not found) etc. (Refer Figure.3.)

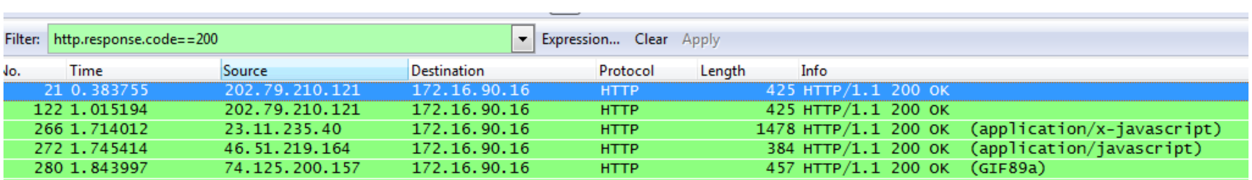


Figure.3. HTTP Response Code

6. Type "http.cookie" in the display-filter-specification window.
A cookie is a small piece of data sent from a website and stored in a user's web browser while the user is browsing that website. Every time the user loads the website, the browser sends the cookie back to the server to notify the website of the user's previous activity. (Refer Figure.4.)

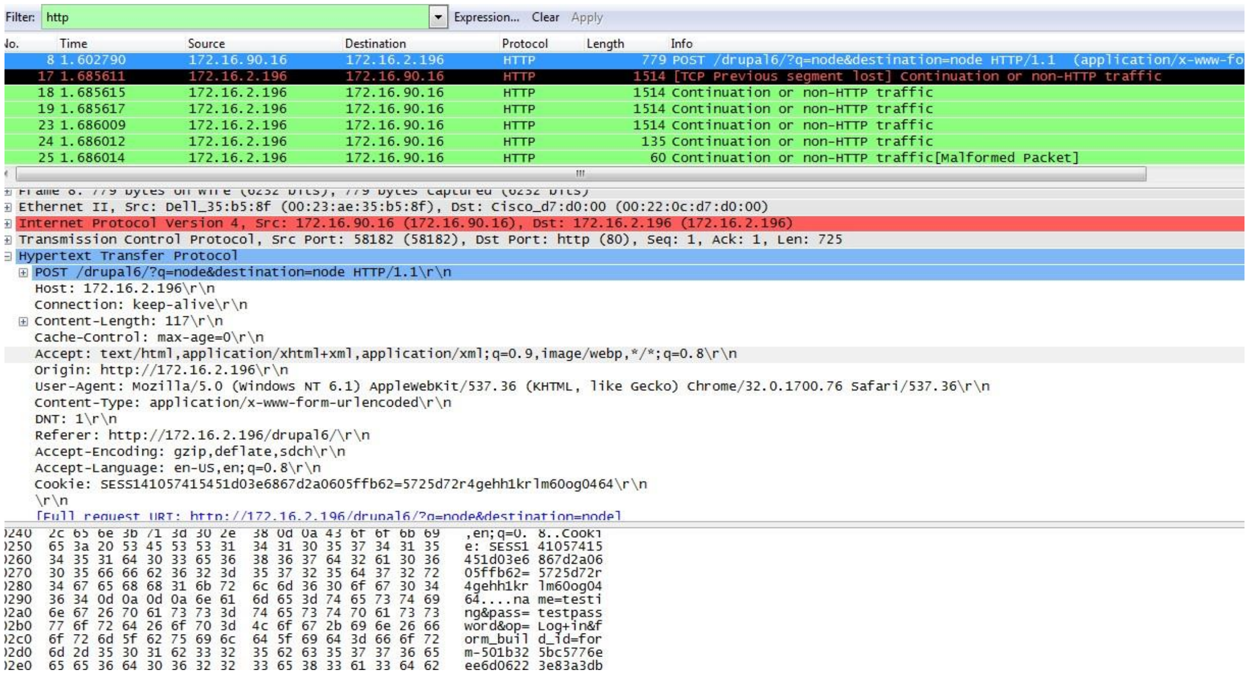
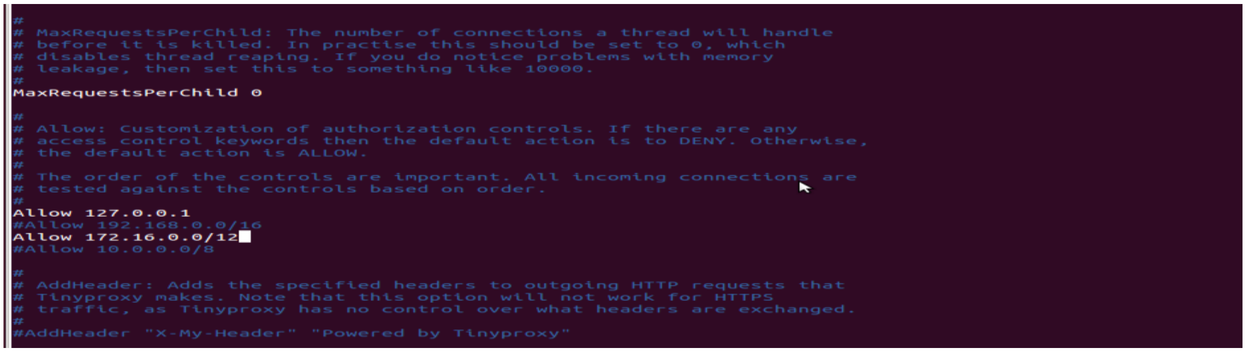


