APJ Abdul Kalam Technological University

Musaliar College of Engineering & Technology, Pathanamthitta

Department of Computer science & Engineering

FACULTY LAB MANUAL

NETWORKING LAB

(CSL 332)

Academic Year: 2023 – 2024 **Year /Semester: III / IV**

TABLE OF CONTENTS

Institute Vision/Mission

VISION

To develop into a world class pace setter with distinct identity and character to meet the demands of a changing global technological competitive scenario with a societal thrust.

Mission

M1. To impart quality Education in Engineering & Management by providing state of the art teaching learning methods.

M2. To foster innovation in Technology and its application for meeting global Challenges.

M3. Inculcate global awareness, communication skills, team building and ethical values.

M4. To collaborate with industry and R&D organization for developing knowledge andsustainabletechnoogies.

Department Vision/Mission/PSO/PEO

Vision

To produce competent and dynamic professionals in the field of Computer Science and Engineering to thrive and cater the changing needs of the society through research and education.

Mission

M1. To impart high quality technical education and knowledge in Computer Science and Engineering.

M2. To introduce moral, ethical and social values to Computer Science and Engineering students.

M3. To establish industry institute interaction to enhance the skills of Computer Science and Engineering students.

M4. To promote research aimed towards betterment of society.

Programme Specific Outcomes.

PSO1: Understand and analyse the principles and working of software in the areas related to data base, machine learning, web technologies and networking for efficient design of computer systems.

PSO 2: The ability to utilize modern computer languages and applications, work with and communicate effectively with professionals in various fields of computing.

Program Educational Objective

PEO1: Work productively and successfully in diverse IT fields.

PEO2: Enhance their skills and embrace new computing technologies.

PEO3:Demonstrate professional attitude and ethics.

PEO4: Excel in higher education.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

PO1: Engineering Knowledge: Apply the knowledge of mathematics, science, engineering Fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/ Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

Institute Vision/Mission

VISION

To develop into a world class pace setter with distinct identity and character to meet the demands of a changing global technological competitive scenario with a societal thrust.

.

Mission

M1. To impart quality Education in Engineering & Management by providing state of the art teaching learning methods.

M2. To foster innovation in Technology and its application for meeting global Challenges.

M3. Inculcate global awareness, communication skills, team building and ethical values.

M4. To collaborate with industry and R&D organization for developing knowledge andsustainabletechnoogies.

Department Vision/Mission/PSO/PEO

Vision

To produce competent and dynamic professionals in the field of Computer Science and Engineering to thrive and cater the changing needs of the society through research and education.

Mission

M1. To impart high quality technical education and knowledge in Computer Science and Engineering.

M2. To introduce moral, ethical and social values to Computer Science and Engineering students.

M3. To establish industry institute interaction to enhance the skills of Computer Science and Engineering students.

M4. To promote research aimed towards betterment of society.

Programme Specific Outcomes.

PSO1: Understand and analyse the principles and working of software in the areas related to data base, machine learning, web technologies and networking for efficient design of computer systems.

PSO 2: The ability to utilize modern computer languages and applications, work with and communicate effectively with professionals in various fields of computing.

Program Educational Objective

PEO1: Work productively and successfully in diverse IT fields.

PEO2: Enhance their skills and embrace new computing technologies.

PEO3:Demonstrate professional attitude and ethics.

PEO4: Excel in higher education.

SYLLABUS

*Mandatory

(Note: At least one program from each topic in the syllabus should be completed in the Lab)

1. Getting started with the basics of network configuration files and networking commands in Linux.*
2. To familiarize and understand the use and functioning of system calls used for

network programming in Linux.*

3. Implement client-server communication using socket programming and TCP as transport layer protocol*

4. Implement client-server communication using socket programming and UDP as transport layer protocol*

5. Simulate sliding window flow control protocols.* (Stop and Wait, Go back N, Selective Repeat ARQ protocols)

6. Implement and simulate algorithm for Distance Vector Routing protocol or Link State Routing protocol.*

7. Implement Simple Mail Transfer Protocol.

8. Implement File Transfer Protocol.*

9. Implement congestion control using a leaky bucket algorithm.*

10. Understanding the Wireshark tool.*

11. Design and configure a network with multiple subnets with wired and wireless LANs using required network devices. Configure commonly used services in the network.*

Study of NS2 simulator*.

Networking Lab-Practice Questions

a) View the configuration, including addresses of your computers network interfaces.

b)Test the network connectivity between your computer and several other computers.

c) View the active TCP connections in the computer after visiting a website.

d)Find the hardware/MAC address of another computer in the network using ARP.

Write the system calls used for creating sockets and transferring data between two nodes.

a) Implement a multi-user chat server using TCP as transport layer protocol.

b) Implement a simple web proxy server that accepts HTTP requests and forwarding to remote servers and returning data to the client using TCP

Implement a Concurrent Time Server application using UDP to execute the program at a remote server. Client sends a time request to the server, server sends its system time back to the client. Client displays the result.

a) Implement Stop-and-Wait ARQ flow control protocol.

b) Implement Go-Back--N ARQ flow control protocol.

c) Implement Selective Repeat ARQ flow control protocol.

6. Implement Distance Vector Routing algorithm or Link State Routing algorithm..

Implement Simple Mail Transfer Protocol.

Develop a concurrent file server which will provide the file requested by a client if it exists. If not, the server sends appropriate message to the client. Server should also send its process ID (PID) to clients for display along with the file or the message.

CSL 332 NETWORKING LAB

Implement leaky bucket algorithm for congestion control.

Using Wireshark, Capture packets transferred while browsing a selected website.

a)Investigate the protocols used in each packet, the values of the header fields and the size of the packet.

b) Using Wireshark, observe three way handshaking connection establishment, three way handshaking connection termination and Data transfer in client server communication using TCP.

c) Explore at least the following features of Wireshark: filters, Flow graphs (TCP), statistics, and protocol hierarchies.

Design and configure a network (wired and wireless LANs) with multiple subnets using required network devices. Configure at least three of the following services in the network- TELNET, SSH, FTP server, Web server, File server, DHCP server and DNS server.

a) The network consists of TCP source node (n0) and destination node (n1) over an area size of 500m x 500m. Node (n0) uses Agent/TCP/Reno as the sending TCP agent and FTP traffic source. Node (n1) is the receiver of FTP transfers, and it uses Agent/TCP sink as its TCP-agent for the connection establishment. Run the simulation for 150 seconds and show the TCP window size in two static nodes scenario with any dynamic routing protocol. Run the script and analyze the output graph for the given scenario.

b) Simulate the transmission of ping messages over a star network topology consisting of 'n' nodes and find the number of packets dropped due to congestion using NS2simulator.

c) Simulate Link State Protocol or Distance Vector Routing protocol in NS2.

COURSE OUTCOMES(COs)

After the completion of the course the student will be able to:

CO1: Use network related commands and configuration files in Linux Operating System.

CO2: Develop network application programs and protocols.

CO3: Analyze network traffic using network monitoring tools.

CO4: Design and setup a network and configure different network protocols.

Sl No	Experiment Name	Page No
	The basics of network configuration files and networking commands in Linux.	
	Familiarize and understand the use and functioning of system calls used for network programming in Linux	
	Implement client-server communication using socket programming and TCP as transport layer protocol	
	Implement a multi-user chat server using TCP as transport layer protocol.	
	Implement client-server communication using socket programming and UDP as transport layer protocol	
	Implement a Concurrent Time Server application using UDP	
	Simulate sliding window flow control protocols	
	1. Stop and Wait	
	2. Go back N	
	3. Selective Repeat	
	Implement and simulate algorithm for Distance Vector Routing protocol	
	Implement File Transfer Protocol	
	Implement congestion control using a leaky bucket algorithm	
	Familiarization of Wireshark tool.	
	Familiarization of NS2 simulator	
	Design and configure a network with multiple computers with wired connections using a switch. Configure the network and check the connection	

CO5: Develop simulation of programmable network concepts using a network simulator

Software Requirements :

Operating System to Use in Lab : Linux Compiler

Software to Use in Lab : gcc ,NS2

Programming Language to Use in Lab : ANSI C

University Prescribed Lab Experiment

EXPERIMENT NO: 1

Familiarization of basics of network configuration files and networking commands in Linux**.**

AIM:-

Familiarize and understand the use and functioning of network configuration files and networking commands in Linux.

The important network configuration files in Linux operating systems are

1. /etc/hosts

This file is used to resolve hostnames on small networks with no DNS server. This text file contains a mapping of an IP address to the corresponding host name in each line. This file also contains a line specifying the IP address of the loopback device i.e, 127.0.0.1 is mapped to localhost.

