RAMAIAH Institute of Technology

Department of Computer Science & Engineering

IV SEMESTER

STUDENTS LAB MANUAL

for

DATA COMMUNICATION AND NETWORKING LABORATORY

Subject code: CSL47

Course Credits 0:0:1

Term: March - June 2025

Faculties Incharge:

Nandini S B

Dr. Sangeetha V

Soumya C S

Syllabus

Sl. No	Tools/ Environment	Cycle I	Page No.
1.	Wireshark	Trace Hypertext Transfer Protocol using Wireshark	5
2.	Wireshark	Trace Domain Name Server using Wireshark.	12
3.	Wireshark	Trace Internet Protocol and Internet Control Message Protocol using Wireshark.	16
4.	Wireshark	Trace Dynamic Host Configuration Protocol using Wireshark.	20
5.	C/C++/Java	Write a program for error detection using CRC-CCITT(16-bits).	25
6.	C/C++/Java	Write a program to find the shortest path between vertices using Bellman-Ford algorithm.	28
7.	C/C++/Java	Write a program for congestion control using leaky bucket algorithm.	32
Cycle II			
8.	C/C++/Java	Using TCP/IP sockets, write a client – server program where the client send the file name and the server send back the contents of the requested file if present.	35
9.	C/C++/Java	Write a program for Time Division Multiplexing Simulator. Show how the time division multiplexing technique works.	38
10.	Network Simulator - 3	Design and simulate a wired network with duplex links between ‘n’ nodes with CDR over UDP. Set the queue size vary the bandwidth and find the number of packets dropped.	42
11.	Network Simulator - 3	Design and simulate a four node point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP agent between n1-n3. Apply relevant applications over TCP and UDP agents by changing the parameters and determine the number of packets sent by TCP/UDP.	45
12.	Network Simulator - 3	Design and simulate simple Extended Service Set with transmitting nodes in wireless LAN and determine the performance with respect to transmission of Packets.	50
13.	Network Simulator - 3	Design and simulate infrastructure less network, generate two traffic flows between nodes and analyse its performance.	56
14.	Network Simulator - 3	Design a wired network with ‘n’ nodes to observe the performance of two TCP variants (Reno and Tahoe). Simulate the designed network and observe the network performance.	61

Introduction to Wireshark:

Wireshark is a widely used, open source network analyser that can capture and display real-time details of network traffic. It is particularly useful for troubleshooting network issues, analysing network protocols and ensuring network security. Wireshark is one such tool that can offer an in-depth view into network activities, diagnose network performance issues or identify potential security threats.

Wireshark is a **network protocol analyzer**, which means it's a tool used to capture and analyze the data traffic on a computer network in real-time. It's widely used by network administrators, security professionals, developers, and even hobbyists to troubleshoot network issues, monitor network activity, and analyze security problems.

Key Features:

- **Packet Capture**: Captures packets (units of data) as they travel across a network.
- **Detailed Analysis**: Decodes and displays protocols at various levels of the OSI model (like TCP, HTTP, DNS).
- **Filtering**: Powerful display filters help narrow down exactly what you're looking for.
- **Live Capture and Offline Analysis**: You can watch traffic as it happens or open saved packet capture files.
- **Cross-Platform**: Works on Windows, macOS, and Linux.

How to Install:

Linux (Ubuntu/Debian)

Step 1: Open a terminal and run

```
sudo apt update

sudo apt install wireshark
```

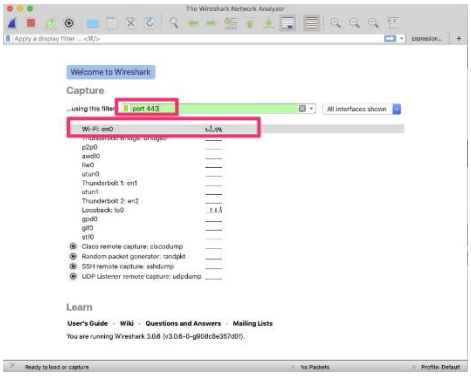
Step 2: During installation, you'll be prompted about allowing non-root users to capture packets. If you say "Yes", add your user to the wireshark group

```
sudo usermod -aG wireshark $USER
```

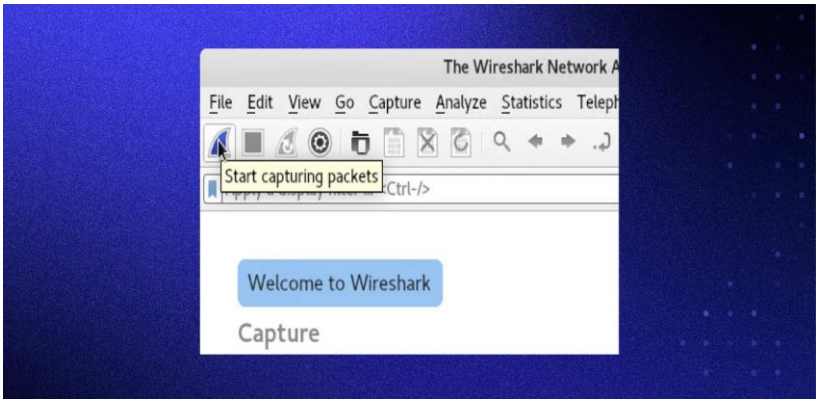
Step 3: Log out and log back in for the group change to take effect.

After Installation

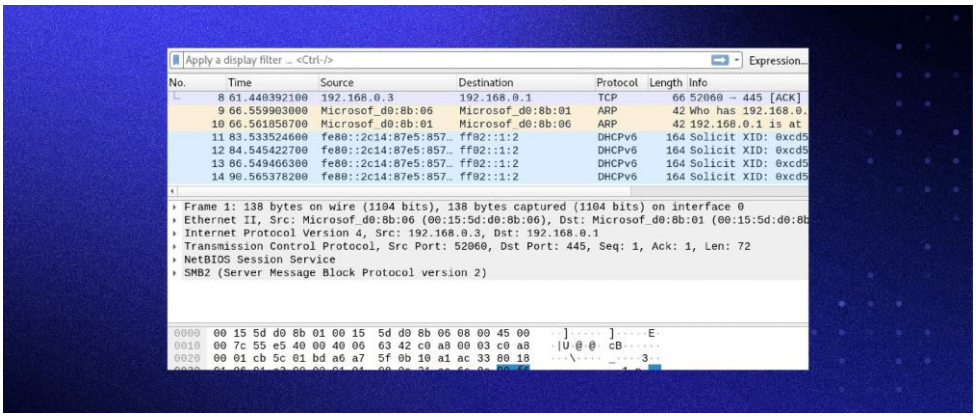
- Launch Wireshark.
  - Open Terminal: sudo wireshark.
- You'll see a list of network interfaces (like Ethernet, Wi-Fi).



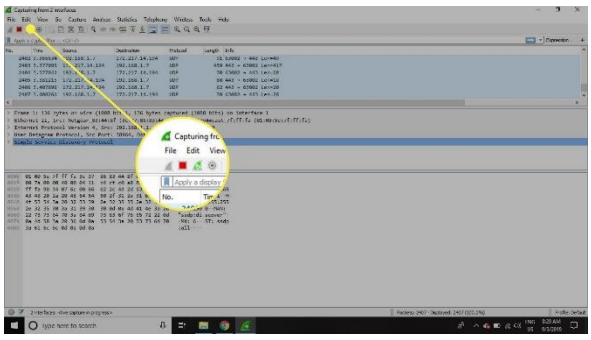
- Select one and click **Start capturing packets**.



- Packet capturing starts:



- Stop Capturing



Cycle - 1

Lab Session 1

Problem Statement: Trace Hypertext Transfer Protocol using Wireshark

Exercise Question with Solution:

Trace Hypertext Transfer Protocol. (Part A)

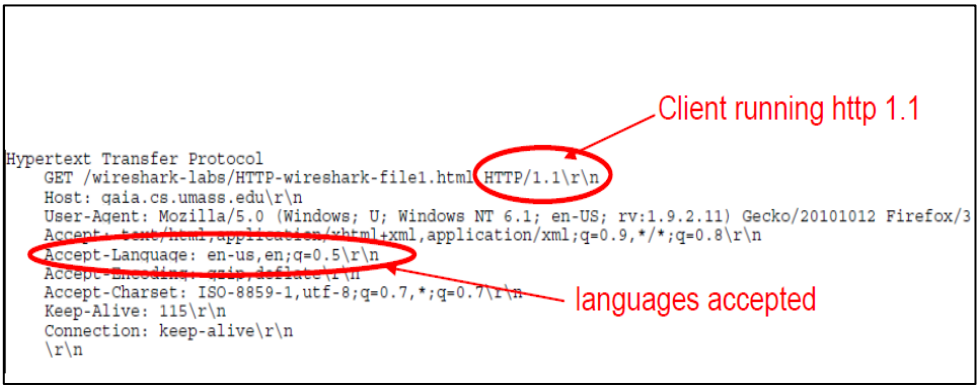
The Basic HTTP GET/response interaction

Let's begin exploration of HTTP by downloading a very simple HTML file - one that is very short, and contains no embedded objects. Do the following:

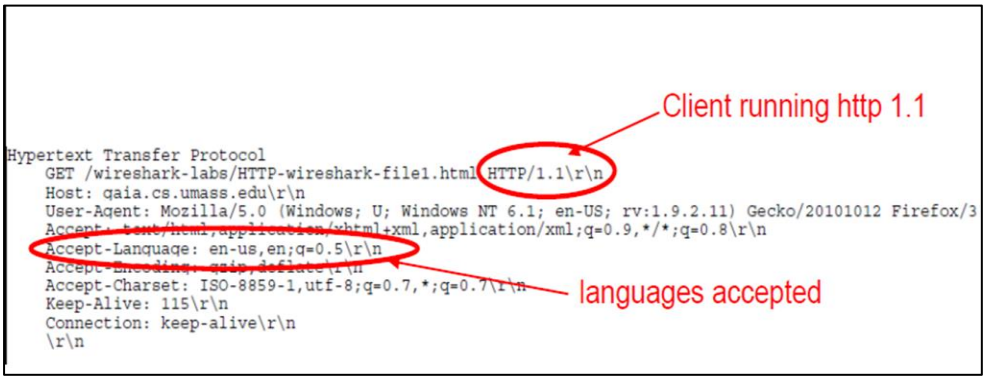
1. **Start web browser.**
2. **Start Wireshark packet sniffer(but don't begin packet capture).**
3. **Enter "http" in the display-filter-specification window**
4. **Wait a bit more than one minute, and then begin Wireshark packet capture.**
5. **Enter the following to your browser**

http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html
browser should display the very simple, one-line HTML file.

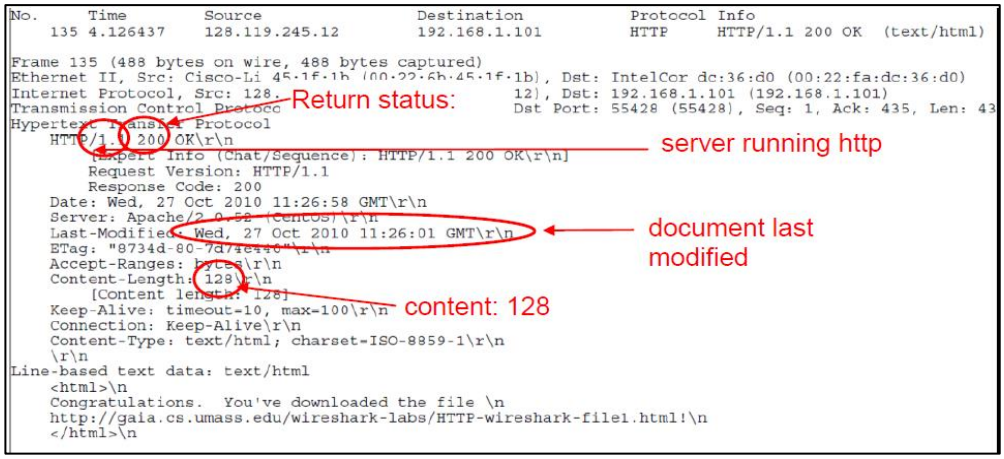
a) Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?



b) What languages does your browser indicate that it can accept from the server?

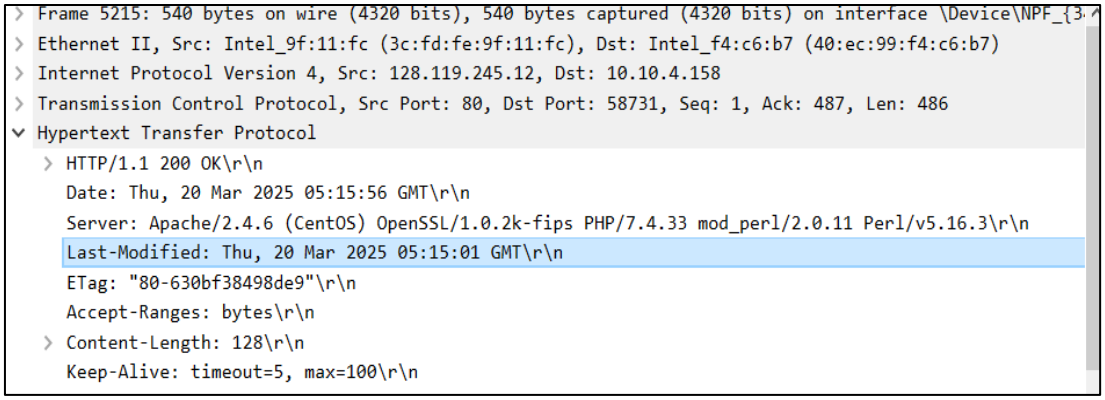


c) What is the status code returned from the server to your browser?



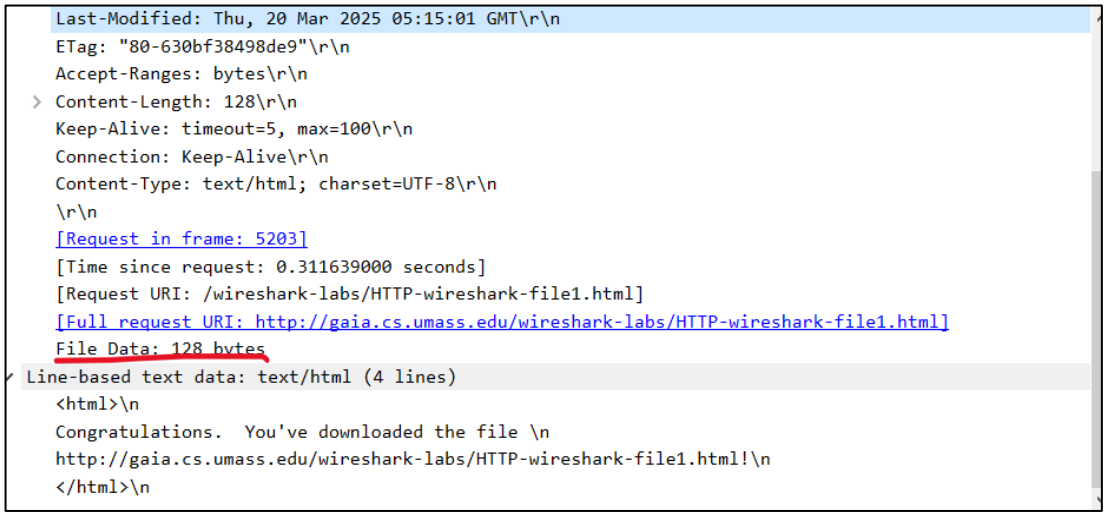

```
No.     Time        Source                Destination           Protocol Info
    135 4.126437    128.119.245.12        192.168.1.101         HTTP     HTTP/1.1 200 OK  (text/html)

Frame 135 (488 bytes on wire, 488 bytes captured)
Ethernet II, Src: Cisco-Li 45:1f:1b (00:22:6b:45:1f:1b), Dst: IntelCor dc:36:d0 (00:22:fa:dc:36:d0)
Internet Protocol, Src: 128.                12], Dst: 192.168.1.101 (192.168.1.101)
Transmission Control Protocc                Dst Port: 55428 (55428), Seq: 1, Ack: 435, Len: 43
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
        [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
        Request Version: HTTP/1.1
        Response Code: 200
    Date: Wed, 27 Oct 2010 11:26:58 GMT\r\n
    Server: Apache/2.0.52 (CentOS)\r\n
    Last-Modified: Wed, 27 Oct 2010 11:26:01 GMT\r\n
    ETag: "8734d-80-7d74e440"\r\n
    Accept-Ranges: bytes\r\n
    Content-Length: 128\r\n
        [Content length: 128]
    Keep-Alive: timeout=10, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=ISO-8859-1\r\n
    \r\n
Line-based text data: text/html
    <html>\n
    Congratulations.  You've downloaded the file \n
    http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html!\n
    </html>\n
```

d) When was the HTML file, that you are retrieving last modified at the server?



```
> Frame 5215: 540 bytes on wire (4320 bits), 540 bytes captured (4320 bits) on interface \Device\NPF_{3
> Ethernet II, Src: Intel_9f:11:fc (3c:fd:fe:9f:11:fc), Dst: Intel_f4:c6:b7 (40:ec:99:f4:c6:b7)
> Internet Protocol Version 4, Src: 128.119.245.12, Dst: 10.10.4.158
> Transmission Control Protocol, Src Port: 80, Dst Port: 58731, Seq: 1, Ack: 487, Len: 486
v Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Date: Thu, 20 Mar 2025 05:15:56 GMT\r\n
    Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.33 mod_perl/2.0.11 Perl/v5.16.3\r\n
    Last-Modified: Thu, 20 Mar 2025 05:15:01 GMT\r\n
    ETag: "80-630bf38498de9"\r\n
    Accept-Ranges: bytes\r\n
  > Content-Length: 128\r\n
    Keep-Alive: timeout=5, max=100\r\n
```

e) How many bytes of content are being returned to your browser?