Figure.4. HTTP Cookie

7. Persistent v/s Non-Persistent HTTP connections. Do you see Connection: keep-alive set in your packet captures?

8. HTTP proxy set-up (Work in a team of two. First member will install proxy server and other will connect from a client)
   i. Install tinyproxy in Ubuntu (In the terminal: sudo apt-get install tinyproxy)
   ii. Open tinyproxy.conf file using vi or gedit (In the terminal: sudo vi /etc/tinyproxy.conf)
   iii. Uncomment the line "Allow 172.16.0.0/12" (Refer the figure below). By this, you are allowing anyone on 172.16.xx.yy to connect to your machine. Alternatively you may restrict access to one or few machines. To allow access to only your client, type the client machine's IP address (add the following entry in the file: Allow <<IP of client>>) if you choose to do this, DO NOT uncomment the entry which was specified above.
   iv. Save the file and exit.



```
#
# MaxRequestsPerChild: The number of connections a thread will handle
# before it is killed. In practise this should be set to 0, which
# disables thread reaping. If you do notice problems with memory
# leakage, then set this to something like 10000.
#
MaxRequestsPerChild 0

#
# Allow: Customization of authorization controls. If there are any
# access control keywords then the default action is to DENY. Otherwise,
# the default action is ALLOW.
#
# The order of the controls are important. All incoming connections are
# tested against the controls based on order.
#
Allow 127.0.0.1
#Allow 192.168.0.0/16
Allow 172.16.0.0/12
#Allow 10.0.0.0/8

#
# AddHeader: Adds the specified headers to outgoing HTTP requests that
# Tinyproxy makes. Note that this option will not work for HTTPS
# traffic, as Tinyproxy has no control over what headers are exchanged.
#
#AddHeader "X-My-Header" "Powered by Tinyproxy"
```

Figure.5. Tinyproxy configuration file

Now, restart Tinyproxy for the changes to take place. In the terminal: sudo /etc/init.d/tinyproxy stop sudo /etc/init.d/tinyproxy start

Default port which is used by Tinyproxy is 8888.
Do the following on the other teammate's PC (i.e., the client):
In Firefox, go to Edit → Preferences → Advanced tab →Network tab → Click on Settings →Check 'Manual proxy configurations' →enter IP of proxy server and port (default port is 8888).