A typical hosts file is as shown

127.0.0.1 localhost

127.0.1.1 anil-300E4Z-300E5Z-300E7Z

2. /etc/resolv.conf

This configuration file contains the IP addresses of DNS servers and the search domain.

A sample file is shown

DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN

nameserver 127.0.1.1

3. /etc/sysconfig/network

This configuration file specifies routing and host information for all network interfaces. It contains directives that are global specific. For example if NETWORKING=yes, then /etc/init.d/network activates network devices.

4. /etc/nsswitch.conf

This file includes database search entries. The directive specifies which database is to be searched first.

The important Linux networking commands are

1. ifconfig

This command gives the configuration of all interfaces in the system.

lap44-admin@LAP44-Admin:~$ ifconfig

enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 10.1.0.124 netmask 255.255.248.0 broadcast 10.1.7.255

inet6 fe80::b96c:305a:805f:f37a prefixlen 64 scopeid 0x20<link>

ether 64:00:6a:0f:fc:b2 txqueuelen 1000 (Ethernet)

RX packets 417276 bytes 537348723 (537.3 MB)

RX errors 0 dropped 182 overruns 0 frame 0

TX packets 110814 bytes 16253750 (16.2 MB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

inet6 ::1 prefixlen 128 scopeid 0x10<host>

loop txqueuelen 1000 (Local Loopback)

RX packets 12980 bytes 1171008 (1.1 MB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 12980 bytes 1171008 (1.1 MB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

Here, **enp2s0 and lo** are the names of the active network interfaces on the system.**enp2s0** is the first Ethernet interface. **lo** is the loopback interface. This is a special network interface that the system uses to communicate with itself. If there is wireless adapter,that interface also shown

It can be run with an interface name to get the details of the interface.

lap44-admin@LAP44-Admin:~$ ifconfig enp2s0

enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 10.1.0.124 netmask 255.255.248.0 broadcast 10.1.7.255

inet6 fe80::b96c:305a:805f:f37a prefixlen 64 scopeid 0x20<link>

ether 64:00:6a:0f:fc:b2 txqueuelen 1000 (Ethernet)

RX packets 417911 bytes 537410804 (537.4 MB)

RX errors 0 dropped 185 overruns 0 frame 0

TX packets 110954 bytes 16273312 (16.2 MB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

Enabling and disabling an interface

When a network interface is active, it can send and receive data; when it is inactive, it is not able to transmit or receive. You can use **ifconfig** to change the status of a network interface from inactive to active, or vice versa.

You can disable an active network interface using the **down** keyword. For instance, to disable the wireless network interface **wlan0**, use the command:
sudo ifconfig wlan0 down

To enable an inactive interface, provide **ifconfig** with the interface name followed by the keyword **up**.
sudo ifconfig wlan0 up

Configuring an interface

ifconfig can be used at the command line to configure (or re-configure) a network interface. This is often unnecessary since this configuration is often handled by a script when you boot the system. If you'd like to do so manually, you need superuser privileges, so we'll use **sudo** again when running these commands.

To assign a static IP address to an interface, specify the interface name and the IP address. For example, to assign the IP address **69.72.169.1** to the interface **wlan0**, use the command:

sudo ifconfig wlan0 69.72.169.1

To assign a network mask to an interface, use the keyword **netmask** and the netmask address. For instance, to configure the interface **eth1** to use a network mask of **255.255.255.0**, the command would be:

sudo ifconfig eth1 netmask 255.255.255.0

To assign a broadcast address to an interface, use the keyword **broadcast** and the broadcast address. For instance, to configure the interface **wlan1** to use a broadcast address of **172.16.25.98**, the command would be:

sudo ifconfig wlan1 broadcast 172.16.25.98

These configurations can combined in a single command. For instance, to configure interface **eth0** to use the static IP address **192.168.2.5**, the network mask **255.255.255.0**, and the broadcast address **192.168.2.7**, the command would be:

sudo ifconfig eth0 192.168.2.5 netmask 255.255.255.0 broadcast 192.168.2.7

2. ping

This is the most commonly used command for checking connectivity.

ping www.google.com

lap44-admin@LAP44-Admin:~$ ping www.google.com
PING www.google.com (142.250.205.228) 56(84) bytes of data.
64 bytes from maa05s28-in-f4.1e100.net (142.250.205.228): icmp_seq=1 ttl=58 time=16.9 ms
64 bytes from maa05s28-in-f4.1e100.net (142.250.205.228): icmp_seq=2 ttl=58 time=16.9 ms
64 bytes from maa05s28-in-f4.1e100.net (142.250.205.228): icmp_seq=3 ttl=58 time=16.8 ms
^C
--- www.google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 16.898/16.939/16.969/0.153 ms

A healthy connection is determined by a steady stream of replies with consistent times. Packet loss is

shown by discontinuity of sequence numbers. Large scale packet loss indicates problem along the path.

PING Version:

To get ping version installed on your system.

ping -V

lap44-admin@LAP44-Admin:~$ ping -V

ping utility, iputils-s20161105

min: minimum time to get a response

avg: average time to get responses

max: maximum time to get a response

Controlling the number of pings:

Earlier we did not define the number of packets to send to the server/host by using **-c** option we can do so.

ping -c 2 www.google.com

lap44-admin@LAP44-Admin:~$ ping -c 2 www.google.com

PING www.google.com (142.250.183.228) 56(84) bytes of data.

64 bytes from maa05s23-in-f4.1e100.net (142.250.183.228): icmp_seq=1 ttl=117 time=17.1 ms

64 bytes from maa05s23-in-f4.1e100.net (142.250.183.228): icmp_seq=2 ttl=117 time=15.8 ms

--- www.google.com ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 1002ms

rtt min/avg/max/mdev = 15.883/16.495/17.107/0.612 ms

3. netstat

This command command gives network status information.

netstat -i

lap44-admin@LAP44-Admin:~$ netstat -i

Kernel Interface table

Iface MTU RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg

enp2s0 1500 428113 0 223 0 113727 0 0 0 BMRU

lo 65536 14345 0 0 0 14345 0 0 0 LRU

As shown above, the command with -i flag provides information on the interfaces.

Examples of some practical netstat command :

-a -all : Show both listening and non-listening sockets. With the –interfaces option, show interfaces that are not up

-at :List all tcp ports.

-au : List all udp ports.

-l : List only listening ports

-lt : List only listening TCP ports.

-lu : List only listening UDP ports.

-lx : List only the listening UNIX ports

-s : List the statistics for all ports.

-st : List the statistics for TCP ports.

-su : List the statistics for UDP ports.

-pt : Display PID and program names in the output.

-c : Print the netstat information continuously.

-r : get the kernel routing information.

4. arp command

ARP stands for Address Resolution Protocol. The primary function of this protocol is to resolve the IP address of a system to its mac address, and hence it works between level 2(Data link layer) and level 3(Network layer).

Syntax:

arp -a [hostname]

Example:

```
lap44-admin@LAP44-Admin:~$ arp -a
? (10.1.2.27) at 18:60:24:70:88:b8 [ether] on enp2s0
? (10.1.0.128) at 64:00:6a:0f:fc:c6 [ether] on enp2s0
_gateway (10.1.0.1) at 00:1a:8c:59:d9:11 [ether] on enp2s0
? (10.1.2.159) at 00:e0:4d:39:ed:91 [ether] on enp2s0
? (10.1.0.27) at 70:5a:0f:10:e8:cd [ether] on enp2s0

lap44-admin@LAP44-Admin:~$ ping 10.1.0.101
PING 10.1.0.101 (10.1.0.101) 56(84) bytes of data.
64 bytes from 10.1.0.101: icmp_seq=1 ttl=64 time=0.268 ms
64 bytes from 10.1.0.101: icmp_seq=2 ttl=64 time=0.125 ms
64 bytes from 10.1.0.101: icmp_seq=3 ttl=64 time=0.150 ms
^C
--- 10.1.0.101 ping statistics ---
```

3 packets transmitted, 3 received, 0% packet loss, time 2051ms
rtt min/avg/max/mdev = 0.125/0.181/0.268/0.062 ms

lap44-admin@LAP44-Admin:~$ arp -a
? (10.1.2.27) at 18:60:24:70:88:b8 [ether] on enp2s0
? (10.1.0.128) at 64:00:6a:0f:fc:c6 [ether] on enp2s0
? (10.1.0.101) at 64:00:6a:0f:ed:c4 [ether] on enp2s0
_gateway (10.1.0.1) at 00:1a:8c:59:d9:11 [ether] on enp2s0
? (10.1.2.159) at 00:e0:4d:39:ed:91 [ether] on enp2s0
? (10.1.0.27) at 70:5a:0f:10:e8:cd [ether] on enp2s0