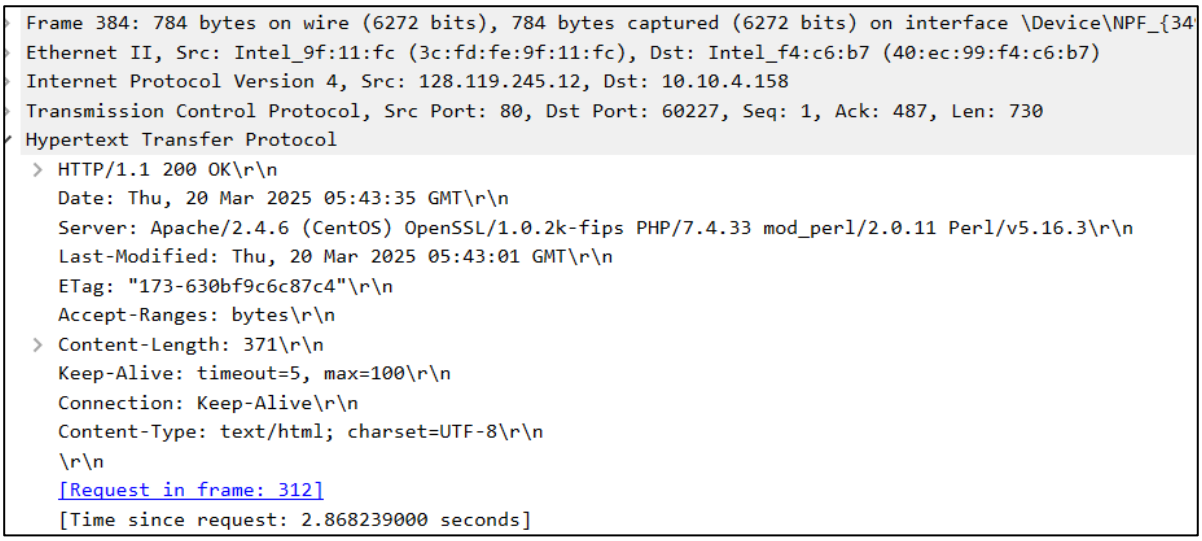
```
    Last-Modified: Thu, 20 Mar 2025 05:15:01 GMT\r\n
    ETag: "80-630bf38498de9"\r\n
    Accept-Ranges: bytes\r\n
  > Content-Length: 128\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [Request in frame: 5203]
    [Time since request: 0.311639000 seconds]
    [Request URI: /wireshark-labs/HTTP-wireshark-file1.html]
    [Full request URI: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html]
    File Data: 128 bytes
v Line-based text data: text/html (4 lines)
    <html>\n
    Congratulations.  You've downloaded the file \n
    http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html!\n
    </html>\n
```

The HTTP CONDITIONAL GET/response interaction

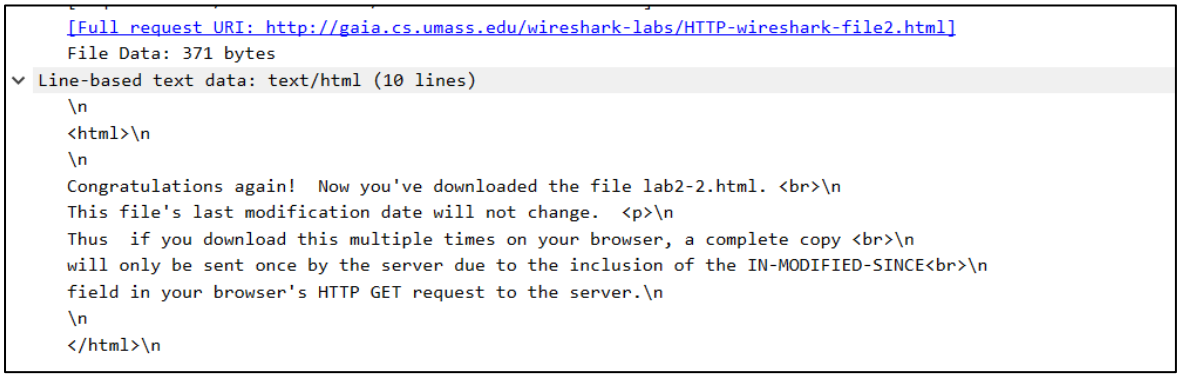
Perform the following:

- Start up your web browser, and make sure your browser's cache is cleared
- Start up the Wireshark packet sniffer.
- Enter the following URL into your browser
  http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html
- Your browser should display a very simple five-line HTML file.
- Open a new tab on the same browser and enter the same URL again

f) Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE" line in the HTTP GET?
Answer is NO

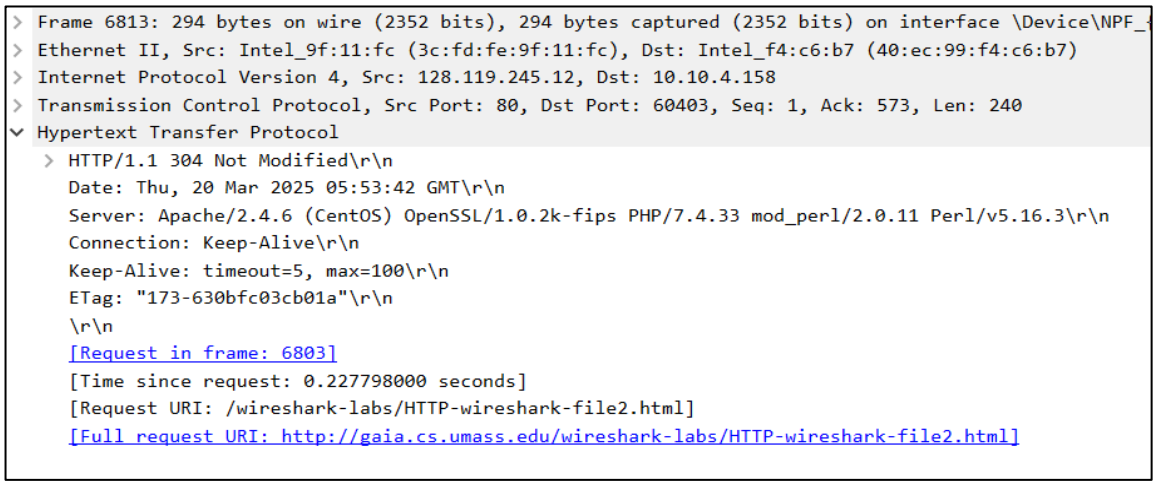


g) Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?
Answer is Yes



h) Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE:” line in the HTTP GET? If so, what information follows the “IF-MODIFIED-SINCE:” header?
Answer is Yes

i) What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.
Answer is 304 Not Modified, No

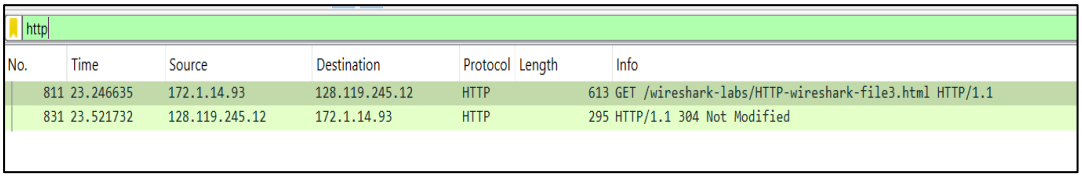


Trace Hypertext Transfer Protocol (Part B)

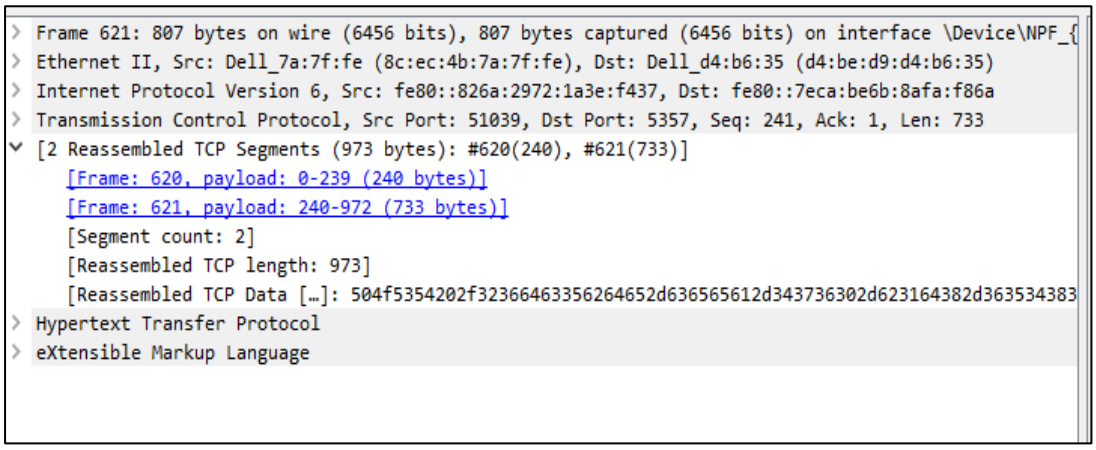
Retrieving Long Documents

URL: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html

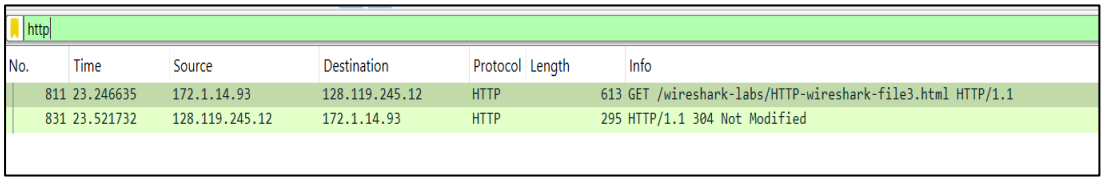
a) How many HTTP GET request messages were sent by your browser?
Answer is 1



b) How many data-containing TCP segments were needed to carry the single HTTP Response?
Answer is 2



c) What is the status code and phrase associated with the response to the HTTP GET Request?
Answer is 304 Not modified



HTML Documents with Embedded Objects

Perform the following:

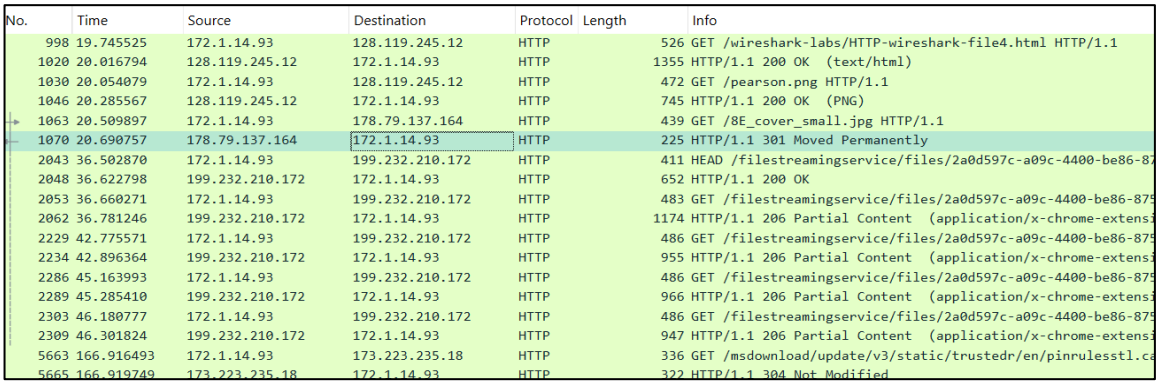
- Start up your web browser, and make sure your browser's cache is cleared
- Start up the Wireshark packet sniffer.
- Enter the following URL into your browser

http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html
Note: Your browser should display a short HTML file with two images.
These two images are referenced in the base HTML file.
That is, the images themselves are not contained in the HTML; instead the URLs for the images are contained in the downloaded HTML file.

Your browser will have to retrieve these logos from the indicated web sites.

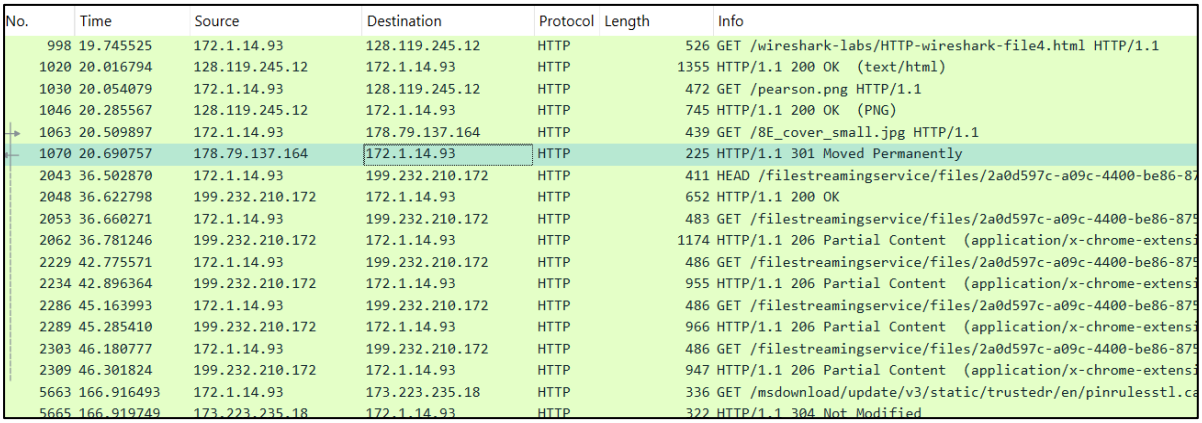
1. The publisher's logo is retrieved from the www.aw-bc.com web site.
2. The image of the book's cover is stored at the manic.cs.umass.edu server.

d) How many HTTP GET request messages were sent by your browser? To which Internet addresses were these GET requests sent?
Answer is 3



No.	Time	Source	Destination	Protocol	Length	Info
998	19.745525	172.1.14.93	128.119.245.12	HTTP	526	GET /wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
1020	20.016794	128.119.245.12	172.1.14.93	HTTP	1355	HTTP/1.1 200 OK (text/html)
1030	20.054079	172.1.14.93	128.119.245.12	HTTP	472	GET /pearson.png HTTP/1.1
1046	20.285567	128.119.245.12	172.1.14.93	HTTP	745	HTTP/1.1 200 OK (PNG)
1063	20.509897	172.1.14.93	178.79.137.164	HTTP	439	GET /8E_cover_small.jpg HTTP/1.1
1070	20.690757	178.79.137.164	172.1.14.93	HTTP	225	HTTP/1.1 301 Moved Permanently
2043	36.502870	172.1.14.93	199.232.210.172	HTTP	411	HEAD /filestreamingservice/files/2a0d597c-a09c-4400-be86-87
2048	36.622798	199.232.210.172	172.1.14.93	HTTP	652	HTTP/1.1 200 OK
2053	36.660271	172.1.14.93	199.232.210.172	HTTP	483	GET /filestreamingservice/files/2a0d597c-a09c-4400-be86-875
2062	36.781246	199.232.210.172	172.1.14.93	HTTP	1174	HTTP/1.1 206 Partial Content (application/x-chrome-extensi
2229	42.775571	172.1.14.93	199.232.210.172	HTTP	486	GET /filestreamingservice/files/2a0d597c-a09c-4400-be86-875
2234	42.896364	199.232.210.172	172.1.14.93	HTTP	955	HTTP/1.1 206 Partial Content (application/x-chrome-extensi
2286	45.163993	172.1.14.93	199.232.210.172	HTTP	486	GET /filestreamingservice/files/2a0d597c-a09c-4400-be86-875
2289	45.285410	199.232.210.172	172.1.14.93	HTTP	966	HTTP/1.1 206 Partial Content (application/x-chrome-extensi
2303	46.180777	172.1.14.93	199.232.210.172	HTTP	486	GET /filestreamingservice/files/2a0d597c-a09c-4400-be86-875
2309	46.301824	199.232.210.172	172.1.14.93	HTTP	947	HTTP/1.1 206 Partial Content (application/x-chrome-extensi
5663	166.916493	172.1.14.93	173.223.235.18	HTTP	336	GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.ca
5665	166.919749	173.223.235.18	172.1.14.93	HTTP	322	HTTP/1.1 304 Not Modified

e) Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.
Answer is Serial



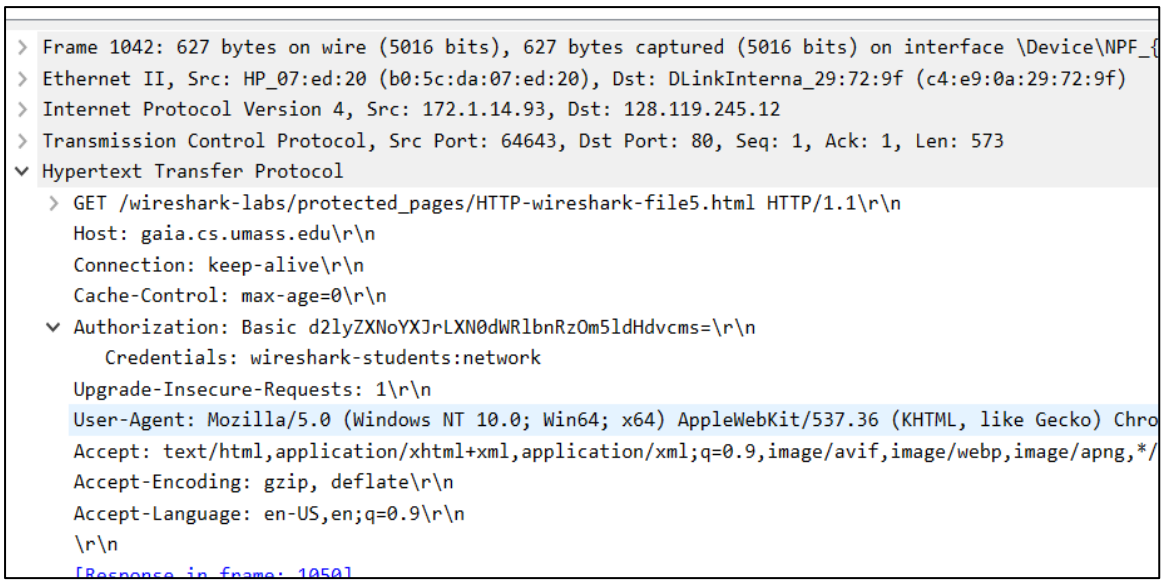
No.	Time	Source	Destination	Protocol	Length	Info
998	19.745525	172.1.14.93	128.119.245.12	HTTP	526	GET /wireshark-labs/HTTP-wireshark-file4.html HTTP/1.1
1020	20.016794	128.119.245.12	172.1.14.93	HTTP	1355	HTTP/1.1 200 OK (text/html)
1030	20.054079	172.1.14.93	128.119.245.12	HTTP	472	GET /pearson.png HTTP/1.1
1046	20.285567	128.119.245.12	172.1.14.93	HTTP	745	HTTP/1.1 200 OK (PNG)
1063	20.509897	172.1.14.93	178.79.137.164	HTTP	439	GET /8E_cover_small.jpg HTTP/1.1
1070	20.690757	178.79.137.164	172.1.14.93	HTTP	225	HTTP/1.1 301 Moved Permanently
2043	36.502870	172.1.14.93	199.232.210.172	HTTP	411	HEAD /filestreamingservice/files/2a0d597c-a09c-4400-be86-87
2048	36.622798	199.232.210.172	172.1.14.93	HTTP	652	HTTP/1.1 200 OK
2053	36.660271	172.1.14.93	199.232.210.172	HTTP	483	GET /filestreamingservice/files/2a0d597c-a09c-4400-be86-875
2062	36.781246	199.232.210.172	172.1.14.93	HTTP	1174	HTTP/1.1 206 Partial Content (application/x-chrome-extensi
2229	42.775571	172.1.14.93	199.232.210.172	HTTP	486	GET /filestreamingservice/files/2a0d597c-a09c-4400-be86-875
2234	42.896364	199.232.210.172	172.1.14.93	HTTP	955	HTTP/1.1 206 Partial Content (application/x-chrome-extensi
2286	45.163993	172.1.14.93	199.232.210.172	HTTP	486	GET /filestreamingservice/files/2a0d597c-a09c-4400-be86-875
2289	45.285410	199.232.210.172	172.1.14.93	HTTP	966	HTTP/1.1 206 Partial Content (application/x-chrome-extensi
2303	46.180777	172.1.14.93	199.232.210.172	HTTP	486	GET /filestreamingservice/files/2a0d597c-a09c-4400-be86-875
2309	46.301824	199.232.210.172	172.1.14.93	HTTP	947	HTTP/1.1 206 Partial Content (application/x-chrome-extensi
5663	166.916493	172.1.14.93	173.223.235.18	HTTP	336	GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.ca
5665	166.919749	173.223.235.18	172.1.14.93	HTTP	322	HTTP/1.1 304 Not Modified

HTTP Authentication

URL:
http://gaia.cs.umass.edu/wireshark-labs/**protected**_pages/HTTP-wireshark-file5.html
username: wireshark-students
password: network

f) What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?
Answer is 401 unauthorized

g) When your browser sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?



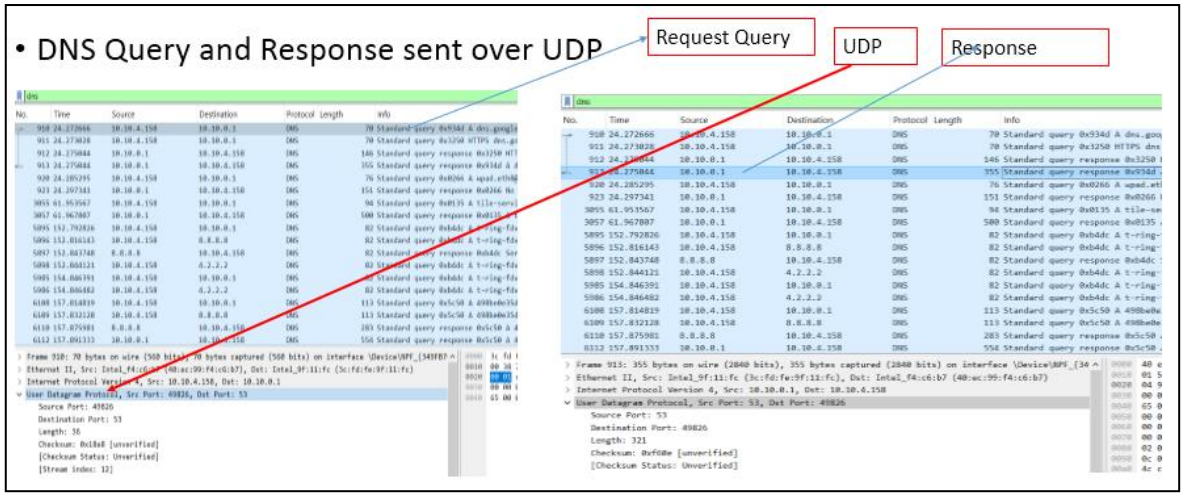
Lab Session 2

Problem Statement: Trace Domain Name Server using packet sniffer and packet analyser.