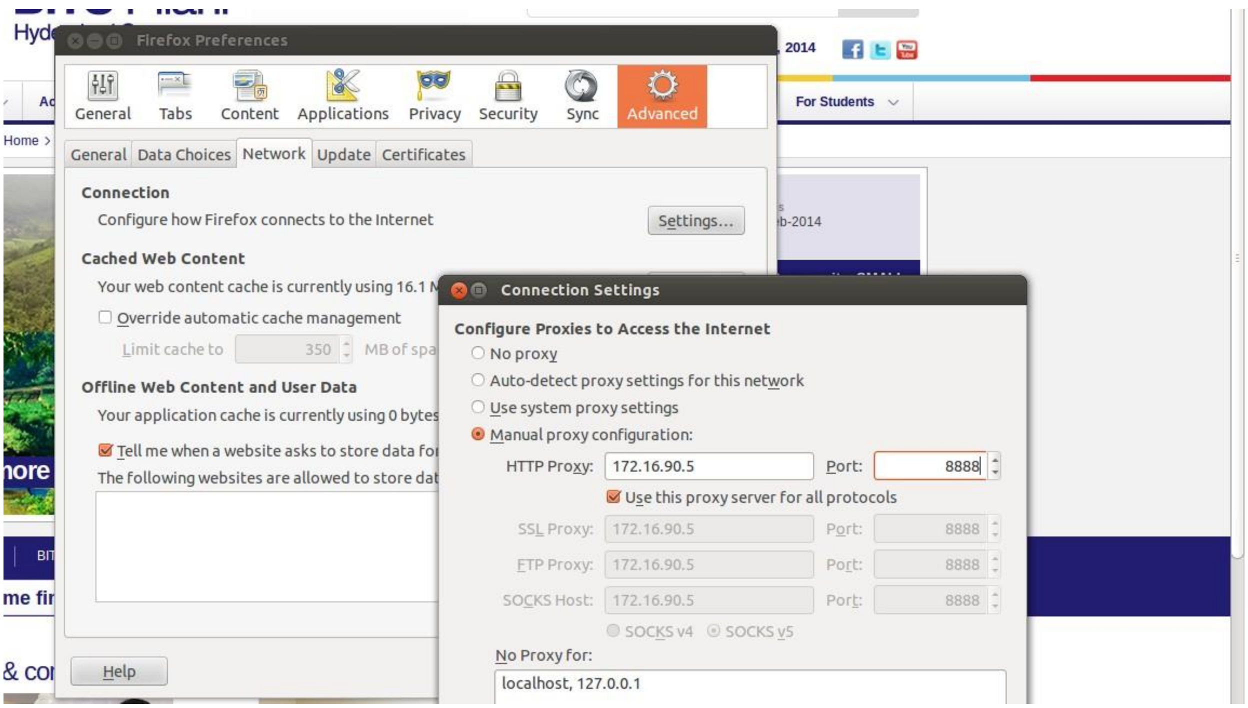


Figure.6. Firefox Settings

Now, on your client's Firefox, connect to www.google.com, and capture packets on Wireshark (i) Firstly, when you are connected to the proxy, and (ii) Secondly, without being connected to the proxy. What differences do you observe?

When not connected to proxy, you can see the connection being made with www.google.com

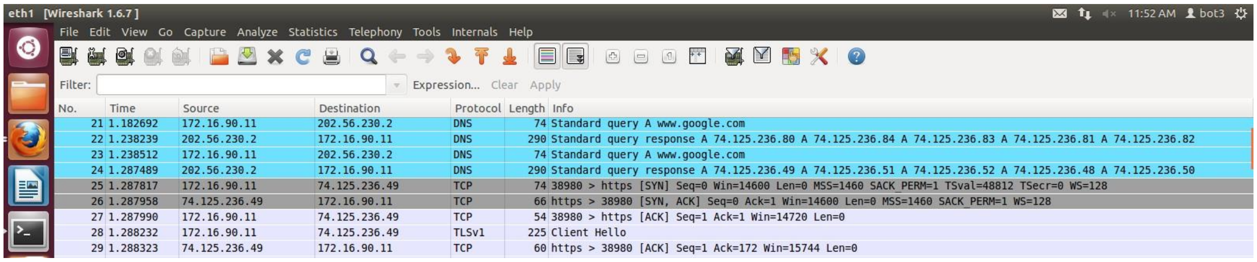


Figure.7. Response while not connected to proxy

i. When connected to the proxy, you can only see the proxy server!! (in this case, 172.26.90.4)