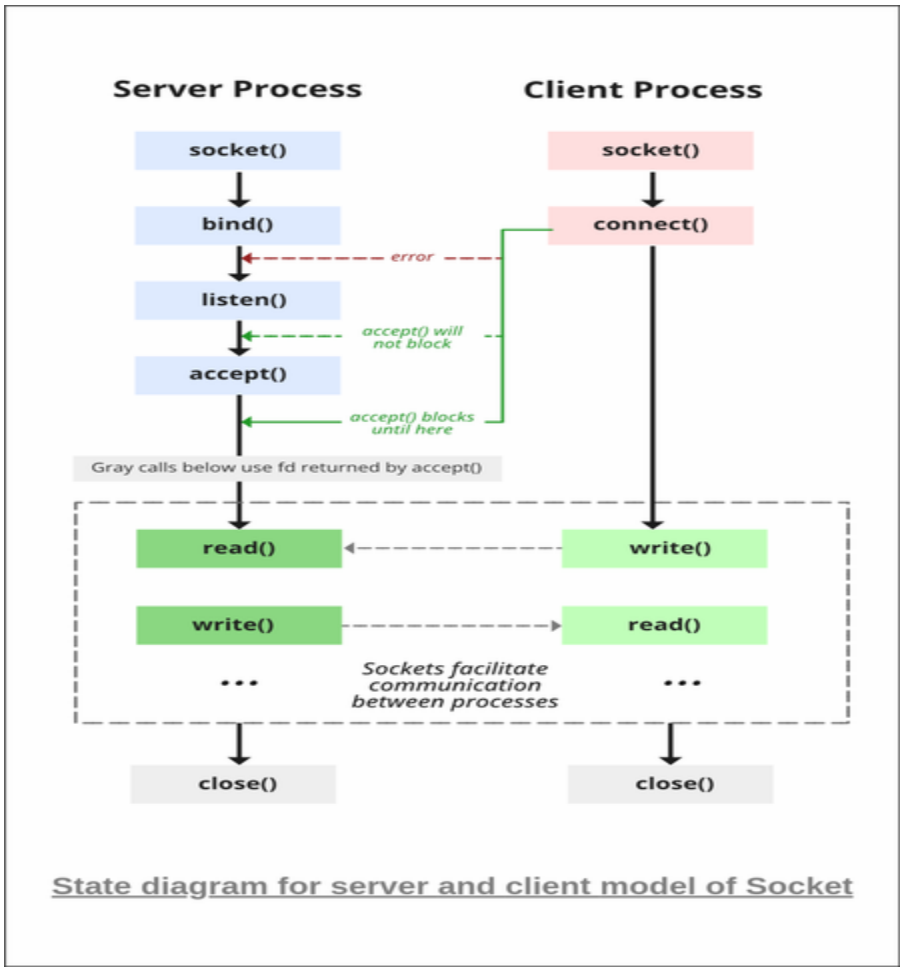
EXPERIMENT NO: 2

Familiarization of system calls used for network programming in Linux.

AIM:-

Familiarize and understand the use and functioning of system calls used for network programming in Linux.

System calls for socket programming



State diagram for server and client model of Socket

In server

1. socket()

This sytem call creates a socket and returns a socket descriptor.

int sockfd = socket(domain, type, protocol)

sockfd: socket descriptor, an integer (like a file-handle)

domain: integer, specifies communication domain. We use AF_ LOCAL as defined in the POSIX standard for communication between processes on the same host. For communicating between processes on different hosts connected by IPV4, we use AF_INET and AF_I NET 6 for processes connected by IPV6.

type: communication type

SOCK_STREAM: TCP(reliable, connection oriented)

SOCK_DGRAM: UDP(unreliable, connectionless)

protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

2. setsockopt(): This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: "address already in use".

int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);

3. bind():

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

4. Listen():

int listen(int sockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

5. Accept():

int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

In Client

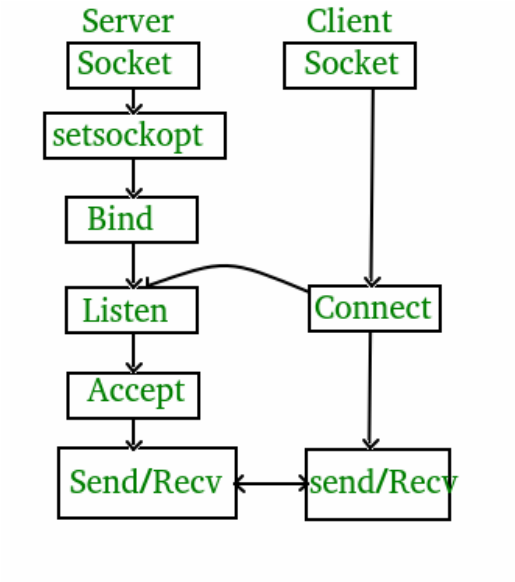
Socket connection: Exactly same as that of server's socket creation

Connect(): The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

System calls in TCP Client Server Communication

TCP Server-Client implementation in C



The entire process can be broken down into following steps:

TCP Server –

using create(), Create TCP socket.

using bind(), Bind the socket to server address.

using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection

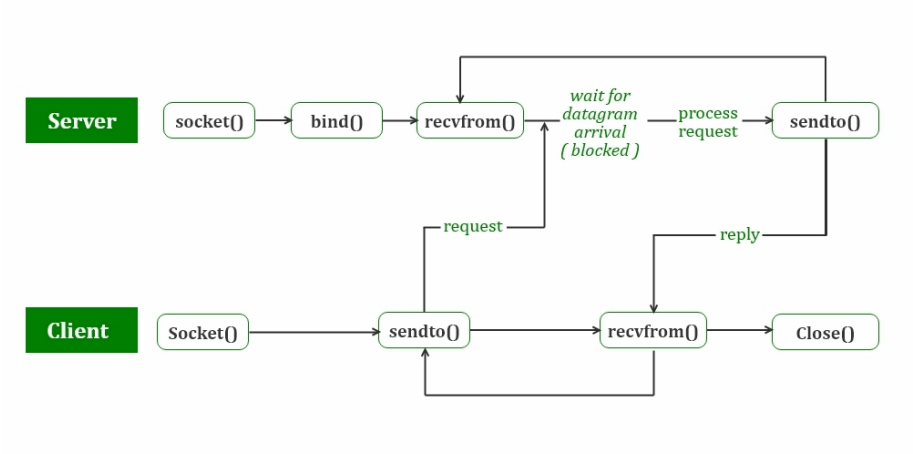
using accept(), At this point, connection is established between client and server, and they are ready to transfer data.

TCP Client –

Create TCP socket .

connect() newly created client socket to server.

UDP Server-Client Communication



UDP Server :

Create a UDP socket using socket().

bind() the socket to the server address.

Wait until the datagram packet arrives from the client using recvfrom().

Process the datagram packet and send a reply to the client using sendto().

UDP Client :

Create a UDP socket using socket()

Send a message to the server using sendto().

Wait until response from the server is received using recvfrom()

Close() socket descriptor

Necessary Functions :

int socket(int domain, int type, int protocol)

Creates an unbound socket in the specified domain.

Returns socket file descriptor.

Arguments :

domain – Specifies the communication

domain (AF_INET for IPv4/ AF_INET6 for IPv6)

type – Type of socket to be created

(SOCK_STREAM for TCP / SOCK_DGRAM for UDP)

protocol – Protocol to be used by the socket.

0 means use default protocol for the address family.

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

Assigns address to the unbound socket.

Arguments :

sockfd – File descriptor of a socket to be bonded

addr – Structure in which address to be bound to is specified

addrlen – Size of *addr* structure

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
const struct sockaddr *dest_addr, socklen_t addrlen)

Send a message on the socket

Arguments :

sockfd – File descriptor of the socket

buf – Application buffer containing the data to be sent

len – Size of *buf* application buffer

flags – Bitwise OR of flags to modify socket behavior

dest_addr – Structure containing the address of the destination

addrlen – Size of *dest_addr* structure

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
struct sockaddr *src_addr, socklen_t *addrlen)

Receive a message from the socket.