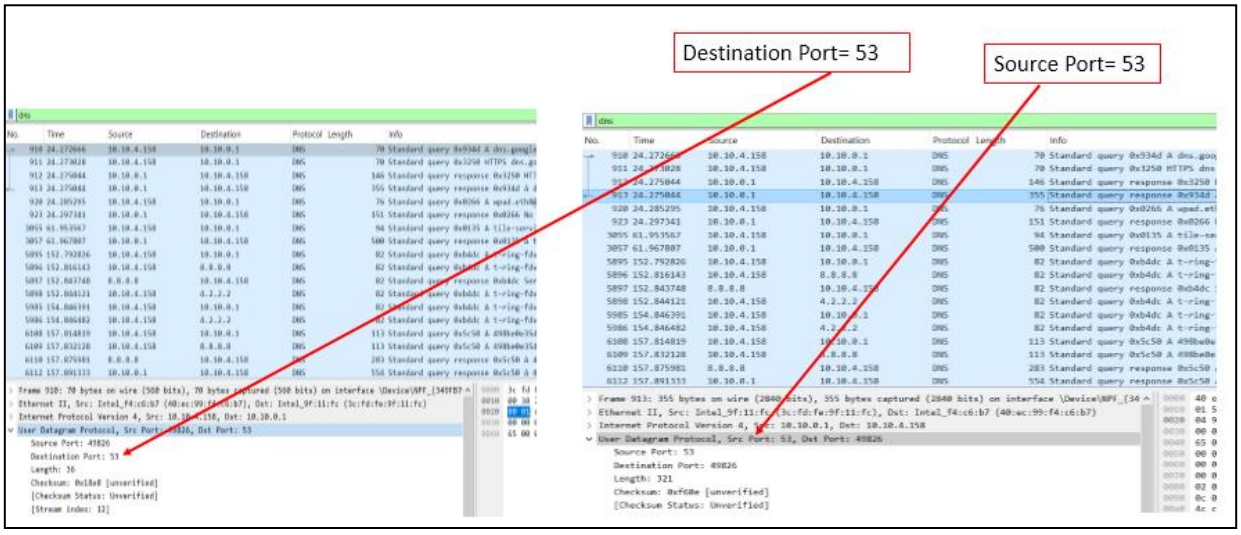
Exercise Question with Solution:

- Start packet capture in Wireshark.
- With your browser, visit the Web page: http://www.ietf.org

a) Locate the DNS query and response messages. Are they sent over UDP or TCP?

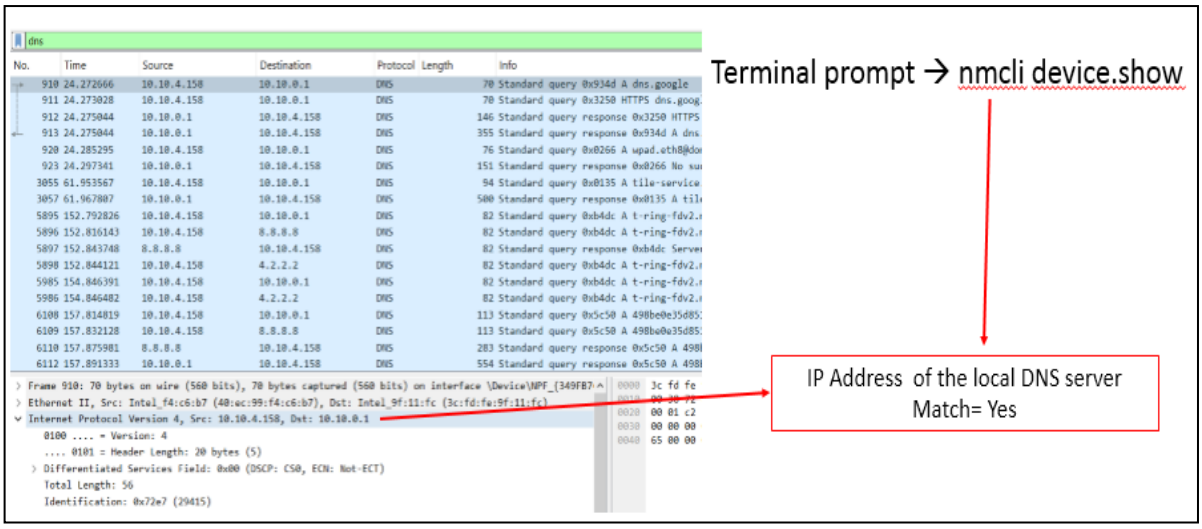


b) What is the destination port for the DNS query message? What is the source port of DNS response message?

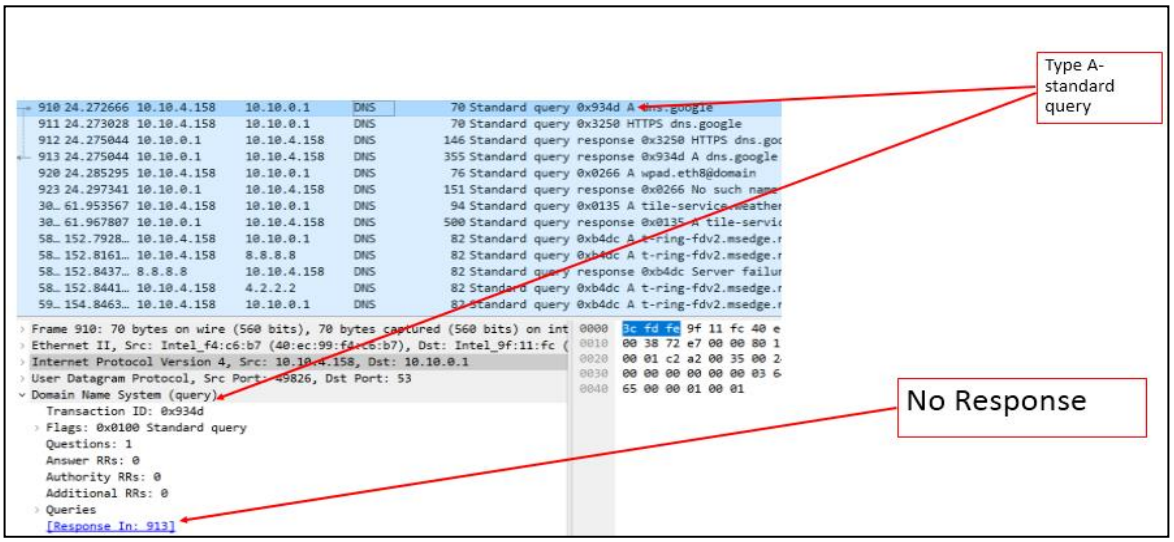


c) To what IP address is the DNS query message sent? Determine the IP address of your local DNS server (*Hint: nmcli*). Are these two IP addresses the same?

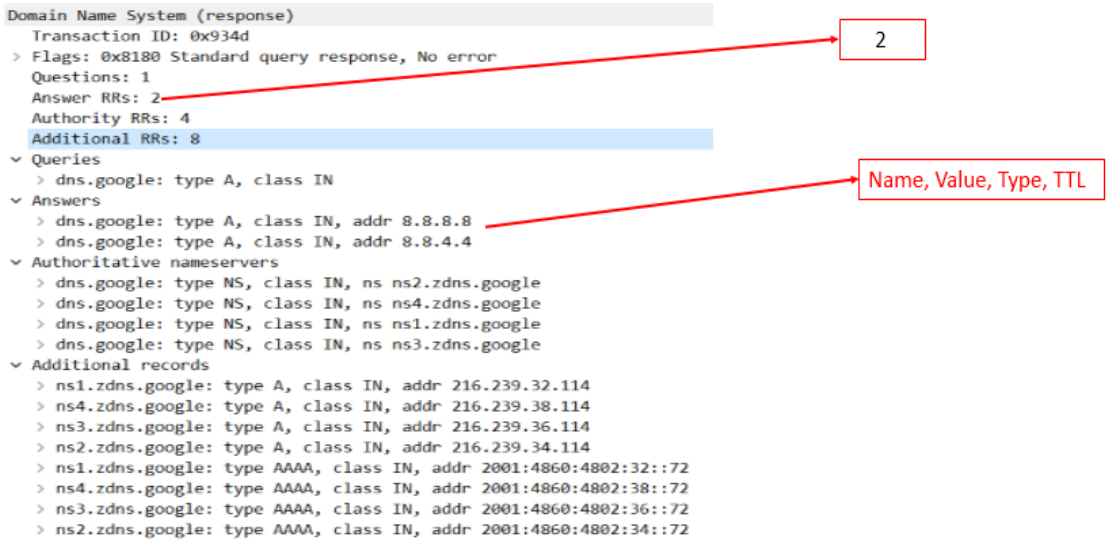
```
msrit@msrit:~$ nmcli device show
GENERAL.DEVICE:                         eno1
GENERAL.TYPE:                           ethernet
GENERAL.HWADDR:                         2C:F0:5D:17:6C:F1
GENERAL.MTU:                            1500
GENERAL.STATE:                          100 (connected)
GENERAL.CONNECTION:                     Wired connection 1
GENERAL.CON-PATH:                       /org/freedesktop/NetworkManager/ActiveC>
WIRED-PROPERTIES.CARRIER:               on
IP4.ADDRESS[1]:                         172.1.6.75/23
IP4.GATEWAY:                            172.1.6.1
IP4.ROUTE[1]:                           dst = 172.1.6.0/23, nh = 0.0.0.0, mt = >
IP4.ROUTE[2]:                           dst = 0.0.0.0/0, nh = 172.1.6.1, mt = 1>
IP4.DNS[1]:                             8.8.8.8
IP4.DNS[2]:                             172.1.2.2
IP4.DNS[3]:                             4.2.2.2
IP4.DNS[4]:                             172.1.20.6
IP4.DOMAIN[1]:                          RIT.EDU
IP6.ADDRESS[1]:                         fe80::4b42:bdaa:546d:a5ac/64
IP6.GATEWAY:                            --
IP6.ROUTE[1]:                           dst = fe80::/64, nh = ::, mt = 1024
```



d) Examine the DNS query message. What "Type" of DNS query is it? Does the query message contain any "answers"?



e) Examine the DNS response message. How many "answers" are provided? What does each of these answers contain?



f) Consider the subsequent TCP SYN packet sent by your host. Does the destination IP address of the SYN packet correspond to any of the IP addresses provided in the DNS response message?
ANSWER: The first SYN packet was sent to 209.173.57.180 which corresponds to the first IP address provided in the DNS response message.

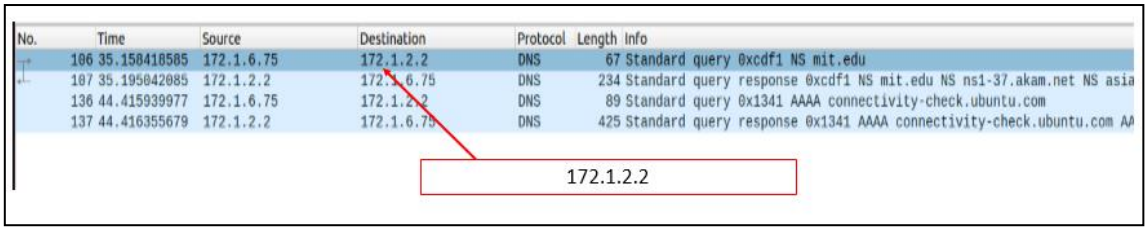
g) This web page contains images. Before retrieving each image, does your host issue new DNS queries?
Answer: No

Do an nslookup on www.mit.edu

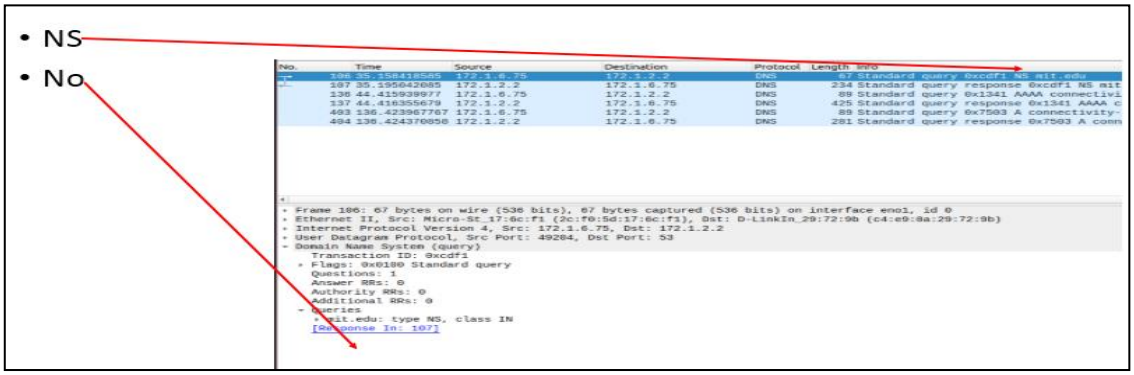
Type the command nslookup –type=NS mit.edu

```
msrit@msrit:~$ nslookup -type=NS mit.edu
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
mit.edu nameserver = ns1-173.akam.net.
mit.edu nameserver = use2.akam.net.
mit.edu nameserver = ns1-37.akam.net.
mit.edu nameserver = use5.akam.net.
mit.edu nameserver = usw2.akam.net.
mit.edu nameserver = asia1.akam.net.
mit.edu nameserver = asia2.akam.net.
mit.edu nameserver = eur5.akam.net.

Authoritative answers can be found from:
```

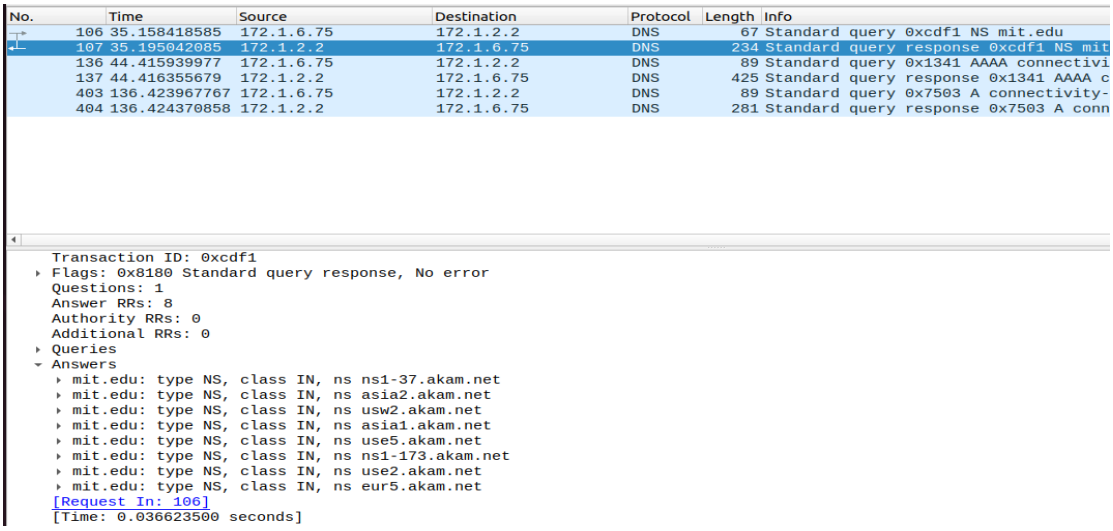
h) To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server?



i) Examine the DNS query message. What "Type" of DNS query is it? Does the query message contain any "answers"?



j) Examine the DNS response message. What MIT name servers does the response message provide? Does this response message also provide the IP addresses of the MIT name servers?

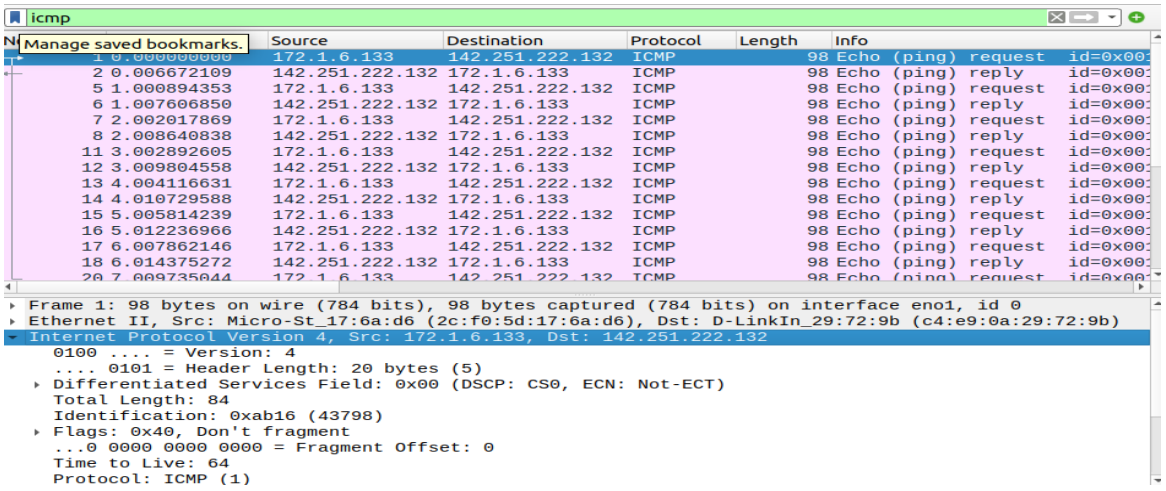


Lab Session 3

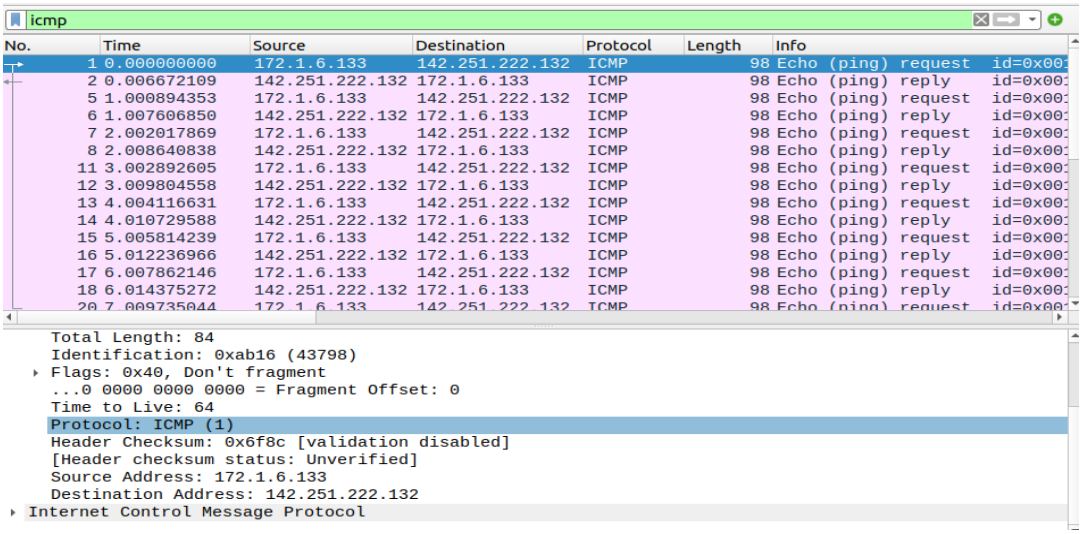
Problem Statement: Trace Internet Protocol and Internet Control Message Protocol using Wireshark.