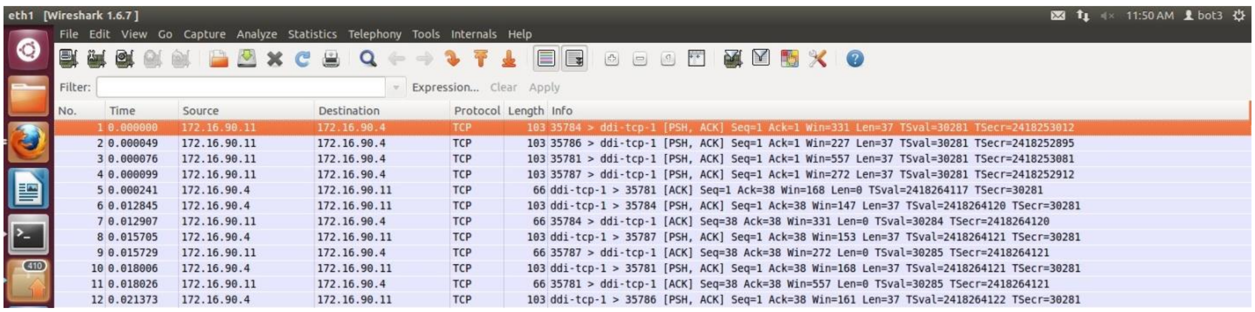


Figure.8. Response while connected to proxy

After you are done with your experiments, it is recommended to stop tinyproxy because if you're going to leave Tinyproxy running all the time, it will eventually eat all your memory and lock up your server.

sudo /etc/init.d/Tinyproxy stop

Before you leave, please remove tinyproxy from your machine so that the next lab's students can have the opportunity to configure it themselves.

sudo apt-get remove tinyproxy

DNS (Domain Name Server):

The Domain Name System (DNS) is a hierarchical naming system for computers participating in the Internet. It associates information with domain names assigned to each of the participants. Most importantly, it translates domain names meaningful to humans into the numerical (binary) identifiers associated with networking equipment for the purpose of locating and addressing these devices world- wide.

An often-used analogy to explain the Domain Name System is that it serves as the phone book for the Internet by translating human-friendly computer hostnames into IP addresses. For example, the domain name “www.example.com” translates to the address 93.184.216.119 (IPv4).

Experiment 2: Understanding the records returned by DNS website

1. Open "network-tools.com/nslook/" in a web browser. This website provides an online tool for DNS lookups.
2. Type "bits-pilani.ac.in" in the domain textbox (as shown in Figure. 9). Click "Go" button.