Arguments :

sockfd – File descriptor of the socket

buf – Application buffer in which to receive data

len – Size of *buf* application buffer

flags – Bitwise OR of flags to modify socket behavior

src_addr – Structure containing source address is returned

addrlen – Variable in which size of *src_addr* structure is returned

int close(int fd)

Close a file descriptor

Arguments:

fd – File descriptor

EXPERIMENT NO: 3

Client-server communication using TCP

AIM:-

To implement client-server communication using socket programming and TCP as transport layer protocol

ALGORITHM

TCP SERVER

Create a socket for TCP using the function call, socket(AF_INET, SOCK_STREAM, 0);

Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port

Bind the socket to its port using bind(server_fd, (struct sockaddr*)&address,sizeof(address))

Listen for any active client connections using listen(server_fd, 3),3 defines

the maximum length to which queue of pending connections for sockfd may grow.

Server accepts client connections using accept function call as follows:

accept(server_fd, (struct sockaddr*)&address,(socklen_t*)&addrlen)

Data is received from client using read(new_socket, buffer, 1024);

Prints the received message in server's terminal.

Sends back received data to client using send(new_socket, hello, strlen(hello), 0) function

Close the socket using close(int server_fd) function.

ALGORITHM

TCP CLIENT

Create a socket for TCP using the function call, socket(AF_INET, SOCK_STREAM, 0);

Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port

Connect using function connect(sock, (struct sockaddr*)&serv_addr,sizeof(serv_addr)

Client reads in the line from server using read(new_socket, buffer, 1024);

Prints the received message in client's terminal.

Client sends data to server using send() function in a loop

Close the socket using close()

PROGRAM

tcps.c

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
```

```
void func(int connfd)
{
    char buff[MAX];
    int n;
    // infinite loop for chat
    for (;;) {
        bzero(buff, MAX);

        // read the message from client and copy it in buffer
        read(connfd, buff, sizeof(buff));
        // print buffer which contains the client contents
        printf("From client: %s\t To client : ", buff);
        bzero(buff, MAX);
        n = 0;
        // copy server message in the buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // and send that buffer to client
        write(connfd, buff, sizeof(buff));

        // if msg contains "Exit" then server exit and chat ended.
        if (strncmp("exit", buff, 4) == 0) {
            printf("Server Exit...\n");
            break;
        }
    }
}

// Driver function
int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");
```

```
    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server accept failed...\n");
        exit(0);
    }
    else
        printf("server accept the client...\n");

    // Function for chatting between client and server
    func(connfd);

    // After chatting close the socket
    close(sockfd);
}
```

tcpc.c

```
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Enter the string : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("From Server : %s", buff);
        if ((strncmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
            break;
        }
```

```
    }
}

int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    // connect the client socket to server socket
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
        != 0) {
        printf("connection with the server failed...\n");
        exit(0);
    }
    else
        printf("connected to the server..\n");

    // function for chat
    func(sockfd);

    // close the socket
    close(sockfd);
}
```

OUTPUT

server

administrator@administrator-Vostro-3800:~/nwnew$ gcc tcps.c -o ser

administrator@administrator-Vostro-3800:~/nwnew$ ./ser

Hello from client

Hello message sent

administrator@administrator-Vostro-3800:~/nwnew$

client

```
administrator@administrator-Vostro-3800:~/nwnew$ gcc tcpc.c -o cli
administrator@administrator-Vostro-3800:~/nwnew$ ./cli
Hello message sent
Hello from server
administrator@administrator-Vostro-3800:~/nwnew$
```

EXPERIMENT NO: 4

Multi-user chat server using TCP

AIM:-

To Implement a multi-user chat server using TCP as transport layer protocol.

ALGORITHM

TCP SERVER

Create a socket for TCP using the function call, socket(AF_INET, SOCK_STREAM, 0);

Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port

Bind the socket to its port using bind(server_fd, (struct sockaddr*)&address,sizeof(address))

Listen for any active client connections using listen(server_fd, 3),3 defines
the maximum length to which queue of pending connections for sockfd may grow.
Server infinitely accepts client connections using accept function call as follows:
accept(server_fd, (struct sockaddr*)&address,(socklen_t*)&addrlen)
Child process is created. Parent process stops listening for new connections. Child will continue to listen. The main (parent) process now handles the connected client.
Data is received from client using read(new_socket, buffer, 1024);
Prints the received message in server's terminal.
Sends back received data to client using send(new_socket, hello, strlen(hello), 0) function
Close the socket using close(int server_fd) function.

ALGORITHM

TCP CLIENT

Create a socket for TCP using the function call, socket(AF_INET, SOCK_STREAM, 0);
Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port
Connect using function connect(sock, (struct sockaddr*)&serv_addr,sizeof(serv_addr)
Client reads in the line and make sure it was successful by processing the line using read() function infinitely in a loop
Prints the received message in client's terminal.
Client sends data to server using send() function in a loop
Client can continue sending messages to server, as long as server is listening.
Close the socket using close()

PROGRAM

multiserver.c

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
```

```
#include <unistd.h>
#define PORT 8080
int main(int argc, char const* argv[])
{
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = { 0 };
    char* hello;
pid_t childpid;

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0))
        == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET,
                   SO_REUSEADDR | SO_REUSEPORT, &opt,
                   sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Forcefully attaching socket to the port 8080
    if (bind(server_fd, (struct sockaddr*)&address,
             sizeof(address))
```

```
        < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }
for(;;)
{
    if ((new_socket
         = accept(server_fd, (struct sockaddr*)&address,
                  (socklen_t*)&addrlen))
        < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }

if((childpid=fork())==0)
{close(server_fd);
for(;;)
{
    valread = read(new_socket, buffer, 1024);
    printf("%s\n", buffer);
printf("Msg to client:");
scanf("%s",hello);
    send(new_socket, hello, strlen(hello), 0);

}
}
close(new_socket);
}
    return 0;
```

```
}
```

multiclient.c

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 8080

int main(int argc, char const* argv[])
{
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char* hello;
    char buffer[1024] = { 0 };
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary
    // form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)
        <= 0) {
        printf(
            "\nInvalid address/ Address not supported \n");
        return -1;
    }
```

```
   if (connect(sock, (struct sockaddr*)&serv_addr,
            sizeof(serv_addr))
      < 0) {
      printf("\nConnection Failed \n");
      return -1;
   }
for(;;)
{printf("From client(type ur Msg): ");
scanf("%s",hello);
   send(sock, hello, strlen(hello), 0);
   valread = read(sock, buffer, 1024);
   printf("%s\n", buffer);
}
   return 0;
}
```

OUTPUT

server

```
administrator@administrator-Vostro-3800:~/nwnew$ gcc multiserver.c -o ser
administrator@administrator-Vostro-3800:~/nwnew$ gcc multiclient.c -o cli
administrator@administrator-Vostro-3800:~/nwnew$ ./ser
hi
Msg to client:hi
howru
Msg to client:fine
hello
Msg to client:hello
goodmrning
```

Msg to client:goodmrning

client1

aadministrator@administrator-Vostro-3800:~/nwnew$ ./cli

From client(type ur Msg): hello

hello

From client(type ur Msg): goodmrning

goodmrning

client2

administrator@administrator-Vostro-3800:~/nwnew$ ./cli

From client(type ur Msg): hello

hello

From client(type ur Msg): goodmrning

Goodmrning

EXPERIMENT NO: 5

Client-server communication using UDP

AIM:-

To Implement client-server communication using socket programming and UDP as transport layer protocol

ALGORITHM

UDP SERVER

1. Create a socket for UDP using the function call, socket(AF_INET, SOCK_DGRAM, 0);
2. Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port

3. Bind the socket to its port using bind(s,(struct sockaddr*)&servaddr,sizeof(servaddr)
4. Receive data from client using recvfrom(s,buffer,1024,0,(struct sockaddr*)&cliaddr,&t)
5. Prints the data on server's terminal.
6. Sends back data to client using sendto(s,buffer,sizeof(buffer),0,(struct sockaddr*)&cliaddr,sizeof(cliaddr)
7. Close the socket using close(int sockfd) function.

ALGORITHM

UDP CLIENT

1. Create a socket for UDP using the function call, socket(AF_INET, SOCK_DGRAM, 0);
2. Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port
3. Bind the socket to its port using bind(s,(struct sockaddr *)&local,sizeof(local))
4. Client sends data to server using sendto(s,buffer,sizeof(buffer),0,(struct sockaddr*)&servaddr,sizeof(servaddr)
5. Client receives data from server using using recvfrom() function as follows: recvfrom(s,buffer,1024,0,(struct sockaddr *)&servaddr,&t)
6. Prints the received message in client's terminal.

PROGRAM

udps.c

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>

int main(void){
    int socket_desc;
    struct sockaddr_in server_addr, client_addr;
    char server_message[2000], client_message[2000];
    int client_struct_length = sizeof(client_addr);

    // Clean buffers:
    memset(server_message, '\0', sizeof(server_message));
    memset(client_message, '\0', sizeof(client_message));
```

```
    // Create UDP socket:
    socket_desc = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    if(socket_desc < 0){
        printf("Error while creating socket\n");
        return -1;
    }
    printf("Socket created successfully\n");

    // Set port and IP:
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(2000);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Bind to the set port and IP:
    if(bind(socket_desc, (struct sockaddr*)&server_addr, sizeof(server_addr)) <
0){
        printf("Couldn't bind to the port\n");
        return -1;
    }
    printf("Done with binding\n");

    printf("Listening for incoming messages...\n\n");

    // Receive client's message:
    if (recvfrom(socket_desc, client_message, sizeof(client_message), 0,
         (struct sockaddr*)&client_addr, &client_struct_length) < 0){
        printf("Couldn't receive\n");
        return -1;
    }
    printf("Received message from IP: %s and port: %i\n",
           inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));

    printf("Msg from client: %s\n", client_message);

    // Respond to client:
    strcpy(server_message, client_message);

    if (sendto(socket_desc, server_message, strlen(server_message), 0,
         (struct sockaddr*)&client_addr, client_struct_length) < 0){
        printf("Can't send\n");
        return -1;
    }

    // Close the socket:
    close(socket_desc);

    return 0;
}
```