Exercise Question with Solution:

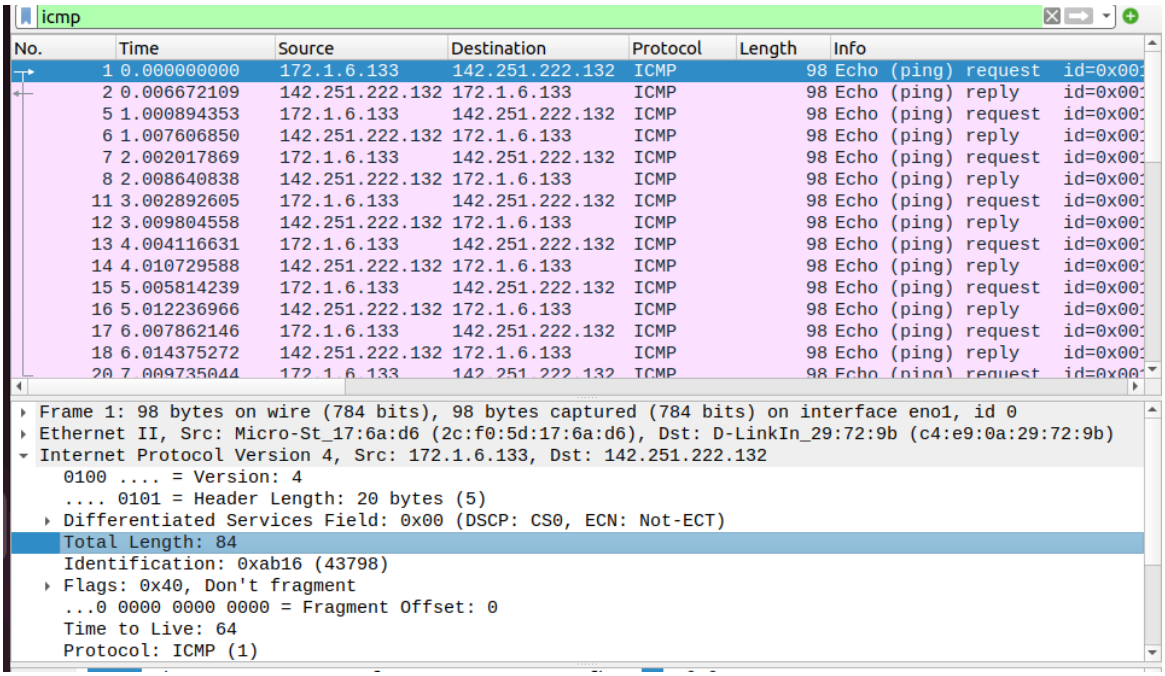
a) Select the first ICMP Echo Request message sent by your computer, and expand the Internet Protocol part of the packet in the packet details window. What is the IP address of your computer?



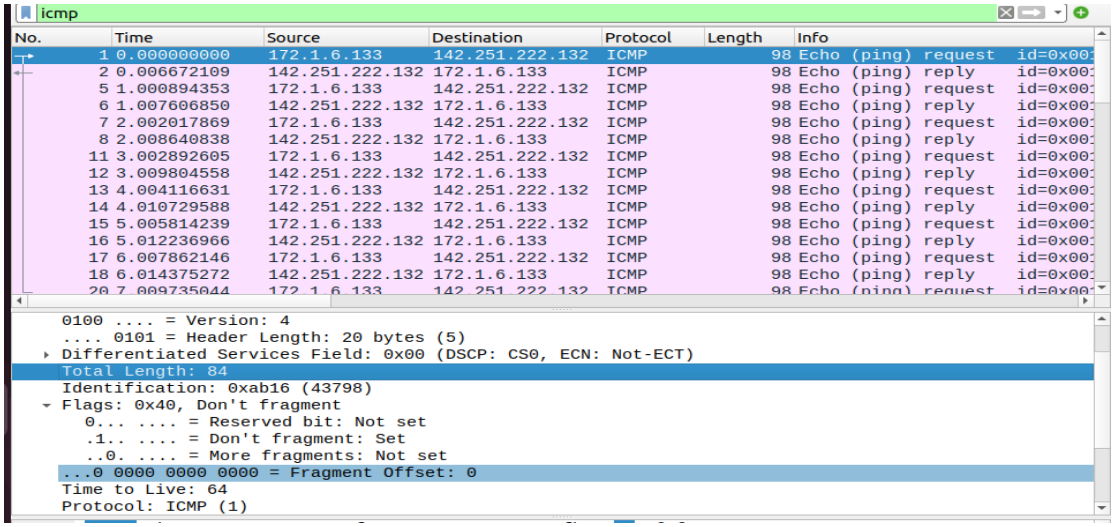
b) Within the IP packet header, what is the value in the upper layer protocol field?



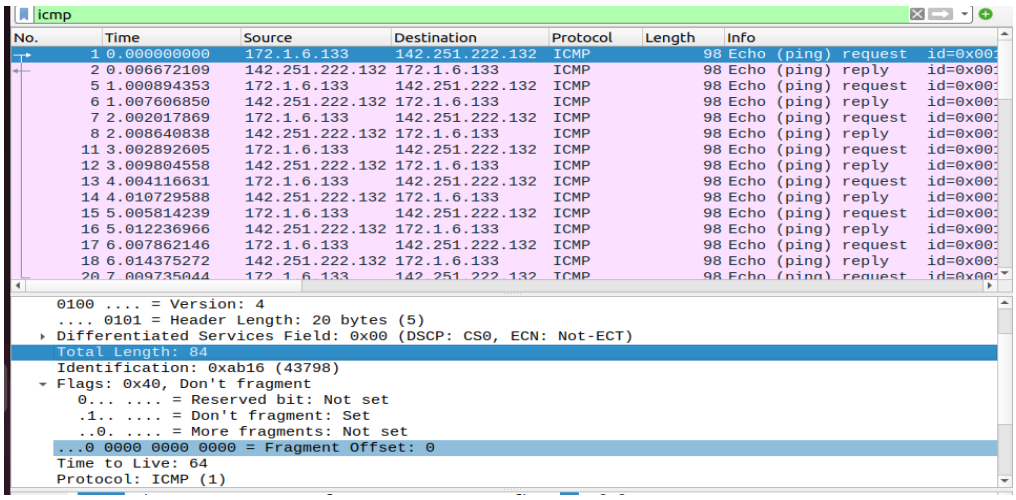
c) How many bytes are in the IP header? How many bytes are in the payload *of the IP datagram*? Explain how you determined the number of payload bytes.



d) Has this IP datagram been fragmented? Explain how you determined whether or not the datagram has been fragmented.



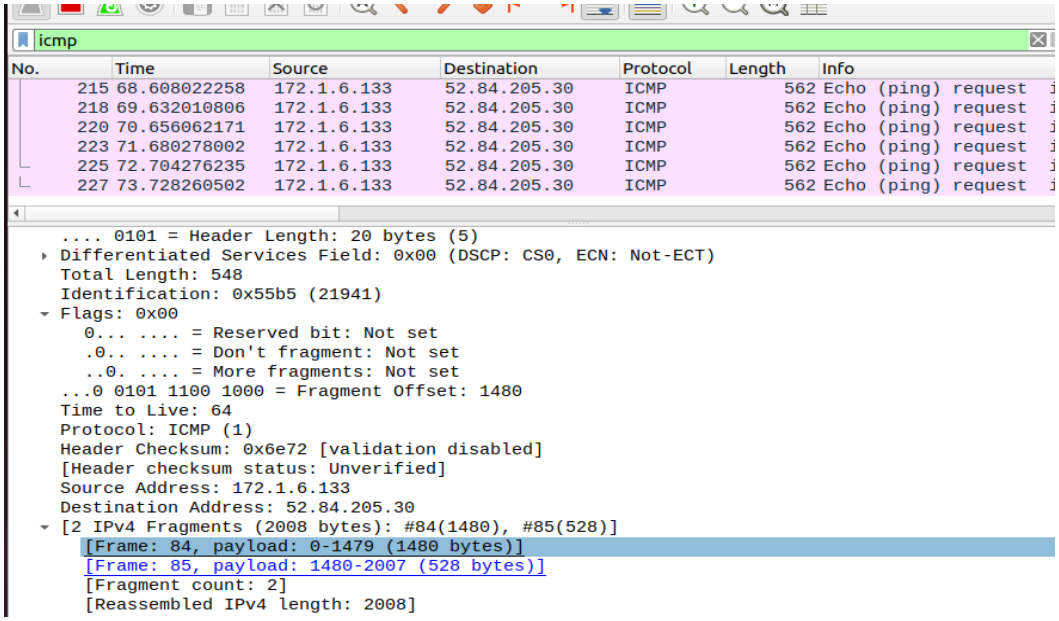
e) Which fields in the IP datagram *always* change from one datagram to the next within this series of ICMP messages sent by your computer?
Answer: Identification, Time to live and Header checksum always change.



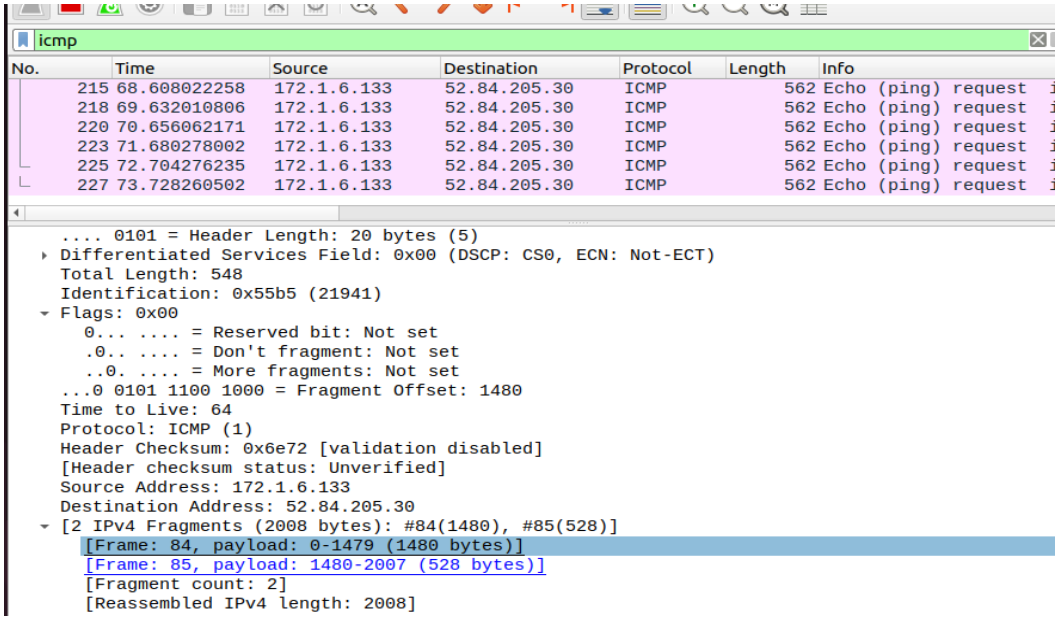
f) Which fields stay constant?
Answer: The fields that stay constant across the IP datagrams are:

- Version (since we are using IPv4 for all packets)
- header length (since these are ICMP packets)
- source IP (since we are sending from the same source)
- destination IP (since we are sending to the same dest)
- Differentiated Services (since all packets are ICMP they use the same Type of Service class)
- Upper Layer Protocol (since these are ICMP packets)

g) Find the first ICMP Echo Request message that was sent by your computer after you changed the *Packet Size* to 2000 (Use command *ping –s 2000 www.msrit.edu* to change the MTU of the packet). Has that message been fragmented across more than one IP datagram



h) Write down the first fragment of the fragmented IP datagram. What information in the IP header indicates that the datagram been fragmented? What information in the IP header indicates whether this is the first fragment versus a latter fragment? How long is this IP datagram?



i) What information in the IP header indicates that this is not the first datagram fragment? Are there more fragments? How can you tell? What fields change in the IP headerbetween the first and second fragment?

Answer: We can tell that this is not the first fragment, since the fragment offset is 1480. It is the last fragment, since the more fragments flag is not set.

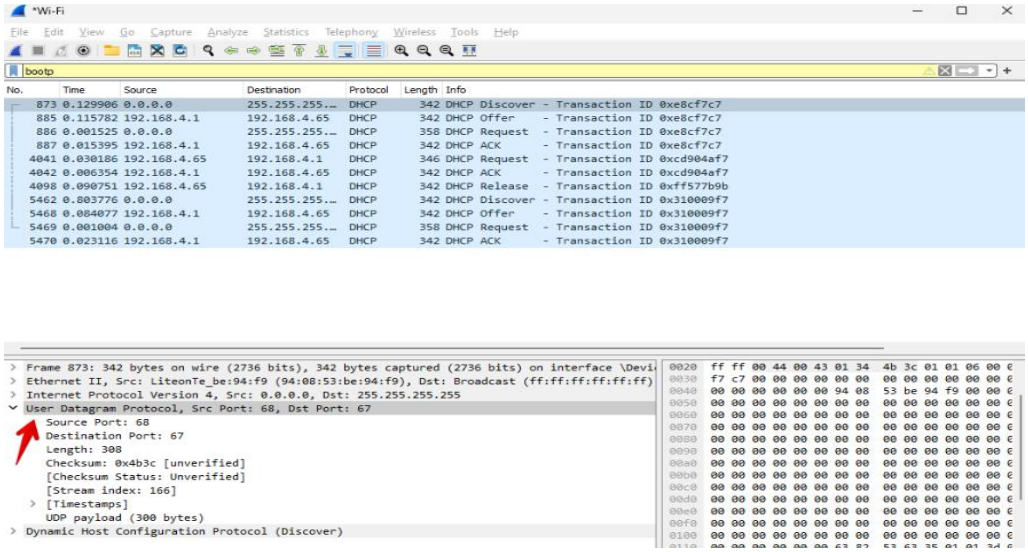
Lab Session 4

Problem Statement: Trace Dynamic Host Configuration Protocol using Wireshark.

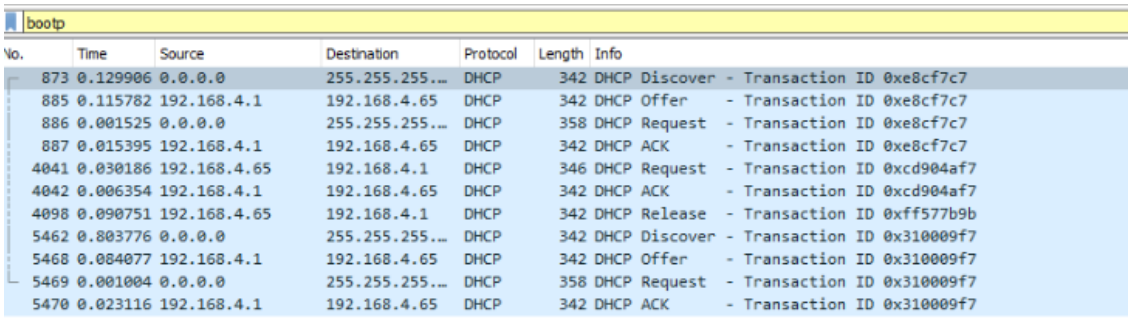
Exercise Question with Solution:

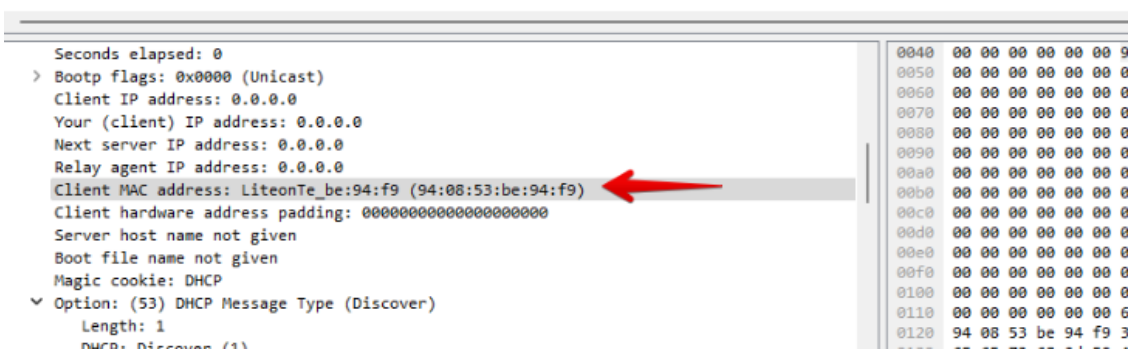
Use *dhclient –r* for releasing ip address, *dhclient <eth0>* for renewing ipaddress

a) Are DHCP messages sent over UDP or TCP?

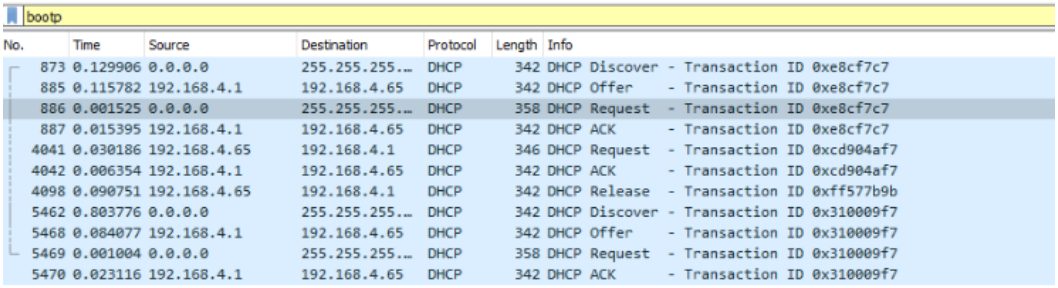


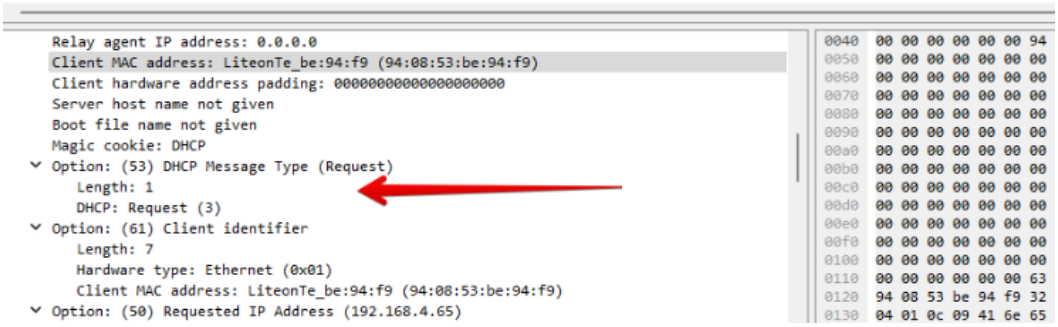
b) What is the link-layer (e.g., Ethernet) address of your host?





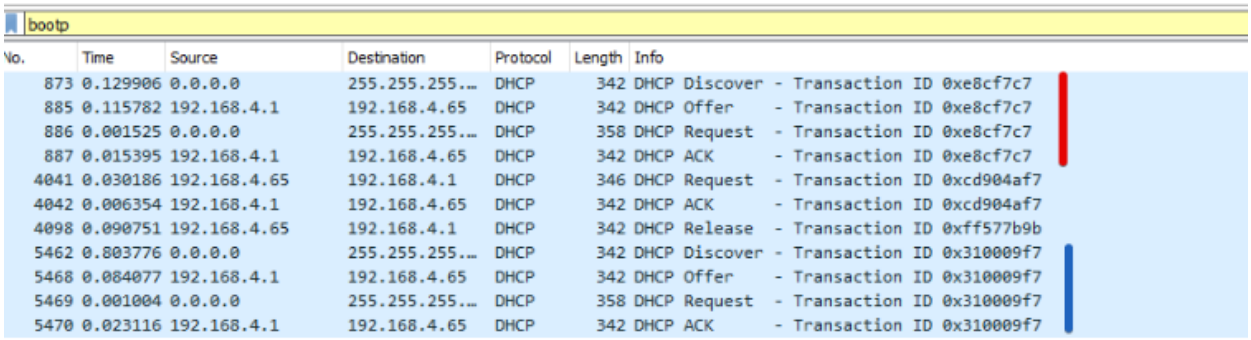
c) What values in the DHCP discover message differentiate this message from the DHCP request message?





d) What is the value of the Transaction-ID in each of the first four discover/Offer/Request/ACK) DHCP messages? What are the values of the Transaction-ID in the second set (Request/ACK) set of DHCP messages? What is the purpose of the Transaction-ID field?

- The client selects the **transaction ID** (xid) (often at random), and the server copies it in the answers. It serves a client-specific purpose, often enabling the client to identify the related dhcp answer to each request.



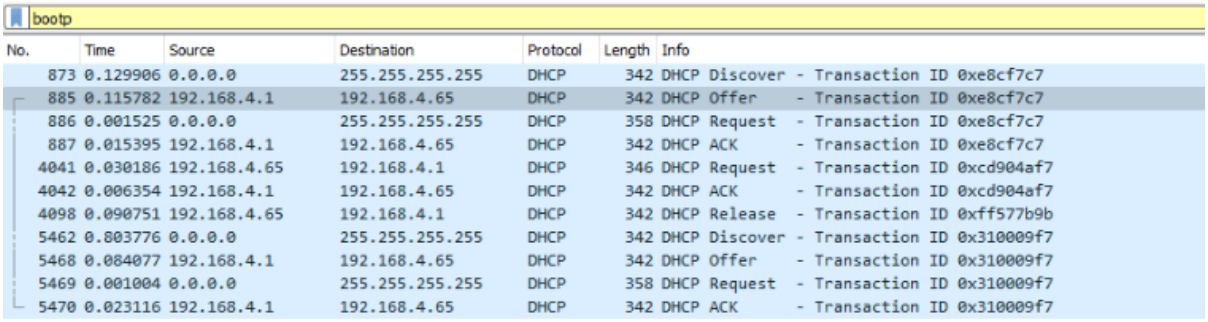
e) A host uses DHCP to obtain an IP address, among other things. But a host's IP address is not confirmed until the end of the four-message exchange! If the IP address is not set until the end of the four-message exchange, then what values are used in the IP datagrams in the four-message exchange? For each of the four DHCP messages (Discover/Offer/Request/ACK DHCP), indicate the source and destination IP addresses that are carried in the encapsulating IP datagram.