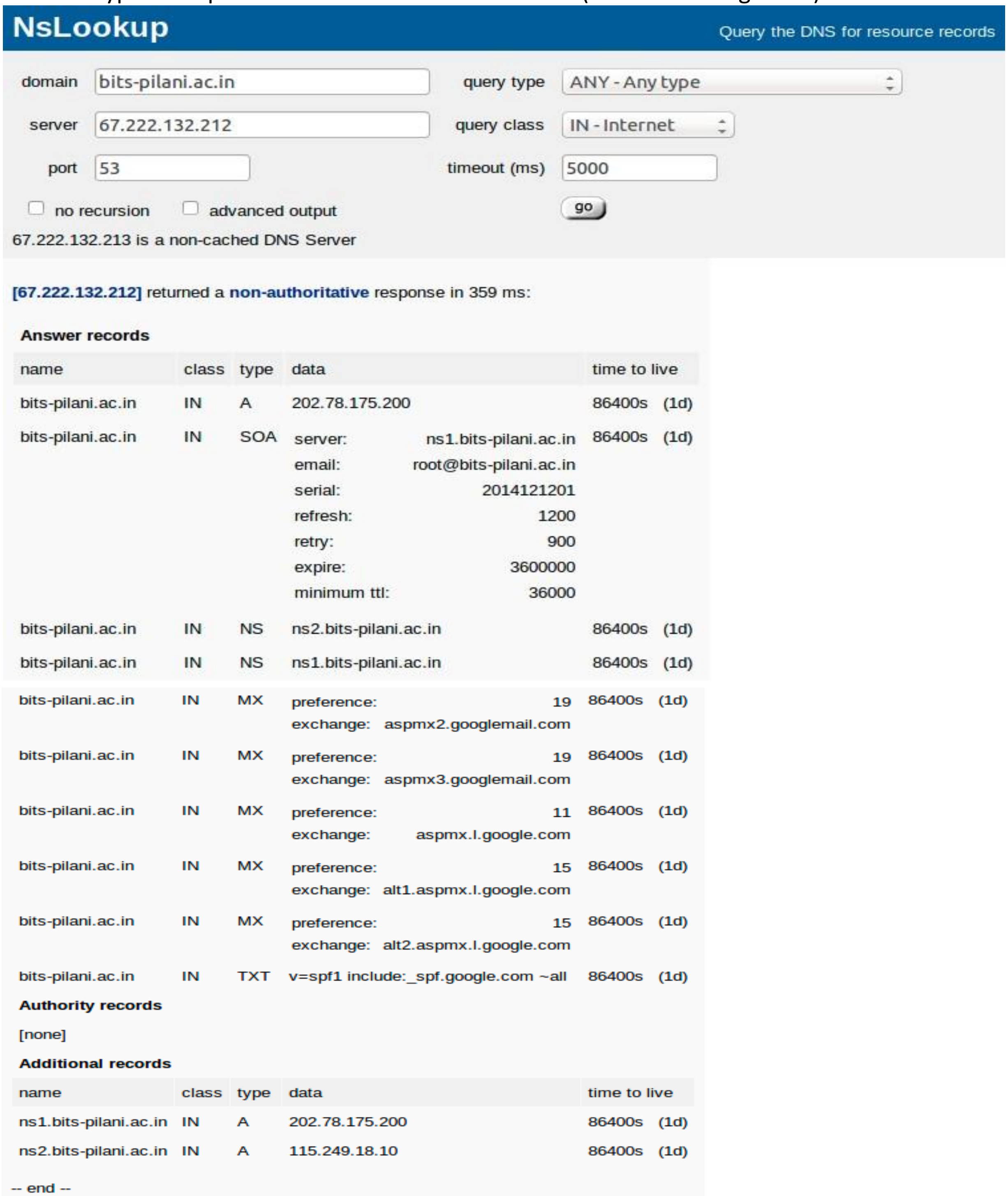


Figure.9. DNS Non-authoritative response

- What is displayed in the Answer records?
- What is the destination port number of the query message? What is the IP address to which the query message is sent?
- How many additional records are found in the DNS response?
- What are the IP addresses of BITS name servers?

3. Type “bits-pilani.ac.in” in the domain (as shown in Figure. 10). Change the server address to “202.78.175.200”. Click “Go” button.