udpc.c

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

```
int main(void){
    int socket_desc;
    struct sockaddr_in server_addr;
    char server_message[2000], client_message[2000];
    int server_struct_length = sizeof(server_addr);

    // Clean buffers:
    memset(server_message, '\0', sizeof(server_message));
    memset(client_message, '\0', sizeof(client_message));

    // Create socket:
    socket_desc = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    if(socket_desc < 0){
        printf("Error while creating socket\n");
        return -1;
    }
    printf("Socket created successfully\n");

    // Set port and IP:
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(2000);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Get input from the user:
    printf("Enter message: ");
    gets(client_message);

    // Send the message to server:
    if(sendto(socket_desc, client_message, strlen(client_message), 0,
         (struct sockaddr*)&server_addr, server_struct_length) < 0){
        printf("Unable to send message\n");
        return -1;
    }

    // Receive the server's response:
    if(recvfrom(socket_desc, server_message, sizeof(server_message), 0,
         (struct sockaddr*)&server_addr, &server_struct_length) < 0){
        printf("Error while receiving server's msg\n");
        return -1;
    }

    printf("Server's response: %s\n", server_message);

    // Close the socket:
    close(socket_desc);

    return 0;
}
```

OUTPUT

server

administrator@administrator-Vostro-3800:~/nwnew$ gcc udps.c -o ser

administrator@administrator-Vostro-3800:~/nwnew$ gcc udpc.c -o cli

administrator@administrator-Vostro-3800:~/nwnew$ ./ser

Client : Hello from client

Hello message sent.

client

administrator@administrator-Vostro-3800:~/nwnew$ ./cli

Hello message sent.

Server : Hello from server

EXPERIMENT NO:6

Concurrent Time Server using UDP

AIM:-

To Implement a Concurrent time server using UDP as transport layer protocol by executing the program at remote server. Client sends a time request to Server and Server sends its system time back to the client. Client displays the result.

ALGORITHM

UDP SERVER

1. Create a socket for UDP using the function call, socket(AF_INET, SOCK_DGRAM, 0);
2. Declare a time object variable ct of data type, time_t
3. Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port
4 Bind the socket to its port using bind(s,(struct sockaddr*)&servaddr,sizeof(servaddr)
5. Receive time request from client using recvfrom(s,buffer,1024,0,(struct sockaddr*)&cliaddr,&t
6. Initializes ct=time(NULL) and Prints the current date and time by calling ctime(&ct).
7. Child process is created. Parent process stops listening for new connections. Child will continue to accept TIME requests from other clients, since it is a concurrent server. The main (parent) process now handles the connected client.
8. TIME request is received from client using recvfrom(s,buffer,1024,0,(struct sockaddr*)&cliaddr,&t
9. Prints the formatted string TIME to buffer.
10. Sends back UPDATED CURRENT TIME to client using sendto(s,buffer,sizeof(buffer),0,(struct sockaddr*)&cliaddr,sizeof(cliaddr)
11. Close the socket using close(int sockfd) function.

ALGORITHM

UDP CLIENT

1. Create a socket for UDP using the function call, socket(AF_INET, SOCK_DGRAM, 0);
2. Initialize the structure sockaddr_in members of sin_family, sin_addr, sin_port
3. Bind the socket to its port using bind(s,(struct sockaddr *)&local,sizeof(local))
4. Client sends TIME request to server using sendto(s,buffer,sizeof(buffer),0,(struct sockaddr*)&servaddr,sizeof(servaddr)
5. Client receives TIME response from server using using recvfrom() function as follows: recvfrom(s,buffer,1024,0,(struct sockaddr *)&servaddr,&t)
6. Prints the received message in client's terminal.

PROGRAM

conudps.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
```

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include<time.h>
#define PORT  8088
#define MAXLINE 1024
// Driver code
int main() {
        int sockfd,cp,t;
        char buffer[MAXLINE];
        time_t ct;
        struct sockaddr_in servaddr, cliaddr;
        // Creating socket file descriptor
        if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
                perror("socket creation failed");
                exit(EXIT_FAILURE);
        }
        memset(&servaddr, 0, sizeof(servaddr));
        memset(&cliaddr, 0, sizeof(cliaddr));

        // Filling server information
        servaddr.sin_family = AF_INET; // IPv4
        servaddr.sin_addr.s_addr = INADDR_ANY;
        servaddr.sin_port = htons(PORT);
        // Bind the socket with the server address
        if ( bind(sockfd, (const struct sockaddr *)&servaddr,
                        sizeof(servaddr)) < 0 )
        {
                perror("bind failed");
                exit(EXIT_FAILURE);
        }

        int len, n;
```

```
        len = sizeof(cliaddr); //len is value/result
        while(1)
        {
        n = recvfrom(sockfd, (char *)buffer, MAXLINE,
        MSG_WAITALL, ( struct sockaddr *) &cliaddr,&len);
                ct=time(NULL);
        sprintf(buffer,"%s",ctime(&ct));
        if((cp=fork())==0)
        {
        while(1)
        {sendto(sockfd, buffer, strlen(buffer),MSG_CONFIRM, (const struct sockaddr *) &cliaddr,len);
        recvfrom(sockfd,buffer,1024,0,(struct sockaddr*)&cliaddr,&t);
        sprintf(buffer,"%s",ctime(&ct));
        }
        }
        else if(cp<0) {
        perror("fork error");
        exit(0);
        }
        }
        close(sockfd);
        return 0;
}
```

conudpc.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

```
#include <netinet/in.h>

#define PORT  8088
#define MAXLINE 1024

// Driver code
int main() {
        int sockfd;
        char buffer[MAXLINE];

        struct sockaddr_in       servaddr;

        // Creating socket file descriptor
        if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
                perror("socket creation failed");
                exit(EXIT_FAILURE);
        }
        memset(&servaddr, 0, sizeof(servaddr));
        // Filling server information
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(PORT);
        servaddr.sin_addr.s_addr = INADDR_ANY;

        int n, len;
        strcpy(buffer,"TIME");
        sendto(sockfd, buffer, strlen(buffer),
                MSG_CONFIRM, (const struct sockaddr *) &servaddr,
                        sizeof(servaddr));
        printf("Time Request sent.\n");
        n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                                MSG_WAITALL, (struct sockaddr *) &servaddr,
                                &len);
        buffer[n] = '\0';
```

```
        printf("Time from Server : %s\n", buffer);

        close(sockfd);
        return 0;
}
```

OUTPUT

Server

```
administrator@administrator-Vostro-3800:~/nwnew$ gcc conudps.c -o ser
administrator@administrator-Vostro-3800:~/nwnew$ gcc conudpc.c -o cli
administrator@administrator-Vostro-3800:~/nwnew$ ./ser
```

Clent1

```
administrator@administrator-Vostro-3800:~/nwnew$ ./cli
Time Request sent.
Time from Server : Wed Jun  1 13:00:00 2022
```