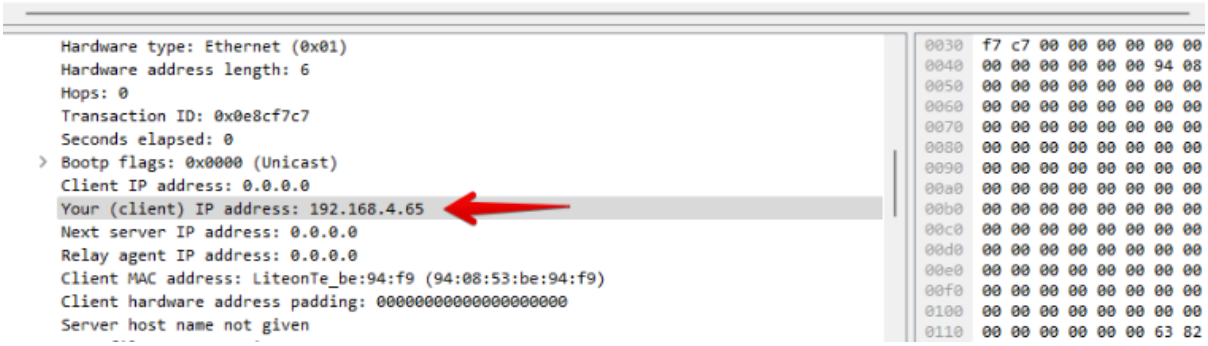
DHCP Message	Who Sends It?	Destination	Purpose
DHCP Discover	Client	Broadcast (255.255.255.255)	Find available DHCP servers
DHCP Offer	Server	Broadcast or Unicast	Offer an IP lease
DHCP Request	Client	Broadcast or Server	Request a specific IP
DHCP ACK	Server	Unicast or Broadcast	Confirm the lease

Type	SRC IP	DST IP
Discover	0.0.0.0	255.255.255.255
Offer	192.168.4.1	192.168.4.65
Request	0.0.0.0	255.255.255.255
ACK	192.168.4.1	192.168.4.65

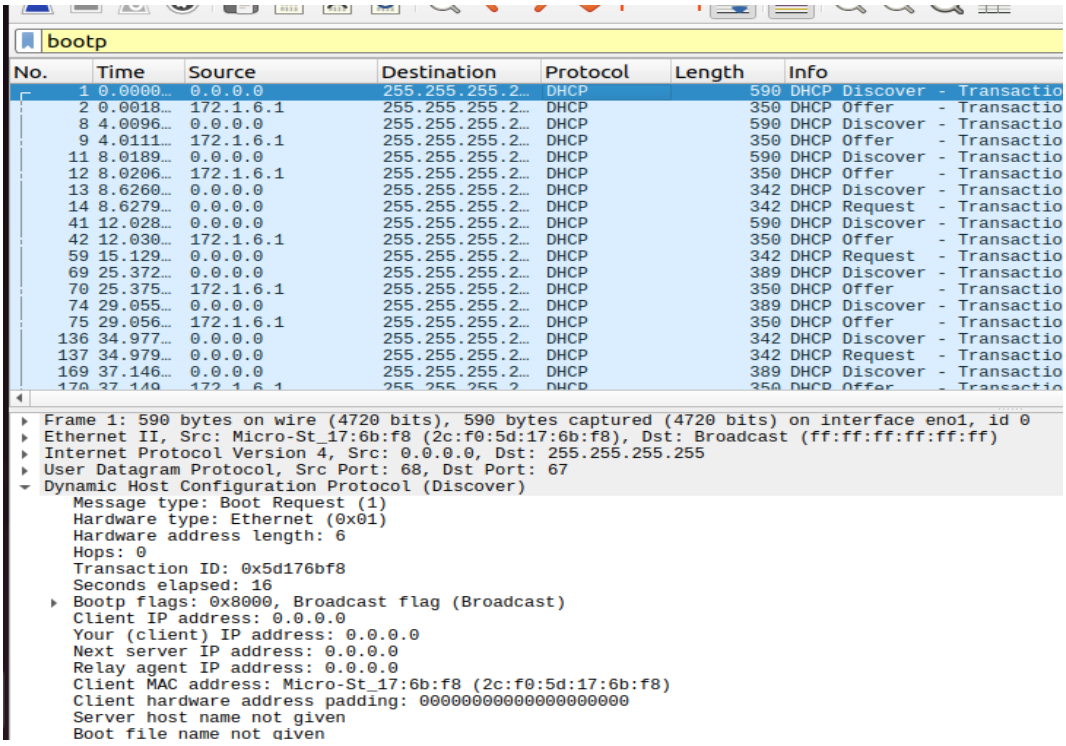
f) What is the IP address of your DHCP server?
IP address of your DHCP server - 192.168.4.65

g) What IP address is the DHCP server offering to your host in the DHCP Offer message? Indicate which DHCP message contains the offered DHCP address.
Answer: IP address offered to client – 192.168.4.65



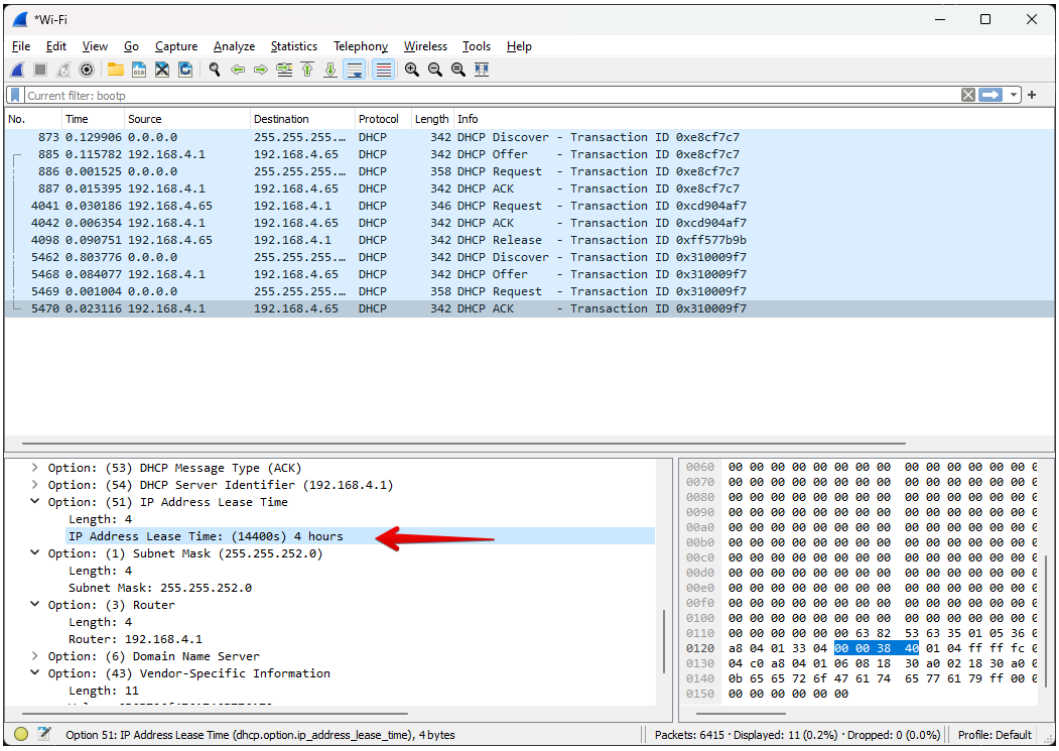


h) In the example screenshot in this assignment, there is no relay agent between the host and the DHCP server. What values in the trace indicate the absence of a relay agent? Is there a relay agent in your experiment? If so what is the IP address of the agent?
Answer: A value of 0:0:0:0 indicate there is no relay agent



i) Explain the purpose of the lease time. How long is the lease time in your experiment?

Lease time refers to the duration for which a network devices as assigned a specific IP address by a DHCP server.
IP Address -172.1.6.1



j) What is the purpose of the DHCP release message? Does the DHCP server issue an acknowledgment of receipt of the client's DHCP request?

- DHCP Release Message is the request to release the IP back to the DHCP Server.
- There is no ACK for this.
- Nothing happens if the release message is lost. The client will continue operation until its IP lease expires.

k) Explain the purpose of the router and subnet mask lines in the DHCP offer message.

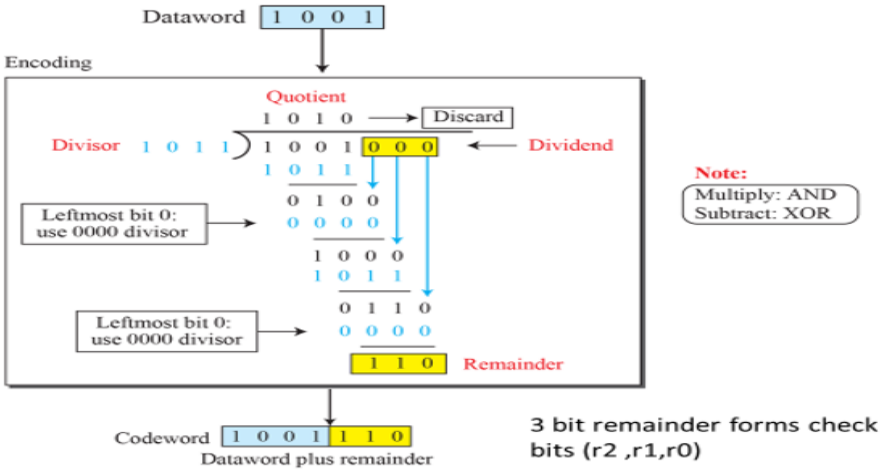
- The router line indicates where the client should send messages by default.
- The subnet mask line tells the client which subnet mask to use.

l) Clear the *bootp* filter from your Wireshark window. Were any ARP packets sent or received during the DHCP packet-exchange period? If so, explain the purpose of those ARP packets.

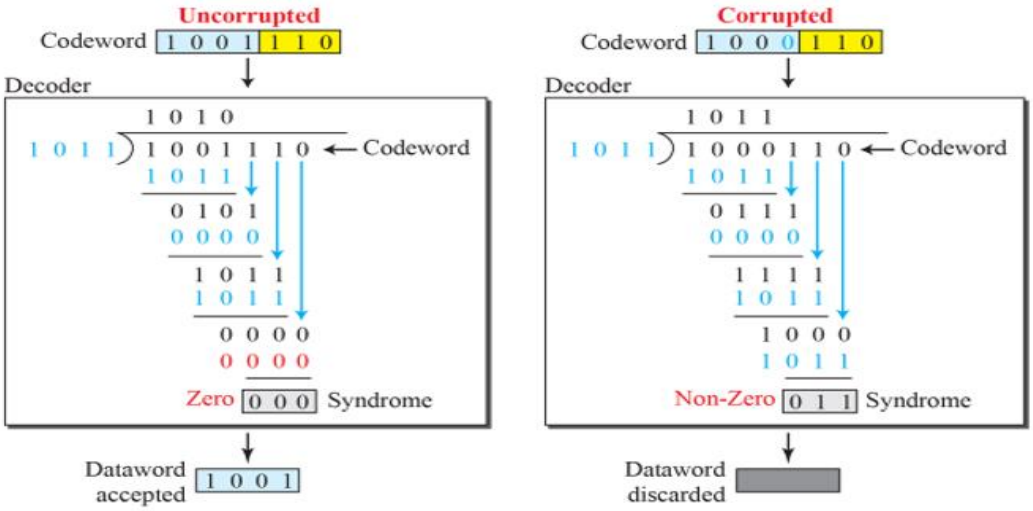
- Yes there numerous ARP messages sent out in my trace.
- An ARP request is sent when a device needs a MAC address associated with an IP address, and it does not have an entry for the IP address in its ARP table. This is used to map MACs to IPs in the local network

Lab Session 5

Write a program for error detection using CRC-CCITT(16-bits).



Division in CRC encoder



Division in the CRC decoder for two cases

Code:

```
import java.util.Scanner;

import java.io.*;

public class CRC1

{       public static void main(String args[])

        {       Scanner sc = new Scanner(System.in);                    //Input Data Stream
                System.out.print("Enter message bits: ");

                String message = sc.nextLine();

                System.out.print("Enter generator: ");
```

```
String generator = sc.nextLine();
int data[] = new int[message.length() + generator.length() - 1];
int divisor[] = new int[generator.length()];
for(int i=0;i<message.length();i++)
data[i] = Integer.parseInt(message.charAt(i)+"");
for(int i=0;i<generator.length();i++)
divisor[i] = Integer.parseInt(generator.charAt(i)+"");
        //Calculation of CRC
for(int i=0;i<message.length();i++)
{       if(data[i]==1)
        for(int j=0;j<divisor.length;j++)
        data[i+j] ^= divisor[j];
}                       //Display CRC
System.out.print("The checksum code is: ");
for(int i=0;i<message.length();i++)
data[i] = Integer.parseInt(message.charAt(i)+"");
for(int i=0;i<data.length;i++)
System.out.print(data[i]);
System.out.println();

//Check for input CRC code
System.out.print("Enter checksum code: ");
message = sc.nextLine();
System.out.print("Enter generator: ");
generator = sc.nextLine();
data = new int[message.length() + generator.length() - 1];
divisor = new int[generator.length()];
for(int i=0;i<message.length();i++)
data[i] = Integer.parseInt(message.charAt(i)+"");
for(int i=0;i<generator.length();i++)
divisor[i] = Integer.parseInt(generator.charAt(i)+"");

//Calculation of remainder
for(int i=0;i<message.length();i++)
```

```
        {       if(data[i]==1)
                for(int j=0;j<divisor.length;j++)
                data[i+j] ^= divisor[j];
                }

        //Display validity of data
        boolean valid = true;
        for(int i=0;i<data.length;i++)
                if(data[i]==1)
                {       valid = false;
                        break;
                }
                if(valid==true)
                        System.out.println("Data stream is valid");
                else
                        System.out.println("Data stream is invalid. CRC error occurred.");
        }
}
```

Output:

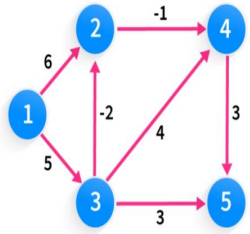
```
Enter message bits: 100010101
Enter generator: 1001
The checksum code is: 100010101011
Enter checksum code: 100010101011
Enter generator: 1001
Data stream is valid

Enter message bits: 1001010101
Enter generator: 1011
The checksum code is: 1001010101110
Enter checksum code: 1001010101111
Enter generator: 1011
Data stream is invalid. CRC error occurred.
```

Lab Session 6

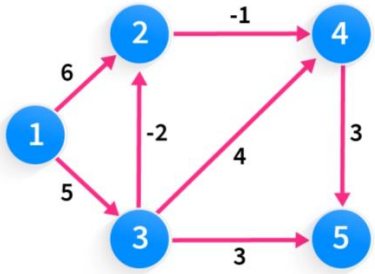
Write a program to find the shortest path between vertices using Bellman-Ford algorithm.

Solution:

- Single source shortest path algorithm.
- Dijkstras Algorithm :
    - Cannot handle negative weight.



Example :



N-1 times (4 times)

if(d[u] + c(u,v) < d[v])

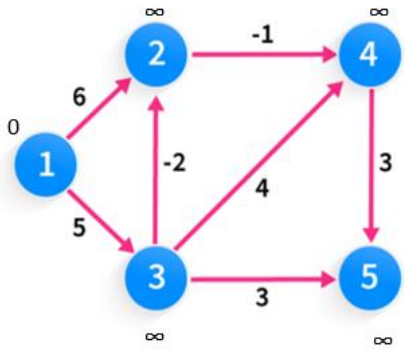
d[v] = d[u] + c(u,v)

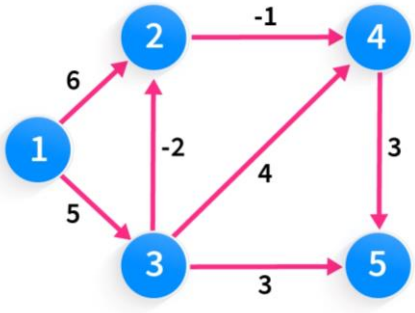
d[u] = source

c(u,v) = cost between u and v

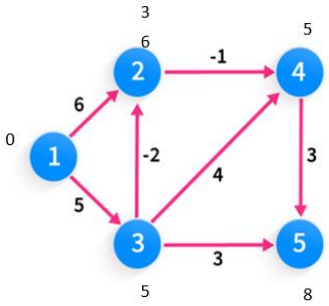
d[v] = destination

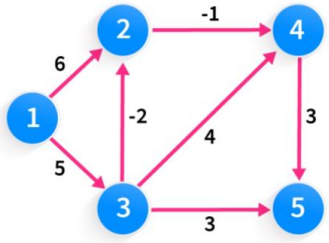
1st iteration



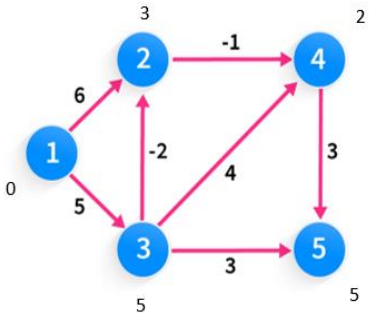


2nd iteration





3rd iteration



- Same in iteration 3 & 4.
- Loop = n-1 =4

Distance:

1 – 1 → 0

1 – 2 → 3

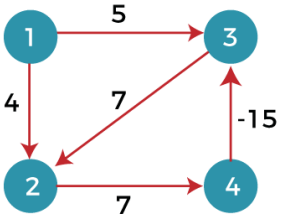
1 – 3 → 5

1 – 4 → 2

1 – 5 → 5

Disadvantage :

- It cannot give shortest path when the graph has negative cycle.
- Example : 7+7-15 = -1
- In code: value changes even after n-1 loop.



Code:

```
import java.util.Scanner;
public class ford
{       private int D[];
        private int num_ver;
```

When source and destination parameter are same, so to indicate which parameter to use.

```
        public static final int MAX_VALUE = 999;
        public ford(int num_ver)
        {       this.num_ver = num_ver;
                D = new int[num_ver + 1];
        }
public void BellmanFordEvaluation(int source, int A[][])
{   for (int node = 1; node <= num_ver; node++)
    {   D[node] = MAX_VALUE;
    }
    D[source] = 0;
    for(int node = 1; node <= num_ver - 1; node++)
    {   for (int sn = 1; sn <= num_ver; sn++)
        {   for (int dn = 1; dn <= num_ver; dn++)
            {   if (A[sn][dn] != MAX_VALUE)
                {   if (D[dn] > D[sn]+ A[sn][dn])
                    D[dn] = D[sn] + A[sn][dn];
                }
            }
        }
     }
for(int sn = 1; sn <= num_ver; sn++)
{   for(int dn = 1; dn <= num_ver; dn++)
    {   if(A[sn][dn] != MAX_VALUE)
        {   if (D[dn] > D[sn]+ A[sn][dn])
                System.out.println("The Graph contains negative edge cycle");
        }
     }
}

for (int vertex = 1; vertex <= num_ver; vertex++)
{   System.out.println("distance of source"+source+"to"+vertex+"is" + D[vertex]);
}
}
```

Check condition and update.

Checking for negative cycle.