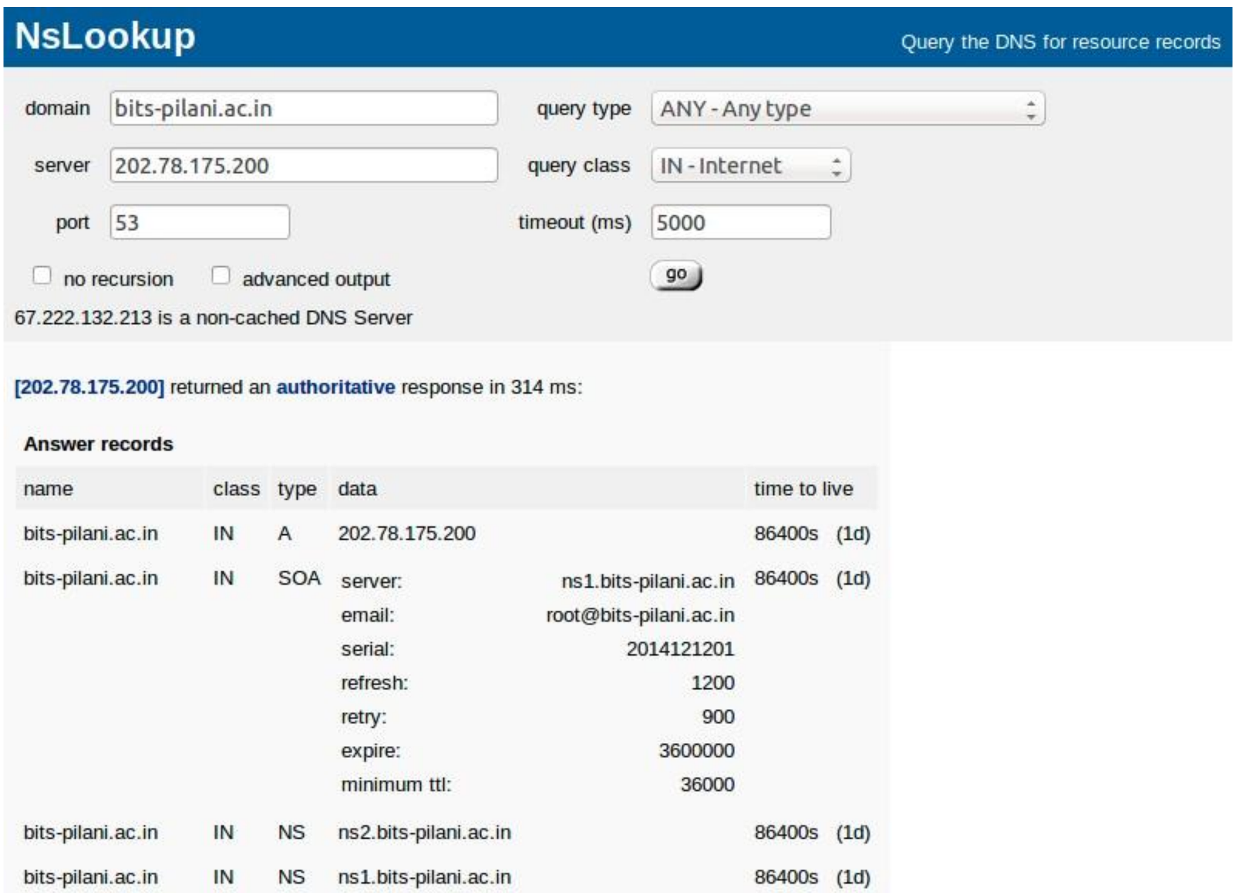
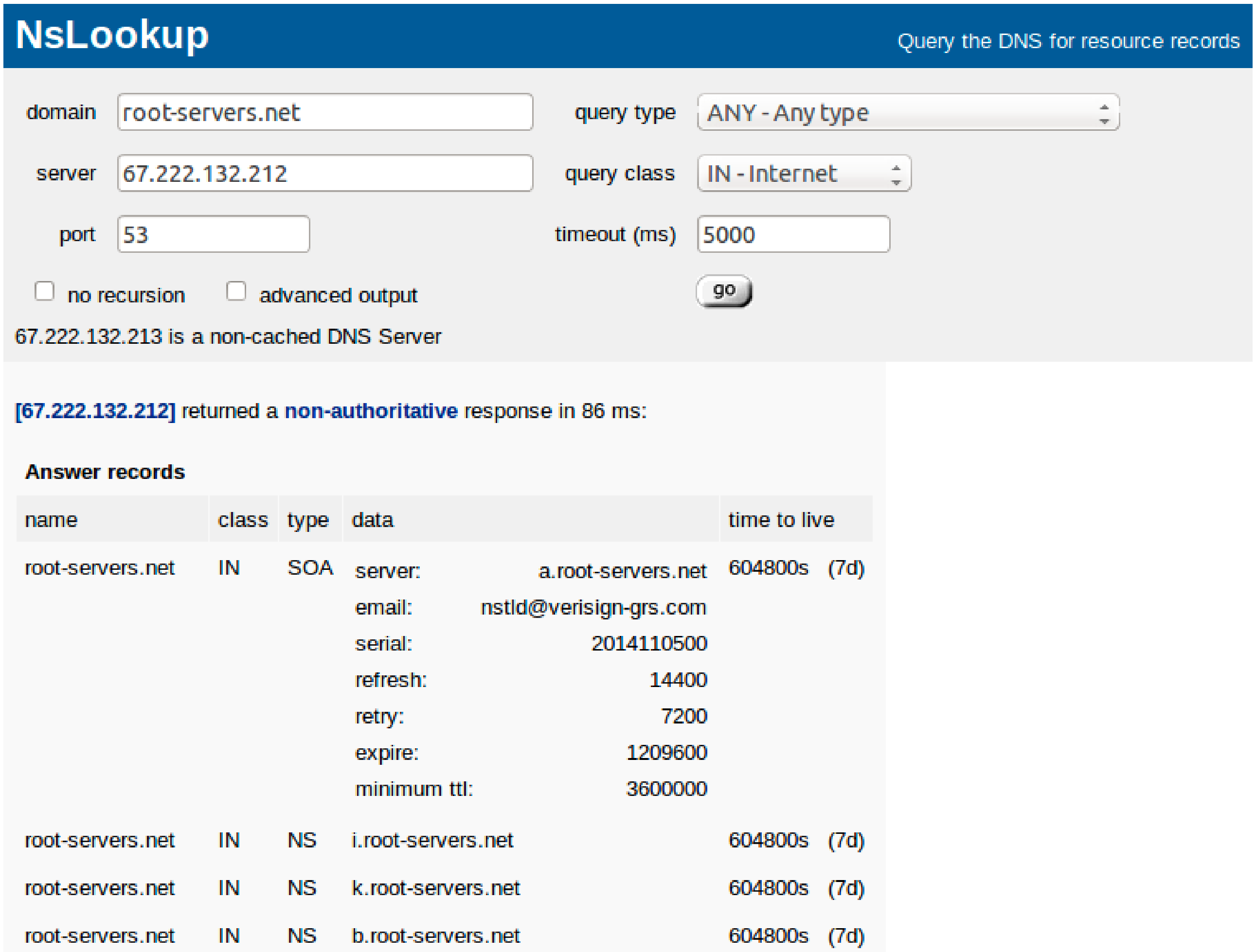