Client2

```
administrator@administrator-Vostro-3800:~/nwnew$ ./cli
Time Request sent.
Time from Server : Wed Jun  1 13:00:04 2022
```

EXPERIMENT NO: 7

Simulate sliding window flow control protocols

1. Stop and Wait

AIM:-

To Implement stop and wait sliding window flow control protocol-

ALGORITHM

Start

Set seq:no=0,waiting time=5,disconnect=0,turn='s',errorframe=1 and errorack=1

repeat step 4 to 6 until disconnect=1

call the function sender();

wait for some time

call the function receiver()

stop

Algorithm for function sender()

Start

set flag=0;

If turn=s Then

If errorack=0 Then

print "sent packet with seq no"

check for errorframe and If errorframe==0 Then

print "Error in sending pkt"

set turn=r

Else

If flag=1 Then

Print "Received ack for the seqno "

If seqno=10 Then

disconnect=1 and returned

Increment the seqno by 1

Print " sent packet with seq no"

check for errorframe and If errorframe=0 Then

print "Error in sending pkt"

set turn=r and flag=1

Else

Decrement waiting time by 1

Display "Waiting time"

If t=0 Then

turn=s

errorack=0

t=5

4. Stop

Algorithm for function receiver()

Start
set frexp=1;
If turn=r Then
    If erroframe!=0 Then
        If seq=frexp Then
            print "received packet with seq no"
            increment frexp by 1
            set ak=seqno
            set turn=s
            check for errorack and If errorack=0 Then
            print "Error in sending ack"

    Else
        Print "Duplicated packet with seqno(frexp-1) "
        set ak=seqno-1
        set turn=s
        check for errorack and If errorack==0 Then
            print "Error in sending ack"
4. Stop

PROGRAMS

```
#include <stdio.h>
#include <stdlib.h>

int ak,seq;
int t=5,k;
int disconnect=0;
char turn='s';
```

```
int errorframe=1,errorack=1;
void sender();
void receiver();
void main()
{
 seq=0;
while(!disconnect)
 {
        sender();
        for(k=1;k<=10000000;k++);
        receiver();
  }
}
void sender()
{
        static int flag=0;
        if(turn=='s')
        {
                if(errorack==0)
                {
                 printf("SENDER: sent packet with seq NO:%d\n",seq);
                 errorframe=rand()%4;
                printf("%s\n",(errorframe==0?"Error While sending Packet":""));
                turn='r';
                }
                else
                 {
                if (flag==1) printf("SENDER: Received ACK for packet %d\n",ak);
                 if (seq==10){  disconnect=1; return;}
                 seq=seq+1;
                printf("SENDER: sent packet with seq NO:%d\n",seq);
                 errorframe=rand()%4;
                printf("%s\n",(errorframe==0?"Error While sending Packet":""));
```

```
                turn='r';
                flag=1;

                }
        }
        else
        {       t--;
                printf("SENDER time reducing\n");
                if(t==0)
                {turn='s'  ;
                errorack=0;
                t=5;
                }
}


}
void receiver()
{   static int frexp=1;
   if(turn=='r')
   {
   if (errorframe!=0)
        { if(seq==frexp)
                {       printf("RECEIVER: Received packet with seq %d\n",seq);
                        ak=seq;
                        frexp=frexp+1;
                        turn='s';
                        errorack=rand()%4;
                        printf("%s\n",(errorack==0?"Error While sending ACK":""));
                }
          else
                {
                        printf("RECEIVER: Duplicated packet with seq %d\n",frexp-1);
```

```
                        ak=frexp-1;
                        turn='s';
                        errorack=rand()%4;
                        printf("%s\n",(errorack==0?"Error While sending ACK":""));
                }
        }
}
}
```

OUTPUT

```
lap44-admin@LAP44-Admin:~$ gcc stopwait.c
lap44-admin@LAP44-Admin:~$ ./a.out
SENDER: sent packet with seq NO:1

RECEIVER: Received packet with seq 1

SENDER: Received ACK for packet 1
SENDER: sent packet with seq NO:2

RECEIVER: Received packet with seq 2

SENDER: Received ACK for packet 2
SENDER: sent packet with seq NO:3

RECEIVER: Received packet with seq 3

SENDER: Received ACK for packet 3
SENDER: sent packet with seq NO:4

RECEIVER: Received packet with seq 4
Error While sending ACK
SENDER: sent packet with seq NO:4
```

RECEIVER: Duplicated packet with seq 4

SENDER: Received ACK for packet 4
SENDER: sent packet with seq NO:5

RECEIVER: Received packet with seq 5

SENDER: Received ACK for packet 5
SENDER: sent packet with seq NO:6

RECEIVER: Received packet with seq 6

SENDER: Received ACK for packet 6
SENDER: sent packet with seq NO:7

RECEIVER: Received packet with seq 7

SENDER: Received ACK for packet 7
SENDER: sent packet with seq NO:8
Error While sending Packet
SENDER time reducing
SENDER time reducing
SENDER time reducing
SENDER time reducing
SENDER time reducing
SENDER: sent packet with seq NO:8

RECEIVER: Received packet with seq 8
Error While sending ACK
SENDER: sent packet with seq NO:8
Error While sending Packet
SENDER time reducing

SENDER time reducing

SENDER time reducing

SENDER time reducing

SENDER time reducing

SENDER: sent packet with seq NO:8

RECEIVER: Duplicated packet with seq 8

Error While sending ACK

SENDER: sent packet with seq NO:8

RECEIVER: Duplicated packet with seq 8

SENDER: Received ACK for packet 8

SENDER: sent packet with seq NO:9

RECEIVER: Received packet with seq 9

SENDER: Received ACK for packet 9

SENDER: sent packet with seq NO:10

RECEIVER: Received packet with seq 10

SENDER: Received ACK for packet 10

2. Go Back N

AIM:-

To Implement Go Back N sliding window flow control protocol

ALGORITHM

Start

Read no of frames

Read window size

Call The algorithm Transmission()

Print The Total frames sent

Stop

Algorithm for transmission()

Start

Repeat steps 3 to 8 when no of frames sent <= total frames Then

Repeat step 4 and 5 until frames sent<window size and not exceeds to total frames

Sending Frame

Increment the frame sending count

Repeat step 7 and 8 and 5 when frames sent<window size and not exceeds to total frames

f=rand()%2

If !f Then

Print acknowledgement is received

Increment the ack count

Else

Print "Nor receved the frame, retransmitting the window"

Break from the loop(Go to step 2)

9. Return total frames sent

PROGRAMS

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>

int main()
{
 int nf,N;
 int tr=0;
 srand(time(NULL));
 printf("Enter the number of frames : ");
 scanf("%d",&nf);
 printf("Enter the Window Size : ");
 scanf("%d",&N);
 int i=1;
 while(i<=nf)
 {
     int x=0;
     for(int j=i;j<i+N && j<=nf;j++)
     {
         printf("Sent Frame %d \n", j);
         tr++;
     }
     for(int j=i;j<i+N && j<=nf;j++)
```