Printing distance from source to destination.

```
public static void main(String[ ] args)
{    int num_ver = 0;
    int source;
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the number of vertices");
    num_ver = scanner.nextInt();
    int A[][] = new int[num_ver + 1][num_ver + 1];
    System.out.println("Enter the adjacency matrix");
    for (int sn = 1; sn <= num_ver; sn++)
    {    for (int dn = 1; dn <= num_ver; dn++)
        {    A[sn][dn] = scanner.nextInt();
            if (sn == dn)
            {     A[sn][dn] = 0;
                continue;
            }
```

//Scanning no of vertices = n = 5

	1	2	3	4	5
1	0	6	5	0	0
2	0	0	0	-1	0
3	0	-2	0	4	3
4	0	0	0	0	3
5	0	0	0	0	0

```
if (A[sn][dn] == 0)
            {    A[sn][dn] = MAX_VALUE;
            }
        }
}
System.out.println("Enter the source vertex");
source = scanner.nextInt();
ford b = new ford (num_ver);
b.BellmanFordEvaluation(source, A);
scanner.close();
}
}
```

Lab Session 7

Write a program for congestion control using leaky bucket algorithm.

Solution:

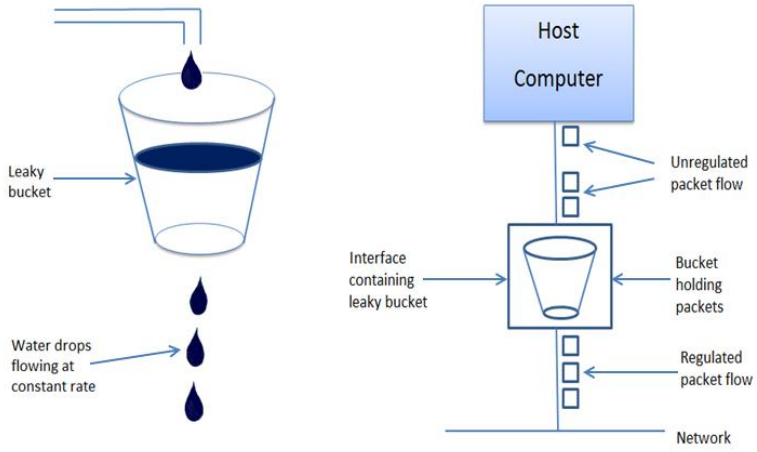


Fig: Leaky Bucket Algorithm

What is Leaky Bucket

- There is a bucket with a hole at the bottom.
- Water will be pouring into the bucket from the top.
- Water flows out at a constant rate irrespective of input of water into the bucket.

Lets apply the same concept to our network bucket.

- Tap → Host Computer
- Water flow → unregulated packet flow.
- Bucket → Interface

Result:

- Packet Size beyond capacity → Dropped
- Packet Size within the capacity →Accept

Example:

Initially the bucket is empty : Remaining = 0, Bucket Capacity = 4, Rate = 3

I	A[i]	Accept by Bucket	Sent	Remaining
1	2	2	2	0
2	4	4	3	1
3	1	1	2	0
4	5	Dropped	0	0
5	3	3	3	0

Code:

```
import java.util.Scanner;
import java.lang.*;
public class LeakyBucket
{	public static void main(String[] args)
	{	int i;
		int a[]=new int[20];
		int buck_rem=0,buck_cap=4,rate=3,sent,recv;
		Scanner in = new Scanner(System.in);
		System.out.println("Enter the number of packets");
		int n = in.nextInt();
		System.out.println("Enter the packets");
		for(i=1;i<=n;i++)
		a[i]= in.nextInt();
		System.out.println("Clock \t packet size \t accept \t sent \t remaining");
		 for(i=1;i<=n;i++)
		{	if(a[i]!=0)
			{	 if(buck_rem+a[i]>buck_cap)
				 recv=-1;
				 else
				 {	recv=a[i]; buck_rem+=a[i];
				}
			}
			else recv=0;
			if(buck_rem!=0)
			{	if(buck_rem<rate)
				{	sent=buck_rem; buck_rem=0;
				}
				else
				{	sent=rate;
					buck_rem=buck_rem-rate;
				}
			}
			else
```

```
                    sent=0;
                if(recv==-1)
                System.out.println(+i+ "\t\t" +a[i]+ "\t dropped \t" + sent +"\t" +buck_rem); else
                System.out.println(+i+ "\t\t" +a[i] +"\t\t" +recv +"\t" +sent + "\t" +buck_rem);
                }
        }
}
```

Cycle – 2

Lab Session 8

Using TCP/IP sockets, write a client – server program where the client send the file name and the server send back the contents of the requested file if present.

Solution:

Socket Programming:

- Socket programming is a way of connecting two nodes on a network to communicate with each other.
- One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection.

How to use sockets?

- Set up a socket.
- Send and Receive the packets.
- Close the socket.

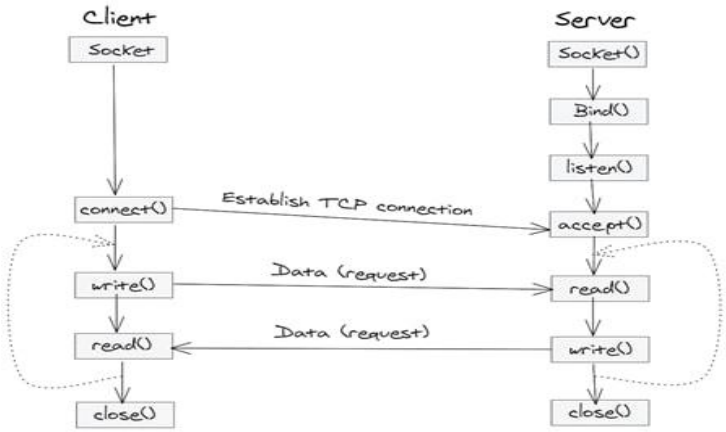
Typical Server Program Using TCP:

- Set up a Socket (Prepare to communicate)
- Wait to hear from a client
- Send and receive packets (Exchange data with the client over the new socket s_new)

Typical Client Program Using TCP:

- Set up a Socket (Prepare to communicate)
  - Create a socket
  - Determine server IP address and port number
  - Initiate the connection to the server
- Send and receive packets (Exchange data with the server)
  - Write data (i.e., request) to the socket
  - Read data (i.e., response) from the socket
  - Do stuff with the data (e.g., display a Web page)
- Close the socket.

STATE DIAGRAM:



Code: Client.java:

```
import java.net.*;
import java.io.*;
public class TCPC
{       public static void main(String[] args) throws Exception
        {       Socket sock=new Socket("127.0.01",4000);
                System.out.println("Enter the filename");
                BufferedReader keyRead=new BufferedReader(new InputStreamReader(System.in));
                String fname=keyRead.readLine();
                OutputStream ostream=sock.getOutputStream();
                PrintWriter pwrite=new PrintWriter(ostream,true);
                pwrite.println(fname);
                InputStream istream=sock.getInputStream();
                BufferedReader socketRead=new BufferedReader(new InputStreamReader(istream));
                String str;
                while((str=socketRead.readLine())!=null)
                {       System.out.println(str);
                }
                pwrite.close();
                socketRead.close();
                keyRead.close();
        }
}
```

Code: Server.java

```
import java.net.*;
import java.io.*;
public class TCPS
{       public static void main(String[] args) throws Exception
        {       ServerSocket sersock=new ServerSocket(4000);
                System.out.println("Server ready for connection");
                Socket sock=sersock.accept();
                System.out.println("Connection Is successful and waiting for chatting");
                InputStream istream=sock.getInputStream();
```

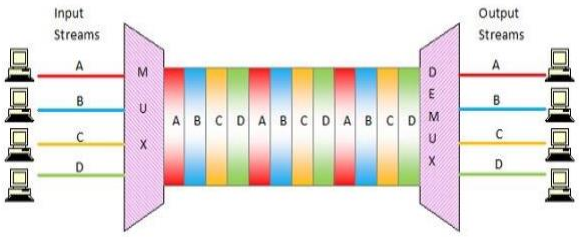
```
                BufferedReader fileRead=new BufferedReader(new InputStreamReader(istream));
                String fname=fileRead.readLine();
                BufferedReader ContentRead=new BufferedReader(new FileReader(fname));
                OutputStream ostream=sock.getOutputStream();
                PrintWriter pwrite=new PrintWriter(ostream,true);
                String str;
                while((str=ContentRead.readLine())!=null){
                pwrite.println(str);
                }
                sock.close();
                sersock.close();
                pwrite.close();
                fileRead.close();
                ContentRead.close();
        }
}
```

Lab Session 9

Write a program for Time Division Multiplexing Simulator. Show how the time division multiplexing technique works.

Solution:

- In TDM, the data flow of each input stream is divided into units. One unit may be 1 bit, 1 byte, or a block of few bytes.
- Each input unit is allotted an input time slot. One input unit corresponds to one output unit and is allotted an output time slot.
- During transmission, one unit of each of the input streams is allotted one-time slot, periodically, in a sequence, on a rotational basis. This system is popularly called round-robin system. Example Consider a system having four input streams, A, B, C and D.
- Each of the data streams is divided into units which are allocated time slots in the round – robin manner.
- Hence, the time slot 1 is allotted to A, slot 2 is allotted to B, slot 3 is allotted to C, slot 4 is allotted to D, slot 5 is allocated to A again, and this goes on till the data in all the streams are transmitted.



Code:

```
import java.util.Scanner;

public class TDM

{       public static void main(String args[])

        {       int n,i,qt,count=0,temp,sq=0,bt[],wt[],tat[],rem_bt[];

                float awt=0,atat=0;

                bt = new int[10];

                wt = new int[10];

                tat = new int[10];

                rem_bt = new int[10];

                Scanner s=new Scanner(System.in);

                System.out.print("Enter the number of stations (maximum 10) = ");

                n = s.nextInt();

                System.out.print("Enter the processing time for each channel\n");

                for (i=0;i<n;i++)
```

```
        {       System.out.print("S"+i+" = ");                  //stations Input
                bt[i] = s.nextInt();
                rem_bt[i] = bt[i];
        }
        System.out.print("Enter the frame size: ");      // Frame size for each station
        qt = s.nextInt();
        while(true)
        {       for (i=0,count=0;i++)
                {       temp = qt;
                        if(rem_bt[i] == 0)
                        {       count++;
                                continue;
                        }
                        if(rem_bt[i]>qt)
                                rem_bt[i]= rem_bt[i] - qt;
                        else if(rem_bt[i]>=0)
                        {       temp = rem_bt[i];
                                rem_bt[i] = 0;
                        }
                        sq = sq + temp;
                        tat[i] = sq;
                }
                if(n == count)
                break;
        }
System.out.print("---------------------------------------------------------------------------");
System.out.print("\nStation\t Processing Time\t Completion Time\t Waiting Time\n");
System.out.print("---------------------------------------------------------------------------");
for(i=0;i<n;i++)
{       wt[i]=tat[i]-bt[i];
        awt=awt+wt[i];
        atat=atat+tat[i];
        System.out.print("\n \t"+(i+1)+"\t \t"+bt[i]+"\t\t "+tat[i]+"\t\t\t "+wt[i]+"\n");
}
```

```
        }
}
```

OUTPUT:

Enter the number of stations (maximum 10) = 4

Enter the processing time for each channel

S0 = 4

S1 = 5

S2 = 3

S3 = 2

Enter the frame size: 2

Station	Processing Time	Completion Time	Waiting Time
1	4	10	6
2	5	14	9
3	3	13	10
4	2	8	6

Network Simulator – 3

NS-3 (Network Simulator 3) is a popular, open-source discrete-event network simulator used mainly for research and educational purposes. It's a **simulator** designed to model how computer networks behave. You can simulate **wired and wireless networks**, internet protocols, and network topologies without needing real hardware. Mainly used by researchers, network engineers, and students to test new protocols, network algorithms, or study network performance. Written mostly in **C++**, with Python bindings to make simulations easier to script.

Wireshark captures and analyzes *real* network traffic live or from files.**NS-3** simulates a network *in software* for experimentation and research without real hardware.

Installing NS-3 on Ubuntu/Linux

Step 1: Update packages and install prerequisites

Open a terminal and run:

- sudo apt update
- sudo apt install gcc g++ python3 python3-dev python3-setuptools git mercurial qt5-default qtcreator cmake libc6-dev libc6 libc6-dev python3-pygraphviz python3-pip graphviz pkg-config

Step 2: Download NS-3 source code

Go to your home directory or preferred folder:

- **cd ~**

Download the latest stable NS-3 release (replace version number as needed):

- **wget https://www.nsnam.org/releases/ns-allinone-3.39.tar.bz2**
- **tar xjf ns-allinone-3.39.tar.bz2**
- **cd ns-allinone-3.39**

Step 3: **Build and install NS-3**

Run the build script:

- **./build.py --enable-examples --enable-tests**

Step 4: **Set environment variables**

Once the build is done, go to the ns-3 directory:

- **cd ns-3.39**

You can test it by running: **./waf --run hello-simulator**

NS3

Lab Session 10

Problem Statement : Design and simulate a wired network with duplex links between ‘n’ nodes with CDR over UDP. Set the queue size vary the bandwidth and find the number of packets dropped.

File Path:

- **~/ns-allinone-3.xx/ns-3.xx/Example/traffic-control/**
- You can find examples like traffic-control-simple.cc.
- Copy and Paste it in Scratch.
- Open Scratch and Rename the File.
- Open the Renamed File in Scratch

Code:

```
#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/traffic-control-module.h"

using namespace ns3;
NS_LOG_COMPONENT_DEFINE("TrafficControlExample");

int main(int argc, char* argv[])
{
  double simulationTime = 10; // seconds

  std::string transportProt = "Udp";
  std::string socketType;

  CommandLine cmd(__FILE__);
  cmd.AddValue("transportProt", "Transport protocol to use: Tcp, Udp", transportProt);
  cmd.Parse(argc, argv);

  if (transportProt == "Tcp")
  {
    socketType = "ns3::TcpSocketFactory";
  }
  else
  {
    socketType = "ns3::UdpSocketFactory";
  }

  NodeContainer nodes;
```

```
nodes.Create(3);


PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
pointToPoint.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("100p"));

NetDeviceContainer devices;
devices = pointToPoint.Install(nodes.Get(0), nodes.Get(1));

NetDeviceContainer devices1;
devices1 = pointToPoint.Install(nodes.Get(1), nodes.Get(2));

InternetStackHelper stack;
stack.Install(nodes);


Ipv4AddressHelper address;

address.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign(devices);

address.SetBase("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces1 = address.Assign(devices1);


Ipv4GlobalRoutingHelper::PopulateRoutingTables();    // Routing table from third.cc


// Flow
//Define the port number 7 that the PacketSink will listen on
uint16_t port = 7;
// binds to any available IP address (0.0.0.0) and uses the previously defined port (port 7).
Address localAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
//specifying socketType and the local address (IP and port) to bind the PacketSink application.
PacketSinkHelper packetSinkHelper(socketType, localAddress);
// Install the PacketSink application on node 2
ApplicationContainer sinkApp = packetSinkHelper.Install(nodes.Get(2));

sinkApp.Start(Seconds(0.0));
sinkApp.Stop(Seconds(simulationTime + 0.1));

uint32_t payloadSize = 1448;
Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(payloadSize));

OnOffHelper onoff(socketType, Ipv4Address::GetAny());
onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
onoff.SetAttribute("PacketSize", UintegerValue(payloadSize));
onoff.SetAttribute("DataRate", StringValue("50Mbps")); // bit/s
```

```
    ApplicationContainer apps;

    InetSocketAddress rmt(interfaces1.GetAddress(1), port);
    rmt.SetTos(0xb8);
    AddressValue remoteAddress(rmt);
    onoff.SetAttribute("Remote", remoteAddress);

    apps.Add(onoff.Install(nodes.Get(0)));
    apps.Start(Seconds(1.0));
    apps.Stop(Seconds(simulationTime + 0.1));

    FlowMonitorHelper flowmon;
    Ptr<FlowMonitor> monitor = flowmon.InstallAll();


    Simulator::Stop(Seconds(simulationTime + 5));
    Simulator::Run();

    Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
    std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();

    std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
    std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets << " / " << stats[1].txBytes
          << std::endl;
    std::cout << "  Offered Load: "
          << stats[1].txBytes * 8.0 /
              (stats[1].timeLastTxPacket.GetSeconds() -
               stats[1].timeFirstTxPacket.GetSeconds()) /
              1000000
          << " Mbps" << std::endl;
    std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets << " / " << stats[1].rxBytes
          << std::endl;

    //Delete from here to till Destroy

    Simulator::Destroy();

// Delete Till return 0

    return 0;
}
```

Lab Session 11

Problem Statement: Design and simulate a four node point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP agent between n1-n3. Apply relevant applications over TCP and UDP agents by changing the parameters and determine the number of packets sent by TCP/UDP.

File Path:

- **~/ns-allinone-3.xx/ns-3.xx/Example/traffic-control/**
- You can find examples like traffic-control-simple.cc.
- Copy and Paste it in Scratch.
- Open Scratch and Rename the File.
- Open the Renamed File in Scratch