Figure.10. DNS Authoritative response

- What is an authoritative response?

4. Type “root-servers.net” in the domain (as shown in Figure.11.). Click “Go” button.
   - What is displayed in Answer records?
   - What is the total number of root servers?
   - What is displayed in the Additional records?
   - What is the IPv4 and IPv6 address of a.root-servers.net?


NsLookup
Query the DNS for resource records
domain root-servers.net
query type ANY - Any type
server 67.222.132.212
query class IN - Internet
port 53
timeout (ms) 5000
no recursion
advanced output
go
67.222.132.213 is a non-cached DNS Server
[67.222.132.212] returned a non-authoritative response in 86 ms:
Answer records
name class type data time to live
root-servers.net IN SOA server: a.root-servers.net 604800s (7d)
email: nstld@verisign-grs.com
serial: 2014110500
refresh: 14400
retry: 7200
expire: 1209600
minimum ttl: 3600000
root-servers.net IN NS i.root-servers.net 604800s (7d)
root-servers.net IN NS k.root-servers.net 604800s (7d)
root-servers.net IN NS b.root-servers.net 604800s (7d)

root-servers.net	IN	NS	f.root-servers.net	604800s (7d)
root-servers.net	IN	NS	g.root-servers.net	604800s (7d)
root-servers.net	IN	NS	h.root-servers.net	604800s (7d)
root-servers.net	IN	NS	d.root-servers.net	604800s (7d)
root-servers.net	IN	NS	e.root-servers.net	604800s (7d)
root-servers.net	IN	NS	a.root-servers.net	604800s (7d)
root-servers.net	IN	NS	j.root-servers.net	604800s (7d)
root-servers.net	IN	NS	m.root-servers.net	604800s (7d)
root-servers.net	IN	NS	l.root-servers.net	604800s (7d)
root-servers.net	IN	NS	c.root-servers.net	604800s (7d)

Authority records

[none]

Additional records

name	class	type	data	time to live
a.root-servers.net	IN	A	198.41.0.4	604800s (7d)
b.root-servers.net	IN	A	192.228.79.201	604800s (7d)
c.root-servers.net	IN	A	192.33.4.12	604800s (7d)
d.root-servers.net	IN	A	199.7.91.13	604800s (7d)
e.root-servers.net	IN	A	192.203.230.10	604800s (7d)
f.root-servers.net	IN	A	192.5.5.241	604800s (7d)
g.root-servers.net	IN	A	192.112.36.4	604800s (7d)
h.root-servers.net	IN	A	128.63.2.53	604800s (7d)
i.root-servers.net	IN	A	192.36.148.17	604800s (7d)
j.root-servers.net	IN	A	192.58.128.30	604800s (7d)
k.root-servers.net	IN	A	193.0.14.129	604800s (7d)
l.root-servers.net	IN	A	199.7.83.42	604800s (7d)
m.root-servers.net	IN	A	202.12.27.33	604800s (7d)

-- end --

Figure.11. DNS Root servers

References

- ➢ Wireshark HTTP: http://code.bretonstyle.net/?page_id=176
- ➢ Tinyproxy link1: http://www.justinmccandless.com/blog/Set+Up+Tinyproxy+in+Ubuntu
- ➢ Tinyproxy link2: http://www.gypthecat.com/tinyproxy-a-quick-and-easy-proxy-server-on-ubuntu
- ➢ Online DNS Lookup Tool: network-tools.com/nslook