```
    {
        int flag = rand()%2;
        if(!flag)
            {
                printf("%d : Acknowledged! \n", j);
                x++;
            }
        else
            {   printf("Frame %d Not Received \n", j);
                printf("Retransmitting Window \n");
                break;
            }
    }
    printf("\n");
    i+=x;
 }
 printf("Total number of transmissions : %d \n", tr);
 return 0;
}
```

OUTPUT

administrator@administrator-HCL-Desktop:~$ gcc gobackn.c

administrator@administrator-HCL-Desktop:~$ ./a.out

Enter the Total number of frames : 9

Enter the Window Size : 3

Sending Frame 1...

Sending Frame 2...

Sending Frame 3...

Timeout!! Frame Number :1 Not Received ...

Retransmitting Window...

Sending Frame 1...

Sending Frame 2...

Sending Frame 3...

Acknowledgment for Frame 1...

Timeout!! Frame Number :2 Not Received ...

Retransmitting Window...

Sending Frame 2...

Sending Frame 3...

Sending Frame 4...
Timeout!! Frame Number :2 Not Received ...
Retransmitting Window...

Sending Frame 2...
Sending Frame 3...
Sending Frame 4...
Timeout!! Frame Number :2 Not Received ...
Retransmitting Window...

Sending Frame 2...
Sending Frame 3...
Sending Frame 4...
Timeout!! Frame Number :2 Not Received ...
Retransmitting Window...

Sending Frame 2...
Sending Frame 3...
Sending Frame 4...
Acknowledgment for Frame 2...
Acknowledgment for Frame 3...
Timeout!! Frame Number :4 Not Received ...
Retransmitting Window...

Sending Frame 4...
Sending Frame 5...
Sending Frame 6...
Timeout!! Frame Number :4 Not Received ...
Retransmitting Window...

Sending Frame 4...
Sending Frame 5...
Sending Frame 6...

Acknowledgment for Frame 4...
Timeout!! Frame Number :5 Not Received ...
Retransmitting Window...

Sending Frame 5...
Sending Frame 6...
Sending Frame 7...
Acknowledgment for Frame 5...
Timeout!! Frame Number :6 Not Received ...
Retransmitting Window...

Sending Frame 6...
Sending Frame 7...
Sending Frame 8...
Timeout!! Frame Number :6 Not Received ...
Retransmitting Window...

Sending Frame 6...
Sending Frame 7...
Sending Frame 8...
Acknowledgment for Frame 6...
Acknowledgment for Frame 7...
Acknowledgment for Frame 8...

Sending Frame 9...
Acknowledgment for Frame 9...

Total number of frames which were sent and resent are :34...

3. Selective Repeat

AIM:-

To Implement Selective Repeat sliding window flow control protocol

ALGORITHM

Start

Read no of frames

Read window size

Call The algorithm for input()

Call the algorithm for display()

Call the algorithm for Selective_Repeat()

Stop

Algorithm for input()

Start

repeat step 3 for count<= total frame size

Read the frame

Stop

Algorithm for display()

Start

repeat step 3 for count<= total frame size

Print the frame

Stop

Algorithm for selective_repeat()

Start

repeat step 3 and 4 for i=1 to frame size

Call sender() with orginal frame data

If i % window size=0 Then

    Call sender() with Not acknoledged frames in current window

After transmission of last window call sender() with not acknowledged frames

Stop

Algorithm for sender()

Start

Calculate Flag=rand()%2

If flag and first transmission Then

Print “Frame send and acknowledged”

Else if Flag and retranmission Then

Print “Frame retransmitted and acknowledged”

Else If it is a retransmission Then

Print “Frame retransmitted and not acknowledged”

Else Print “Frame send and not acknowledged”

Stop

Programs

```
#include<stdio.h>

#include<stdlib.h>

int input(int a[] , int frame_size)
{
    printf("\n\n Input \n\n");
    for(int i = 1 ; i <= frame_size ; i++)
    {
        printf(" Enter Value For Frame[%d] : " , i);
        scanf("%d",&a[i]);
        printf("\n");
    }
    printf("\n\n");
    return 1;
}
int display(int a[] , int frame_size)
{
    printf("\n\n Display \n\n");
    for(int i = 1 ; i <= frame_size ; i++)
    {
        printf(" Frame[%d] : %d " , i , a[i]);
        printf("\n");
    }
    printf("\n\n");
    return 1;
}
int selective_repeat(int frames[] , int window_size , int frame_size)
{
        int nt =0;

        int k = 0;

        int left[10000] = {-1};
```

```
                int i ;

                for(i = 1 ; i <= frame_size ; i++)
        {
                int flag = rand() % 2;

                if(flag)
                {
                        printf(" Frame[%d] with value %d Acknowledged !!! \n\n",
i , frames[i]);
                        nt++;
                }
                else
                {
                        printf(" Frame[%d] with value %d Not Acknowledged !!!
\n\n", i , frames[i]);

                        left[k++] = frames[i];

                        nt++;
                }
                if(i % window_size == 0)
                {
                        for(int x = 0 ; x < k ; x++)
                        {
                                printf(" Frame[%d] with value %d Retransmitted
\n\n", x , left[x]);

                                nt++;

                                printf(" Frame[%d] with value %d Acknowledged on
Second Attempt \n\n", x , left[x]);

                        }
                        k = 0;
                }
        }
         for(i = 0 ; i < k ; i++)
         {
              printf(" Frame[%d] with value %d Retransmitted \n\n", i ,
left[i]);

                nt++;

                printf(" Frame[%d] with value %d Acknowledged on Second Attempt
\n\n", i , left[i]);
         }
         printf(" Total Transmissions :  %d \n\n", nt);
                return 0;
}
int main()
{
         int frames[50];
```

```
        int window_size;
        int frame_size;
        printf("\n\n Selective Repeat \n\n");
                printf(" Enter Window Size : ");
                scanf("%d",&window_size);
                printf(" Enter Number Of Frames To Be Transmitted : ");
                scanf("%d",&frame_size);
                input(frames , frame_size);
                display(frames , frame_size);
                selective_repeat(frames , window_size , frame_size);
                return 0;
}
```

OUTPUT

administrator@administrator-HCL-Desktop:~$ gcc selrepeat.c
administrator@administrator-HCL-Desktop:~$ ./a.out
Enter Window Size : 3
Enter Number Of Frames To Be Transmitted : 10

Input

Enter Value For Frame[1] : 1

Enter Value For Frame[2] : 2

Enter Value For Frame[3] : 3

Enter Value For Frame[4] : 4

Enter Value For Frame[5] : 5

Enter Value For Frame[6] : 6

Enter Value For Frame[7] : 7

Enter Value For Frame[8] : 8

Enter Value For Frame[9] : 9

Enter Value For Frame[10] : 99

Display

Frame[1] : 1
Frame[2] : 2
Frame[3] : 3
Frame[4] : 4
Frame[5] : 5
Frame[6] : 6
Frame[7] : 7
Frame[8] : 8
Frame[9] : 9
Frame[10] : 99

Frame[1] with value 1 send !!!

Frame Acknowledged !!!

Frame[2] with value 2 send !!!

Frame not Acknowledged !!!

Frame[3] with value 3 send !!!

Frame Acknowledged !!!

Frame[0] retransmitted with value 2 !!!

Frame Acknowledged !!!

Frame[4] with value 4 send !!!

Frame Acknowledged !!!

Frame[5] with value 5 send !!!

Frame Acknowledged !!!

Frame[6] with value 6 send !!!

Frame not Acknowledged !!!

Frame[0] retransimitted with value 6 !!!

Frame not Acknowledged !!!

Frame[1] retransmitted with value 6 !!!

Frame Acknowledged !!!

Frame[7] with value 7 send !!!

Frame Acknowledged !!!

Frame[8] with value 8 send !!!

Frame not Acknowledged !!!

Frame[9] with value 9 send !!!

Frame Acknowledged !!!

Frame[0] retransimitted with value 8 !!!

Frame not Acknowledged !!!

Frame[1] retransmitted with value 8 !!!

Frame Acknowledged !!!

Frame[10] with value 99 send !!!

Frame Acknowledged !!!

Total Transmissions : 15

EXPERIMENT :8

Implement and simulate algorithm for Distance Vector Routing protocol

AIM:-

To Implement and simulate algorithm for Distance Vector Routing protocol

ALGORITHM

A router transmits its distance vector to each of its neighbors in a routing packet.

Each router receives and saves the most recently received distance vector from each of its neighbors.

A router recalculates its distance vector when:

It receives a distance vector from a neighbor containing different information than before.

It discovers that a link to a neighbor has gone down.

The DV calculation is based on minimizing the cost to each destination

Dx(y) = Estimate of least cost from x to y

C(x,v) = Node x knows cost to each neighbor v

Dx = [Dx(y): y ∈ N] = Node x maintains distance vector

Node x also maintains its neighbors' distance vectors

For each neighbor v, x maintains Dv = [Dv(y): y ∈ N]

PROGRAMS

```
#include<stdio.h>
struct node {
unsigned dist[20];
unsigned from[20];
}rt[10];
int main()
{ int costmat[20][20];
```

```
int nodes,i,j,k,count=0;
printf("\nEnter the number of nodes : ");
scanf("%d",&nodes);
printf("\nEnter the cost matrix :\n");
for(i=0;i<nodes;i++)
{
for(j=0;j<nodes;j++)
{
scanf("%d",&costmat[i][j]);
costmat[i][i]=0;
rt[i].dist[j]=costmat[i][j];//initialize the distance equal to cost matrix
rt[i].from[j]=j; // initialize the source node
}
}
do {
count=0;
for(i=0;i<nodes;i++) //We choose arbitrary vertex k and we calculate the direct distance from node i to k
//using cost matrix.
//and add the distance from k to node j
for(j=0;j<nodes;j++)
for(k=0;k<nodes;k++)
if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])
{//We calculate the minimum distance
rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
rt[i].from[j]=k;
count++;
}
}while(count!=0);
for(i=0;i<nodes;i++)
{
printf("\n\n For router %d\n",i+1);
for(j=0;j<nodes;j++)
```