Code:

```
#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/traffic-control-module.h"

// Network topology
//
//      10.1.1.0      10.1.2.0
// n0 ---------------- n2 ------------- n3
//            point-to-point


//      10.1.1.0                  10.1.3.0
// n1 ---------------- n2 ------------- n3
//                      point-to-point
//


using namespace ns3;
NS_LOG_COMPONENT_DEFINE("TrafficControlExample");

int
main(int argc, char* argv[])
{
   double simulationTime = 10; // seconds
   //std::string transportProt = "Tcp";
  // std::string socketType;
```

```
  CommandLine cmd(__FILE__);
 // cmd.AddValue("transportProt", "Transport protocol to use: Tcp, Udp", transportProt);
  cmd.Parse(argc, argv);

 /* if (transportProt == "Tcp")
  {
     socketType = "ns3::TcpSocketFactory";
  }
  else
  {
     socketType = "ns3::UdpSocketFactory";
  }*/

  NodeContainer nodes;
  nodes.Create(4);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
  pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
  //pointToPoint.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("1p"));

  NetDeviceContainer devices;
  devices = pointToPoint.Install(nodes.Get(0), nodes.Get(1));

  NetDeviceContainer devices1;
  devices1 = pointToPoint.Install(nodes.Get(1), nodes.Get(2));

  NetDeviceContainer devices2;
  devices2 = pointToPoint.Install(nodes.Get(3), nodes.Get(1));

  NetDeviceContainer devices3;
  devices3 = pointToPoint.Install(nodes.Get(1), nodes.Get(2));

  InternetStackHelper stack;
  stack.Install(nodes);

  /*TrafficControlHelper tch;
  tch.SetRootQueueDisc("ns3::RedQueueDisc");
  QueueDiscContainer qdiscs = tch.Install(devices);

  Ptr<QueueDisc> q = qdiscs.Get(1);
  q->TraceConnectWithoutContext("PacketsInQueue", MakeCallback(&TcPacketsInQueueTrace));
  Config::ConnectWithoutContext(
     "/NodeList/1/$ns3::TrafficControlLayer/RootQueueDiscList/0/SojournTime",
     MakeCallback(&SojournTimeTrace));

  Ptr<NetDevice> nd = devices.Get(1);
  Ptr<PointToPointNetDevice> ptpnd = DynamicCast<PointToPointNetDevice>(nd);
  Ptr<Queue<Packet>> queue = ptpnd->GetQueue();
  queue->TraceConnectWithoutContext("PacketsInQueue",
MakeCallback(&DevicePacketsInQueueTrace));*/
```

```
    Ipv4AddressHelper address;
    address.SetBase("10.1.1.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces = address.Assign(devices);

    address.SetBase("10.1.2.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces1 = address.Assign(devices1);

    Ipv4AddressHelper address1;
    address1.SetBase("10.1.3.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces2 = address1.Assign(devices2);

    address1.SetBase("10.1.4.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces3 = address1.Assign(devices3);

Ipv4GlobalRoutingHelper::PopulateRoutingTables();

    // Flow
    uint16_t port = 7;
    Address localAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
    PacketSinkHelper packetSinkHelper("ns3::TcpSocketFactory", localAddress);
    ApplicationContainer sinkApp = packetSinkHelper.Install(nodes.Get(3));

    sinkApp.Start(Seconds(0.0));
    sinkApp.Stop(Seconds(simulationTime + 0.1));

    uint16_t port1 = 9;
    Address localAddress1(InetSocketAddress(Ipv4Address::GetAny(), port1));
    PacketSinkHelper packetSinkHelper1("ns3::UdpSocketFactory", localAddress1);
    ApplicationContainer sinkApp1 = packetSinkHelper1.Install(nodes.Get(3));

    sinkApp1.Start(Seconds(0.0));
    sinkApp1.Stop(Seconds(simulationTime + 0.1));

    uint32_t payloadSize = 1448;
    Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(payloadSize));

    OnOffHelper onoff("ns3::TcpSocketFactory", Ipv4Address::GetAny());
    onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
    onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
    onoff.SetAttribute("PacketSize", UintegerValue(payloadSize));
    onoff.SetAttribute("DataRate", StringValue("50Mbps")); // bit/s
    ApplicationContainer apps;

    OnOffHelper onoff1("ns3::UdpSocketFactory", Ipv4Address::GetAny());
    onoff1.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
    onoff1.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
    onoff1.SetAttribute("PacketSize", UintegerValue(payloadSize));
    onoff1.SetAttribute("DataRate", StringValue("50Mbps")); // bit/s
    ApplicationContainer apps1;

    InetSocketAddress rmt(interfaces1.GetAddress(2), port);
```

```
rmt.SetTos(0xb8);
AddressValue remoteAddress(rmt);
onoff.SetAttribute("Remote", remoteAddress);
apps.Add(onoff.Install(nodes.Get(0)));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(simulationTime + 0.1));

InetSocketAddress rmt1(interfaces3.GetAddress(1), port1);
rmt.SetTos(0xb8);
AddressValue remoteAddress1(rmt1);
onoff1.SetAttribute("Remote", remoteAddress1);
apps1.Add(onoff1.Install(nodes.Get(1)));
apps1.Start(Seconds(1.5));
apps1.Stop(Seconds(simulationTime + 0.1));

FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();

Simulator::Stop(Seconds(simulationTime + 5));
Simulator::Run();

 /*Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets << " / " << stats[1].txBytes
        << std::endl;
std::cout << "  Offered Load: "
        << stats[1].txBytes * 8.0 /
             (stats[1].timeLastTxPacket.GetSeconds() -
              stats[1].timeFirstTxPacket.GetSeconds()) /
             1000000
        << " Mbps" << std::endl;
std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets << " / " << stats[1].rxBytes
        << std::endl;*/
/*  uint32_t packetsDroppedByQueueDisc = 0;
uint64_t bytesDroppedByQueueDisc = 0;
if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE_DISC)
{
   packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
   bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
}
std::cout << "  Packets/Bytes Dropped by Queue Disc:   " << packetsDroppedByQueueDisc << " / "
        << bytesDroppedByQueueDisc << std::endl;
uint32_t packetsDroppedByNetDevice = 0;
uint64_t bytesDroppedByNetDevice = 0;
if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE)
{
   packetsDroppedByNetDevice = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE];
   bytesDroppedByNetDevice = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE];
}
std::cout << "  Packets/Bytes Dropped by NetDevice:   " << packetsDroppedByNetDevice << " / "
```

```
        << bytesDroppedByNetDevice << std::endl;
  std::cout << "  Throughput: "
        << stats[1].rxBytes * 8.0 /
            (stats[1].timeLastRxPacket.GetSeconds() -
             stats[1].timeFirstRxPacket.GetSeconds()) /
            1000000
        << " Mbps" << std::endl;
  std::cout << "  Mean delay:   " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
        << std::endl;
  std::cout << "  Mean jitter:   " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
        << std::endl;
  auto dscpVec = classifier->GetDscpCounts(1);
  for (auto p : dscpVec)
  {
    std::cout << "  DSCP value:   0x" << std::hex << static_cast<uint32_t>(p.first) << std::dec
          << "  count:   " << p.second << std::endl;
  } */


 monitor->CheckForLostPackets ();
 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
 std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

 for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iter = stats.begin (); iter != stats.end
(); ++iter)
   {
 Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);
    NS_LOG_UNCOND("Flow ID: " << iter->first << " Src Addr " << t.sourceAddress << " Dst Addr " <<
t.destinationAddress);
    NS_LOG_UNCOND("Tx Packets = " << iter->second.txPackets);
    NS_LOG_UNCOND("Rx Packets = " << iter->second.rxPackets);
    NS_LOG_UNCOND("lostPackets Packets = " << iter->second.lostPackets);
    NS_LOG_UNCOND("Throughput: " << iter->second.rxBytes * 8.0 / (iter-
>second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSeconds()) / 1024  << "
Kbps");
    NS_LOG_UNCOND("----------------------" );
   }

  Simulator::Destroy();

  /* std::cout << std::endl << "*** Application statistics ***" << std::endl;
  double thr = 0;
  uint64_t totalPacketsThr = DynamicCast<PacketSink>(sinkApp.Get(0))->GetTotalRx();
  thr = totalPacketsThr * 8 / (simulationTime * 1000000.0); // Mbit/s
  std::cout << "  Rx Bytes: " << totalPacketsThr << std::endl;
  std::cout << "  Average Goodput: " << thr << " Mbit/s" << std::endl;
  std::cout << std::endl << "*** TC Layer statistics ***" << std::endl;
  std::cout << q->GetStats() << std::endl; */
  return 0;
}
```

Lab Session 12

Problem Statement: Design and simulate simple Extended Service Set with transmitting nodes in wireless LAN and determine the performance with respect to transmission of Packets.

File Path:

- **~/ns-allinone-3.xx/ns-3.xx/Example/tutorial**
- You can find examples like third.cc.
- Copy and Paste it in Scratch.
- Open Scratch and Rename the File.
- Open the Renamed File in Scratch

Code:

```
#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/ssid.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/traffic-control-module.h"
// 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

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("ThirdScriptExample");

int
main(int argc, char* argv[])
{
   bool verbose = true;
   uint32_t nCsma = 3;
   uint32_t nWifi = 3;
   bool tracing = false;
```

```
double simulationTime = 10.0;

CommandLine cmd(__FILE__);
cmd.AddValue("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
cmd.AddValue("nWifi", "Number of wifi STA devices", nWifi);
cmd.AddValue("verbose", "Tell echo applications to log if true", verbose);
cmd.AddValue("tracing", "Enable pcap tracing", tracing);

cmd.Parse(argc, argv);

// The underlying restriction of 18 is due to the grid position
// allocator's configuration; the grid layout will exceed the
// bounding box if more than 18 nodes are provided.
if (nWifi > 18)
{
   std::cout << "nWifi should be 18 or less; otherwise grid layout exceeds the bounding box"
         << std::endl;
   return 1;
}

if (verbose)
{
   LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
   LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
}

NodeContainer p2pNodes;
p2pNodes.Create(2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install(p2pNodes);

NodeContainer csmaNodes;
csmaNodes.Add(p2pNodes.Get(1));
csmaNodes.Create(nCsma);

CsmaHelper csma;
csma.SetChannelAttribute("DataRate", StringValue("100Mbps"));
csma.SetChannelAttribute("Delay", TimeValue(NanoSeconds(6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install(csmaNodes);

NodeContainer wifiStaNodes;
wifiStaNodes.Create(nWifi);
NodeContainer wifiApNode = p2pNodes.Get(0);
```

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
YansWifiPhyHelper phy;
phy.SetChannel(channel.Create());

WifiMacHelper mac;
Ssid ssid = Ssid("ns-3-ssid");

WifiHelper wifi;

NetDeviceContainer staDevices;
mac.SetType("ns3::StaWifiMac", "Ssid", SsidValue(ssid), "ActiveProbing", BooleanValue(false));
staDevices = wifi.Install(phy, mac, wifiStaNodes);

NetDeviceContainer apDevices;
mac.SetType("ns3::ApWifiMac", "Ssid", SsidValue(ssid));
apDevices = wifi.Install(phy, mac, wifiApNode);

MobilityHelper mobility;

mobility.SetPositionAllocator("ns3::GridPositionAllocator",
                      "MinX",
                      DoubleValue(0.0),
                      "MinY",
                      DoubleValue(0.0),
                      "DeltaX",
                      DoubleValue(5.0),
                      "DeltaY",
                      DoubleValue(10.0),
                      "GridWidth",
                      UintegerValue(3),
                      "LayoutType",
                      StringValue("RowFirst"));

mobility.SetMobilityModel("ns3::RandomWalk2dMobilityModel",
                    "Bounds",
                    RectangleValue(Rectangle(-50, 50, -50, 50)));
mobility.Install(wifiStaNodes);

mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(wifiApNode);

InternetStackHelper stack;
stack.Install(csmaNodes);
stack.Install(wifiApNode);
stack.Install(wifiStaNodes);

Ipv4AddressHelper address;

address.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign(p2pDevices);
```

```
address.SetBase("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign(csmaDevices);

address.SetBase("10.1.3.0", "255.255.255.0");
address.Assign(staDevices);
address.Assign(apDevices);


Ipv4GlobalRoutingHelper::PopulateRoutingTables();

//Remove Echo application and Include onoff application

// Flow
uint16_t port = 7;
Address localAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
PacketSinkHelper packetSinkHelper("ns3::UdpSocketFactory", localAddress);
ApplicationContainer sinkApp = packetSinkHelper.Install(csmaNodes.Get(2));

sinkApp.Start(Seconds(0.0));
sinkApp.Stop(Seconds(simulationTime + 0.1));

uint32_t payloadSize = 1448;
Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(payloadSize));

OnOffHelper onoff("ns3::UdpSocketFactory", Ipv4Address::GetAny());
onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
onoff.SetAttribute("PacketSize", UintegerValue(payloadSize));
onoff.SetAttribute("DataRate", StringValue("50Mbps")); // bit/s
ApplicationContainer apps;

InetSocketAddress rmt(csmaInterfaces.GetAddress(nCsma), port);
rmt.SetTos(0xb8);
AddressValue remoteAddress(rmt);
onoff.SetAttribute("Remote", remoteAddress);
apps.Add(onoff.Install(wifiStaNodes.Get(0)));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(simulationTime + 0.1));

FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();

Simulator::Stop(Seconds(simulationTime + 5));
Simulator::Run();

Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets << " / " << stats[1].txBytes
        << std::endl;
```

```
std::cout << "  Offered Load: "
        << stats[1].txBytes * 8.0 /
             (stats[1].timeLastTxPacket.GetSeconds() -
              stats[1].timeFirstTxPacket.GetSeconds()) /
             1000000
        << " Mbps" << std::endl;
std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets << " / " << stats[1].rxBytes
        << std::endl;
uint32_t packetsDroppedByQueueDisc = 0;
uint64_t bytesDroppedByQueueDisc = 0;
if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE_DISC)
{
   packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
   bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
}
std::cout << "  Packets/Bytes Dropped by Queue Disc:   " << packetsDroppedByQueueDisc << " / "
        << bytesDroppedByQueueDisc << std::endl;
uint32_t packetsDroppedByNetDevice = 0;
uint64_t bytesDroppedByNetDevice = 0;
if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE)
{
   packetsDroppedByNetDevice = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE];
   bytesDroppedByNetDevice = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE];
}
std::cout << "  Packets/Bytes Dropped by NetDevice:   " << packetsDroppedByNetDevice << " / "
        << bytesDroppedByNetDevice << std::endl;
std::cout << "  Throughput: "
        << stats[1].rxBytes * 8.0 /
             (stats[1].timeLastRxPacket.GetSeconds() -
              stats[1].timeFirstRxPacket.GetSeconds()) /
             1000000
        << " Mbps" << std::endl;
std::cout << "  Mean delay:   " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
        << std::endl;
std::cout << "  Mean jitter:   " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
        << std::endl;
auto dscpVec = classifier->GetDscpCounts(1);
for (auto p : dscpVec)
{
   std::cout << "  DSCP value:   0x" << std::hex << static_cast<uint32_t>(p.first) << std::dec
          << "  count:   " << p.second << std::endl;
}


Simulator::Stop(Seconds(10.0));

if (tracing)
{
   phy.SetPcapDataLinkType(WifiPhyHelper::DLT_IEEE802_11_RADIO);
   pointToPoint.EnablePcapAll("third");
   phy.EnablePcap("third", apDevices.Get(0));
```

```
        csma.EnablePcap("third", csmaDevices.Get(0), true);
    }

    Simulator::Run();
    Simulator::Destroy();
    return 0;
}
```

Lab Session 13

Problem Statement: Design and simulate infrastructure less network, generate two traffic flows between nodes and analyse its performance.

File Path:

- ~/ns-allinone-3.xx/ns-3.xx/Example/Wireless
- You can find examples like wifi_simple_adhoc.cc.
- Copy and Paste it in Scratch.
- Open Scratch and Rename the File.
- Open the Renamed File in Scratch

Code:

```
#include "ns3/command-line.h"
#include "ns3/config.h"
#include "ns3/double.h"
#include "ns3/internet-stack-helper.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/log.h"
#include "ns3/mobility-helper.h"
#include "ns3/mobility-model.h"
#include "ns3/string.h"
#include "ns3/yans-wifi-channel.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/traffic-control-module.h"
#include "ns3/applications-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("WifiSimpleAdhoc");

/**
 * Function called when a packet is received.
 *
 * \param socket The receiving socket.
 */
void
ReceivePacket(Ptr<Socket> socket)
{
   while (socket->Recv())
   {
      NS_LOG_UNCOND("Received one packet!");
   }
}

/**
```

```
 * Generate traffic.
 *
 * \param socket The sending socket.
 * \param pktSize The packet size.
 * \param pktCount The packet count.
 * \param pktInterval The interval between two packets.
 */
static void
GenerateTraffic(Ptr<Socket> socket, uint32_t pktSize, uint32_t pktCount, Time pktInterval)
{
   if (pktCount > 0)
   {
      socket->Send(Create<Packet>(pktSize));
      Simulator::Schedule(pktInterval,
                          &GenerateTraffic,
                          socket,
                          pktSize,
                          pktCount - 1,
                          pktInterval);
   }
   else
   {
      socket->Close();
   }
}

int
main(int argc, char* argv[])
{
   std::string phyMode("DsssRate1Mbps");
   double rss = -80;          // -dBm
   uint32_t packetSize = 1000; // bytes
   uint32_t numPackets = 1;
   double interval = 1.0; // seconds
   bool verbose = false;
   double simulationTime = 10;

   CommandLine cmd(__FILE__);
   cmd.AddValue("phyMode", "Wifi Phy mode", phyMode);
   cmd.AddValue("rss", "received signal strength", rss);
   cmd.AddValue("packetSize", "size of application packet sent", packetSize);
   cmd.AddValue("numPackets", "number of packets generated", numPackets);
   cmd.AddValue("interval", "interval (seconds) between packets", interval);
   cmd.AddValue("verbose", "turn on all WifiNetDevice log components", verbose);
   cmd.Parse(argc, argv);
   // Convert to time object
   Time interPacketInterval = Seconds(interval);

   // Fix non-unicast data rate to be the same as that of unicast
   Config::SetDefault("ns3::WifiRemoteStationManager::NonUnicastMode", StringValue(phyMode));
```

```
NodeContainer c;
c.Create(4);

// The below set of helpers will help us to put together the wifi NICs we want
WifiHelper wifi;
if (verbose)
{
   WifiHelper::EnableLogComponents(); // Turn on all Wifi logging
}
wifi.SetStandard(WIFI_STANDARD_80211b);

YansWifiPhyHelper wifiPhy;
// This is one parameter that matters when using FixedRssLossModel
// set it to zero; otherwise, gain will be added
wifiPhy.Set("RxGain", DoubleValue(0));
// ns-3 supports RadioTap and Prism tracing extensions for 802.11b
wifiPhy.SetPcapDataLinkType(WifiPhyHelper::DLT_IEEE802_11_RADIO);

YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
// The below FixedRssLossModel will cause the rss to be fixed regardless
// of the distance between the two stations, and the transmit power
wifiChannel.AddPropagationLoss("ns3::FixedRssLossModel", "Rss", DoubleValue(rss));
wifiPhy.SetChannel(wifiChannel.Create());

// Add a mac and disable rate control
WifiMacHelper wifiMac;
wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
                             "DataMode",
                             StringValue(phyMode),
                             "ControlMode",
                             StringValue(phyMode));
// Set it to adhoc mode
wifiMac.SetType("ns3::AdhocWifiMac");
NetDeviceContainer devices = wifi.Install(wifiPhy, wifiMac, c);

// Note that with FixedRssLossModel, the positions below are not
// used for received signal strength.
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator>();
positionAlloc->Add(Vector(0.0, 0.0, 0.0));
positionAlloc->Add(Vector(5.0, 0.0, 0.0));
positionAlloc->Add(Vector(0.0, 5.0, 0.0));
positionAlloc->Add(Vector(5.0, 5.0, 0.0));
mobility.SetPositionAllocator(positionAlloc);
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(c);

InternetStackHelper internet;
internet.Install(c);
```

```
Ipv4AddressHelper ipv4;
NS_LOG_INFO("Assign IP Addresses.");
ipv4.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer i = ipv4.Assign(devices);

// Flow
uint16_t port = 7;
Address localAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
PacketSinkHelper packetSinkHelper("ns3::TcpSocketFactory", localAddress);
ApplicationContainer sinkApp = packetSinkHelper.Install(c.Get(2));

sinkApp.Start(Seconds(0.0));
sinkApp.Stop(Seconds(simulationTime + 0.1));

uint32_t payloadSize = 1448;
Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(payloadSize));

OnOffHelper onoff("ns3::TcpSocketFactory", Ipv4Address::GetAny());
onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
onoff.SetAttribute("PacketSize", UintegerValue(payloadSize));
onoff.SetAttribute("DataRate", StringValue("50Mbps")); // bit/s
ApplicationContainer apps;

InetSocketAddress rmt(i.GetAddress(1), port);
rmt.SetTos(0xb8);
AddressValue remoteAddress(rmt);
onoff.SetAttribute("Remote", remoteAddress);
apps.Add(onoff.Install(c.Get(0)));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(simulationTime + 0.1));

FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();

Simulator::Stop(Seconds(simulationTime + 5));
Simulator::Run();

Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets << " / " << stats[1].txBytes
        << std::endl;
std::cout << "  Offered Load: "
        << stats[1].txBytes * 8.0 /
             (stats[1].timeLastTxPacket.GetSeconds() -
              stats[1].timeFirstTxPacket.GetSeconds()) /
             1000000
        << " Mbps" << std::endl;
std::cout << "  Rx Packets/Bytes:   " << stats[1].rxPackets << " / " << stats[1].rxBytes
        << std::endl;
```

```
  uint32_t packetsDroppedByQueueDisc = 0;
  uint64_t bytesDroppedByQueueDisc = 0;
  if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE_DISC)
  {
    packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
    bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
  }
  std::cout << "  Packets/Bytes Dropped by Queue Disc:   " << packetsDroppedByQueueDisc << " / "
        << bytesDroppedByQueueDisc << std::endl;

  Simulator::Run();
  Simulator::Destroy();

  return 0;
}
```

Lab Session 14

Problem Statement: Design a wired network with ‘n’ nodes to observe the performance of two TCP variants (Reno and Tahoe). Simulate the designed network and observe the network performance.

File Path:

- **~/ns-allinone-3.xx/ns-3.xx/Example/Wireless**
- You can find examples like wifi_simple_adhoc.cc.
- Copy and Paste it in Scratch.
- Open Scratch and Rename the File.
- Open the Renamed File in Scratch

Code:

```
#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/enum.h"
#include "ns3/error-model.h"
#include "ns3/event-id.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/internet-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/tcp-header.h"
#include "ns3/traffic-control-module.h"
#include "ns3/udp-header.h"
#include <fstream>
#include <iostream>
#include <string>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("TcpVariantsComparison");

static std::map<uint32_t, bool> firstCwnd;                  //!< First congestion window.
static std::map<uint32_t, bool> firstSshThr;                //!< First SlowStart threshold.
static std::map<uint32_t, bool> firstRtt;                  //!< First RTT.
static std::map<uint32_t, bool> firstRto;                   //!< First RTO.
static std::map<uint32_t, Ptr<OutputStreamWrapper>> cWndStream; //!< Congstion window output
stream.
static std::map<uint32_t, Ptr<OutputStreamWrapper>>
   ssThreshStream; //!< SlowStart threshold output stream.
static std::map<uint32_t, Ptr<OutputStreamWrapper>> rttStream;     //!< RTT output stream.
static std::map<uint32_t, Ptr<OutputStreamWrapper>> rtoStream;     //!< RTO output stream.
```

```
static std::map<uint32_t, Ptr<OutputStreamWrapper>> nextTxStream;   //!< Next TX output stream.
static std::map<uint32_t, Ptr<OutputStreamWrapper>> nextRxStream;   //!< Next RX output stream.
static std::map<uint32_t, Ptr<OutputStreamWrapper>> inFlightStream; //!< In flight output stream.
static std::map<uint32_t, uint32_t> cWndValue;                      //!< congestion window value.
static std::map<uint32_t, uint32_t> ssThreshValue;                  //!< SlowStart threshold value.

/**
 * Get the Node Id From Context.
 *
 * \param context The context.
 * \return the node ID.
 */
static uint32_t
GetNodeIdFromContext(std::string context)
{
   const std::size_t n1 = context.find_first_of('/', 1);
   const std::size_t n2 = context.find_first_of('/', n1 + 1);
   return std::stoul(context.substr(n1 + 1, n2 - n1 - 1));
}

/**
 * Congestion window tracer.
 *
 * \param context The context.
 * \param oldval Old value.
 * \param newval New value.
 */
static void
CwndTracer(std::string context, uint32_t oldval, uint32_t newval)
{
   uint32_t nodeId = GetNodeIdFromContext(context);

   if (firstCwnd[nodeId])
   {
      *cWndStream[nodeId]->GetStream() << "0.0 " << oldval << std::endl;
      firstCwnd[nodeId] = false;
   }
   *cWndStream[nodeId]->GetStream() << Simulator::Now().GetSeconds() << " " << newval <<
std::endl;
   cWndValue[nodeId] = newval;

   if (!firstSshThr[nodeId])
   {
      *ssThreshStream[nodeId]->GetStream()
         << Simulator::Now().GetSeconds() << " " << ssThreshValue[nodeId] << std::endl;
   }
}
```

```
/**
 * Slow start threshold tracer.
 *
 * \param context The context.
 * \param oldval Old value.
 * \param newval New value.
 */
static void
SsThreshTracer(std::string context, uint32_t oldval, uint32_t newval)
{
   uint32_t nodeId = GetNodeIdFromContext(context);

   if (firstSshThr[nodeId])
   {
      *ssThreshStream[nodeId]->GetStream() << "0.0 " << oldval << std::endl;
      firstSshThr[nodeId] = false;
   }
   *ssThreshStream[nodeId]->GetStream()
      << Simulator::Now().GetSeconds() << " " << newval << std::endl;
   ssThreshValue[nodeId] = newval;

   if (!firstCwnd[nodeId])
   {
      *cWndStream[nodeId]->GetStream()
         << Simulator::Now().GetSeconds() << " " << cWndValue[nodeId] << std::endl;
   }
}

/**
 * RTT tracer.
 *
 * \param context The context.
 * \param oldval Old value.
 * \param newval New value.
 */
static void
RttTracer(std::string context, Time oldval, Time newval)
{
   uint32_t nodeId = GetNodeIdFromContext(context);

   if (firstRtt[nodeId])
   {
      *rttStream[nodeId]->GetStream() << "0.0 " << oldval.GetSeconds() << std::endl;
      firstRtt[nodeId] = false;
   }
   *rttStream[nodeId]->GetStream()
      << Simulator::Now().GetSeconds() << " " << newval.GetSeconds() << std::endl;
}
```

```
/**
 * RTO tracer.
 *
 * \param context The context.
 * \param oldval Old value.
 * \param newval New value.
 */
static void
RtoTracer(std::string context, Time oldval, Time newval)
{
   uint32_t nodeId = GetNodeIdFromContext(context);

   if (firstRto[nodeId])
   {
      *rtoStream[nodeId]->GetStream() << "0.0 " << oldval.GetSeconds() << std::endl;
      firstRto[nodeId] = false;
   }
   *rtoStream[nodeId]->GetStream()
      << Simulator::Now().GetSeconds() << " " << newval.GetSeconds() << std::endl;
}

/**
 * Next TX tracer.
 *
 * \param context The context.
 * \param old Old sequence number.
 * \param nextTx Next sequence number.
 */
static void
NextTxTracer(std::string context, SequenceNumber32 old [[maybe_unused]], SequenceNumber32
nextTx)
{
   uint32_t nodeId = GetNodeIdFromContext(context);

   *nextTxStream[nodeId]->GetStream()
      << Simulator::Now().GetSeconds() << " " << nextTx << std::endl;
}

/**
 * In-flight tracer.
 *
 * \param context The context.
 * \param old Old value.
 * \param inFlight In flight value.
 */
static void
InFlightTracer(std::string context, uint32_t old [[maybe_unused]], uint32_t inFlight)
{
```

```
  uint32_t nodeId = GetNodeIdFromContext(context);

  *inFlightStream[nodeId]->GetStream()
    << Simulator::Now().GetSeconds() << " " << inFlight << std::endl;
}

/**
 * Next RX tracer.
 *
 * \param context The context.
 * \param old Old sequence number.
 * \param nextRx Next sequence number.
 */
static void
NextRxTracer(std::string context, SequenceNumber32 old [[maybe_unused]], SequenceNumber32
nextRx)
{
  uint32_t nodeId = GetNodeIdFromContext(context);

  *nextRxStream[nodeId]->GetStream()
    << Simulator::Now().GetSeconds() << " " << nextRx << std::endl;
}

/**
 * Congestion window trace connection.
 *
 * \param cwnd_tr_file_name Congestion window trace file name.
 * \param nodeId Node ID.
 */
static void
TraceCwnd(std::string cwnd_tr_file_name, uint32_t nodeId)
{
  AsciiTraceHelper ascii;
  cWndStream[nodeId] = ascii.CreateFileStream(cwnd_tr_file_name);
  Config::Connect("/NodeList/" + std::to_string(nodeId) +
                  "/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow",
                MakeCallback(&CwndTracer));
}

/**
 * Slow start threshold trace connection.
 *
 * \param ssthresh_tr_file_name Slow start threshold trace file name.
 * \param nodeId Node ID.
 */
static void
TraceSsThresh(std::string ssthresh_tr_file_name, uint32_t nodeId)
{
```

```
  AsciiTraceHelper ascii;
  ssThreshStream[nodeId] = ascii.CreateFileStream(ssthresh_tr_file_name);
  Config::Connect("/NodeList/" + std::to_string(nodeId) +
                  "/$ns3::TcpL4Protocol/SocketList/0/SlowStartThreshold",
                  MakeCallback(&SsThreshTracer));
}

/**
 * RTT trace connection.
 *
 * \param rtt_tr_file_name RTT trace file name.
 * \param nodeId Node ID.
 */
static void
TraceRtt(std::string rtt_tr_file_name, uint32_t nodeId)
{
  AsciiTraceHelper ascii;
  rttStream[nodeId] = ascii.CreateFileStream(rtt_tr_file_name);
  Config::Connect("/NodeList/" + std::to_string(nodeId) + "/$ns3::TcpL4Protocol/SocketList/0/RTT",
                  MakeCallback(&RttTracer));
}

/**
 * RTO trace connection.
 *
 * \param rto_tr_file_name RTO trace file name.
 * \param nodeId Node ID.
 */
static void
TraceRto(std::string rto_tr_file_name, uint32_t nodeId)
{
  AsciiTraceHelper ascii;
  rtoStream[nodeId] = ascii.CreateFileStream(rto_tr_file_name);
  Config::Connect("/NodeList/" + std::to_string(nodeId) + "/$ns3::TcpL4Protocol/SocketList/0/RTO",
                  MakeCallback(&RtoTracer));
}

/**
 * Next TX trace connection.
 *
 * \param next_tx_seq_file_name Next TX trace file name.
 * \param nodeId Node ID.
 */
static void
TraceNextTx(std::string& next_tx_seq_file_name, uint32_t nodeId)
{
  AsciiTraceHelper ascii;
  nextTxStream[nodeId] = ascii.CreateFileStream(next_tx_seq_file_name);
```

```
  Config::Connect("/NodeList/" + std::to_string(nodeId) +
                  "/$ns3::TcpL4Protocol/SocketList/0/NextTxSequence",
                  MakeCallback(&NextTxTracer));
}

/**
 * In flight trace connection.
 *
 * \param in_flight_file_name In flight trace file name.
 * \param nodeId Node ID.
 */
static void
TraceInFlight(std::string& in_flight_file_name, uint32_t nodeId)
{
  AsciiTraceHelper ascii;
  inFlightStream[nodeId] = ascii.CreateFileStream(in_flight_file_name);
  Config::Connect("/NodeList/" + std::to_string(nodeId) +
                  "/$ns3::TcpL4Protocol/SocketList/0/BytesInFlight",
                  MakeCallback(&InFlightTracer));
}

/**
 * Next RX trace connection.
 *
 * \param next_rx_seq_file_name Next RX trace file name.
 * \param nodeId Node ID.
 */
static void
TraceNextRx(std::string& next_rx_seq_file_name, uint32_t nodeId)
{
  AsciiTraceHelper ascii;
  nextRxStream[nodeId] = ascii.CreateFileStream(next_rx_seq_file_name);
  Config::Connect("/NodeList/" + std::to_string(nodeId) +
                  "/$ns3::TcpL4Protocol/SocketList/1/RxBuffer/NextRxSequence",
                  MakeCallback(&NextRxTracer));
}

int
main(int argc, char* argv[])
{
  std::string transport_prot = "TcpWestwoodPlus";
  double error_p = 0.0;
  std::string bandwidth = "2Mbps";
  std::string delay = "0.01ms";
  std::string access_bandwidth = "10Mbps";
  std::string access_delay = "45ms";
  bool tracing = true;
  std::string prefix_file_name = "TcpVariantsComparison";
```

```
uint64_t data_mbytes = 0;
uint32_t mtu_bytes = 400;
uint16_t num_flows = 1;
double duration = 100.0;
uint32_t run = 0;
bool flow_monitor = true;
bool pcap = false;
bool sack = true;
std::string queue_disc_type = "ns3::PfifoFastQueueDisc";
std::string recovery = "ns3::TcpClassicRecovery";

CommandLine cmd(__FILE__);
cmd.AddValue("transport_prot",
        "Transport protocol to use: TcpNewReno, TcpLinuxReno, "
        "TcpHybla, TcpHighSpeed, TcpHtcp, TcpVegas, TcpScalable, TcpVeno, "
        "TcpBic, TcpYeah, TcpIllinois, TcpWestwoodPlus, TcpLedbat, "
        "TcpLp, TcpDctcp, TcpCubic, TcpBbr",
        transport_prot);
cmd.AddValue("error_p", "Packet error rate", error_p);
cmd.AddValue("bandwidth", "Bottleneck bandwidth", bandwidth);
cmd.AddValue("delay", "Bottleneck delay", delay);
cmd.AddValue("access_bandwidth", "Access link bandwidth", access_bandwidth);
cmd.AddValue("access_delay", "Access link delay", access_delay);
cmd.AddValue("tracing", "Flag to enable/disable tracing", tracing);
cmd.AddValue("prefix_name", "Prefix of output trace file", prefix_file_name);
cmd.AddValue("data", "Number of Megabytes of data to transmit", data_mbytes);
cmd.AddValue("mtu", "Size of IP packets to send in bytes", mtu_bytes);
cmd.AddValue("num_flows", "Number of flows", num_flows);
cmd.AddValue("duration", "Time to allow flows to run in seconds", duration);
cmd.AddValue("run", "Run index (for setting repeatable seeds)", run);
cmd.AddValue("flow_monitor", "Enable flow monitor", flow_monitor);
cmd.AddValue("pcap_tracing", "Enable or disable PCAP tracing", pcap);
cmd.AddValue("queue_disc_type",
        "Queue disc type for gateway (e.g. ns3::CoDelQueueDisc)",
        queue_disc_type);
cmd.AddValue("sack", "Enable or disable SACK option", sack);
cmd.AddValue("recovery", "Recovery algorithm type to use (e.g., ns3::TcpPrrRecovery", recovery);
cmd.Parse(argc, argv);

transport_prot = std::string("ns3::") + transport_prot;

SeedManager::SetSeed(1);
SeedManager::SetRun(run);

// User may find it convenient to enable logging
// LogComponentEnable("TcpVariantsComparison", LOG_LEVEL_ALL);
// LogComponentEnable("BulkSendApplication", LOG_LEVEL_INFO);
// LogComponentEnable("PfifoFastQueueDisc", LOG_LEVEL_ALL);
```

```
// Calculate the ADU size
Header* temp_header = new Ipv4Header();
uint32_t ip_header = temp_header->GetSerializedSize();
NS_LOG_LOGIC("IP Header size is: " << ip_header);
delete temp_header;
temp_header = new TcpHeader();
uint32_t tcp_header = temp_header->GetSerializedSize();
NS_LOG_LOGIC("TCP Header size is: " << tcp_header);
delete temp_header;
uint32_t tcp_adu_size = mtu_bytes - 20 - (ip_header + tcp_header);
NS_LOG_LOGIC("TCP ADU size is: " << tcp_adu_size);

// Set the simulation start and stop time
double start_time = 0.1;
double stop_time = start_time + duration;

// 2 MB of TCP buffer
Config::SetDefault("ns3::TcpSocket::RcvBufSize", UintegerValue(1 << 21));
Config::SetDefault("ns3::TcpSocket::SndBufSize", UintegerValue(1 << 21));
Config::SetDefault("ns3::TcpSocketBase::Sack", BooleanValue(sack));

Config::SetDefault("ns3::TcpL4Protocol::RecoveryType",
            TypeIdValue(TypeId::LookupByName(recovery)));
// Select TCP variant
TypeId tcpTid;
NS_ABORT_MSG_UNLESS(TypeId::LookupByNameFailSafe(transport_prot, &tcpTid),
             "TypeId " << transport_prot << " not found");
Config::SetDefault("ns3::TcpL4Protocol::SocketType",
            TypeIdValue(TypeId::LookupByName(transport_prot)));

// Create gateways, sources, and sinks
NodeContainer gateways;
gateways.Create(1);
NodeContainer sources;
sources.Create(num_flows);
NodeContainer sinks;
sinks.Create(num_flows);

// Configure the error model
// Here we use RateErrorModel with packet error rate
Ptr<UniformRandomVariable> uv = CreateObject<UniformRandomVariable>();
uv->SetStream(50);
RateErrorModel error_model;
error_model.SetRandomVariable(uv);
error_model.SetUnit(RateErrorModel::ERROR_UNIT_PACKET);
error_model.SetRate(error_p);
```

```
PointToPointHelper UnReLink;
UnReLink.SetDeviceAttribute("DataRate", StringValue(bandwidth));
UnReLink.SetChannelAttribute("Delay", StringValue(delay));
UnReLink.SetDeviceAttribute("ReceiveErrorModel", PointerValue(&error_model));

InternetStackHelper stack;
stack.InstallAll();

TrafficControlHelper tchPfifo;
tchPfifo.SetRootQueueDisc("ns3::PfifoFastQueueDisc");

TrafficControlHelper tchCoDel;
tchCoDel.SetRootQueueDisc("ns3::CoDelQueueDisc");

Ipv4AddressHelper address;
address.SetBase("10.0.0.0", "255.255.255.0");

// Configure the sources and sinks net devices
// and the channels between the sources/sinks and the gateways
PointToPointHelper LocalLink;
LocalLink.SetDeviceAttribute("DataRate", StringValue(access_bandwidth));
LocalLink.SetChannelAttribute("Delay", StringValue(access_delay));

Ipv4InterfaceContainer sink_interfaces;

DataRate access_b(access_bandwidth);
DataRate bottle_b(bandwidth);
Time access_d(access_delay);
Time bottle_d(delay);

uint32_t size = static_cast<uint32_t>((std::min(access_b, bottle_b).GetBitRate() / 8) *
                        ((access_d + bottle_d) * 2).GetSeconds());

Config::SetDefault("ns3::PfifoFastQueueDisc::MaxSize",
            QueueSizeValue(QueueSize(QueueSizeUnit::PACKETS, size / mtu_bytes)));
Config::SetDefault("ns3::CoDelQueueDisc::MaxSize",
            QueueSizeValue(QueueSize(QueueSizeUnit::BYTES, size)));

for (uint32_t i = 0; i < num_flows; i++)
{
   NetDeviceContainer devices;
   devices = LocalLink.Install(sources.Get(i), gateways.Get(0));
   tchPfifo.Install(devices);
   address.NewNetwork();
   Ipv4InterfaceContainer interfaces = address.Assign(devices);

   devices = UnReLink.Install(gateways.Get(0), sinks.Get(i));
   if (queue_disc_type == "ns3::PfifoFastQueueDisc")
```

```
    {
      tchPfifo.Install(devices);
    }
    else if (queue_disc_type == "ns3::CoDelQueueDisc")
    {
      tchCoDel.Install(devices);
    }
    else
    {
      NS_FATAL_ERROR("Queue not recognized. Allowed values are ns3::CoDelQueueDisc or "
                     "ns3::PfifoFastQueueDisc");
    }
    address.NewNetwork();
    interfaces = address.Assign(devices);
    sink_interfaces.Add(interfaces.Get(1));
  }

  NS_LOG_INFO("Initialize Global Routing.");
  Ipv4GlobalRoutingHelper::PopulateRoutingTables();

  uint16_t port = 50000;
  Address sinkLocalAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
  PacketSinkHelper sinkHelper("ns3::TcpSocketFactory", sinkLocalAddress);

  for (uint32_t i = 0; i < sources.GetN(); i++)
  {
    AddressValue remoteAddress(InetSocketAddress(sink_interfaces.GetAddress(i, 0), port));
    Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(tcp_adu_size));
    BulkSendHelper ftp("ns3::TcpSocketFactory", Address());
    ftp.SetAttribute("Remote", remoteAddress);
    ftp.SetAttribute("SendSize", UintegerValue(tcp_adu_size));
    ftp.SetAttribute("MaxBytes", UintegerValue(data_mbytes * 1000000));

    ApplicationContainer sourceApp = ftp.Install(sources.Get(i));
    sourceApp.Start(Seconds(start_time * i));
    sourceApp.Stop(Seconds(stop_time - 3));

    sinkHelper.SetAttribute("Protocol", TypeIdValue(TcpSocketFactory::GetTypeId()));
    ApplicationContainer sinkApp = sinkHelper.Install(sinks.Get(i));
    sinkApp.Start(Seconds(start_time * i));
    sinkApp.Stop(Seconds(stop_time));
  }

  // Set up tracing if enabled
  if (tracing)
  {
    std::ofstream ascii;
    Ptr<OutputStreamWrapper> ascii_wrap;
```

```
  ascii.open(prefix_file_name + "-ascii");
  ascii_wrap = new OutputStreamWrapper(prefix_file_name + "-ascii", std::ios::out);
  stack.EnableAsciiIpv4All(ascii_wrap);

  for (uint16_t index = 0; index < num_flows; index++)
  {
    std::string flowString;
    if (num_flows > 1)
    {
      flowString = "-flow" + std::to_string(index);
    }

    firstCwnd[index + 1] = true;
    firstSshThr[index + 1] = true;
    firstRtt[index + 1] = true;
    firstRto[index + 1] = true;

    Simulator::Schedule(Seconds(start_time * index + 0.00001),
                        &TraceCwnd,
                        prefix_file_name + flowString + "-cwnd.data",
                        index + 1);
    Simulator::Schedule(Seconds(start_time * index + 0.00001),
                        &TraceSsThresh,
                        prefix_file_name + flowString + "-ssth.data",
                        index + 1);
    Simulator::Schedule(Seconds(start_time * index + 0.00001),
                        &TraceRtt,
                        prefix_file_name + flowString + "-rtt.data",
                        index + 1);
    Simulator::Schedule(Seconds(start_time * index + 0.00001),
                        &TraceRto,
                        prefix_file_name + flowString + "-rto.data",
                        index + 1);
    Simulator::Schedule(Seconds(start_time * index + 0.00001),
                        &TraceNextTx,
                        prefix_file_name + flowString + "-next-tx.data",
                        index + 1);
    Simulator::Schedule(Seconds(start_time * index + 0.00001),
                        &TraceInFlight,
                        prefix_file_name + flowString + "-inflight.data",
                        index + 1);
    Simulator::Schedule(Seconds(start_time * index + 0.1),
                        &TraceNextRx,
                        prefix_file_name + flowString + "-next-rx.data",
                        num_flows + index + 1);
  }
}
```

```
    if (pcap)
    {
      UnReLink.EnablePcapAll(prefix_file_name, true);
      LocalLink.EnablePcapAll(prefix_file_name, true);
    }

    // Flow monitor
    FlowMonitorHelper flowHelper;
    if (flow_monitor)
    {
      flowHelper.InstallAll();
    }

    Simulator::Stop(Seconds(stop_time));
    Simulator::Run();
    if (flow_monitor)
    {
      flowHelper.SerializeToXmlFile(prefix_file_name + ".flowmonitor", true, true);
    }
    Simulator::Destroy();
    return 0;
}
```