```
{
printf("\t\nnode %d via %d Distance %d ",j+1,rt[i].from[j]+1,rt[i].dist[j]);
}
}

printf("\n\n");

}
```

OUTPUT

```
administrator@administrator-HCL-Desktop:~$ gcc distvector.c
administrator@administrator-HCL-Desktop:~$ ./a.out

Enter the number of nodes : 3

Enter the cost matrix :
0 2 7
2 0 1
7 1 0

 For router 1

node 1 via 1 Distance 0
node 2 via 2 Distance 2
node 3 via 2 Distance 3

 For router 2

node 1 via 1 Distance 2
node 2 via 2 Distance 0
node 3 via 3 Distance 1
```

For router 3

node 1 via 2 Distance 3
node 2 via 2 Distance 1
node 3 via 3 Distance 0

EXPERIMENT :9

Implement File Transfer Protocol

AIM:-

To Implement File Transfer protocol

ALGORITHM

Steps involved in writing the Server Process:

Create a socket using socket( ) system call with address family AF_INET, type SOCK_STREAM and default protocol.

Bind server's address and port using bind( ) system call.

Wait for client connection to complete accepting connections using accept( ) system call.

Receive the Clients file using recv() system call .

Using *fgets(char *str, int n, FILE *stream) function, we read a line of text from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, or when the end-of-file is reached.

On successful execution i.e. when file pointer reaches end of file, file transfer "completed" message is sent by the server to the accepted client connection using newsd, socket file descriptor.

Steps involved in writing the Client Process:

Create a socket system call with address family AF_INET, type SOCK_STREAM and default protocol.

Enter the client port id

Fill in the internet socket address structure (with server information).

Connect to the server address using connect() system call.

Read the existing and new file name from user.

Send existing file to server using send() system call

Receive feedback from server "Completed", regarding file transfer completion.

Write "File is transferred" to standard output screen.

Close the socket connection and file pointer.

PROGRAMS

/*FTP server*/

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
FILE *fp;
int sd,newsd,ser,n,a,cli,pid,bd,port,clilen;
char name[100],fileread[100],fname[100],ch,file[100],rcv[100];
struct sockaddr_in servaddr,cliaddr;
printf("Enter the port address\n");
scanf("%d",&port);
sd=socket(AF_INET,SOCK_STREAM,0);
if(sd<0)
printf("Cant create\n");
else
printf("Socket is created\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(port);
a=sizeof(servaddr);
bd=bind(sd,(struct sockaddr *)&servaddr,a);
if(bd<0)
printf("Cant bind\n");
else
printf("Binded\n");
listen(sd,5);
clilen=sizeof(cliaddr);
newsd=accept(sd,(struct sockaddr *)&cliaddr,&clilen);
if(newsd<0)
{
printf("Cant accept\n");
}
else
printf("Accepted\n");
n=recv(newsd,rcv,100,0);
rcv[n]='\0';
fp=fopen(rcv,"r");
if(fp==NULL)
{
send(newsd,"error",5,0);
close(newsd);
}
else
{
while(fgets(fileread,sizeof(fileread),fp))
{
if(send(newsd,fileread,sizeof(fileread),0)<0)
{
```

```
printf("Can't send file contents\n");
}
sleep(1);
}
if(!fgets(fileread,sizeof(fileread),fp))
{
//when file pointer reaches end of file, file transfer "completed" message is
send to accepted client connection using newsd, socket file descriptor.
send(newsd,"completed",999999999,0);
}
return(0);
}
}
```

/*FTP Client*/

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<unistd.h>
int main()
{
FILE *fp;
int csd,n,ser,s,cli,cport,newsd;
char name[100],rcvmsg[100],rcvg[100],fname[100];
struct sockaddr_in servaddr;
printf("Enter the port");
scanf("%d",&cport);
csd=socket(AF_INET,SOCK_STREAM,0);
if(csd<0)
{
printf("Error....\n");
exit(0);
}
else
printf("Socket is created\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(cport);
if(connect(csd,(struct sockaddr *)&servaddr,sizeof(servaddr))<0)
printf("Error in connection\n");
else
printf("connected\n");
printf("Enter the existing file name\t");
scanf("%s",name);
printf("Enter the new file name\t");
scanf("%s",fname);
fp=fopen(fname,"w");
send(csd,name,sizeof(name),0);
while(1)
{
s=recv(csd,rcvg,100,0);
rcvg[s]='\0';
```

```
if(strcmp(rcvg,"error")==0)
printf("File is not available\n");
if(strcmp(rcvg,"completed")==0)
{
printf("File is transferred........\n");
fclose(fp);
close(csd);
break;
}
else
fputs(rcvg,stdout);
fprintf(fp,"%s",rcvg);
return 0;
}
}
```

OUTPUT

administrator@administrator-Vostro-3250:~$ gcc ftpser.c -o ser

administrator@administrator-Vostro-3250:~$ ./ser

Enter the port address

8080

Socket is created

Binded

Accepted

administrator@administrator-Vostro-3250:~$ gcc ftpcli.c -o cli

administrator@administrator-Vostro-3250:~$ ./cli

Enter the port8080

Socket is created

connected

Enter the existing file name hi.txt

Enter the new file name hello.txt

sfsighdg

khsgsuihg

File is transferred.....

EXPERIMENT :10

Implement congestion control using a leaky bucket algorithm

AIM:-

To Implement leaky bucket algorithm

ALGORITHM

Start

Read bucket size, outgoing rate and no of inputs

Set store=0

Repeat step 5 to 10 until no of inputs =0

Read incoming packet size

If incoming packet size <= (bucket size – store Then

store =store + incoming packet size;

Print “Bucket remaining buffer size and bucket size”

Else

Print “Dropped packets”

Set store = buck_size;

Print "Bucket remaining buffer size and bucket size);

Set store = store - outgoing;

Print "After outgoing packets left out and bucket size);

Decrement no of inputs by 1

Stop

PROGRAMS

```
#include<stdio.h>
int main(){
    int incoming, outgoing, buck_size, n, store = 0;
    printf("Enter bucket size, outgoing rate and no of inputs: ");
    scanf("%d %d %d", &buck_size, &outgoing, &n);

    while (n != 0) {
        printf("Enter the incoming packet size : ");
        scanf("%d", &incoming);
        printf("Incoming packet size %d\n", incoming);
```

```
        if (incoming <= (buck_size - store)){
            store += incoming;
            printf("Bucket buffer size %d out of %d\n", store, buck_size);
        } else {
            printf("Dropped %d no of packets\n", incoming - (buck_size -
store));
            printf("Bucket buffer size %d out of %d\n", store, buck_size);
            store = buck_size;
        }
        store = store - outgoing;
        printf("After outgoing %d packets left out of %d in buffer\n", store,
buck_size);
        n--;
    }
}
```

OUTPUT

administrator@administrator-Vostro-3800:~$ gcc leaky.c

administrator@administrator-Vostro-3800:~$ ./a.out

Enter bucket size, outgoing rate and no of inputs: 50 20 3

Enter the incoming packet size : 30

Bucket buffer size 30 out of 50

After outgoing 10 packets left out of 50 in buffer

Enter the incoming packet size : 60

Dropped 20 no of packets

Bucket buffer size 50 out of 50

After outgoing 30 packets left out of 50 in buffer

Enter the incoming packet size : 20

Bucket buffer size 50 out of 50

After outgoing 30 packets left out of 50 in buffer

EXPERIMENT :11

Familiarization of Wireshark tool.

AIM:-

To familiarize the Wireshark tool.

Wireshark is a free and open-source network protocol analyzer widely used around the globe.

With Wireshark, you can capture incoming and outgoing packets of a network in real-time and use it for network troubleshooting, packet analysis, software and communication protocol development, and many more.

It is available on all major desktop operating systems like Windows, Linux, macOS and more.

Installing Wireshark on Ubuntu

First make sure that all your system packages are up-to-date by running these following apt-get commands in the terminal.

```
sudo apt-get update
sudo apt-get upgrade
```

Installing Wireshark.

Run the commands below in terminal to install wireshark:

```
apt-get install wireshark
```

During the installation,it will require to confirm security about allowing non-superuser to execute Wireshark. Just confirm YES if you want to. If you check on NO, you must run Wireshark with sudo. Later, if you want to change this:

```
sudo dpkg-reconfigure wireshark-common
```

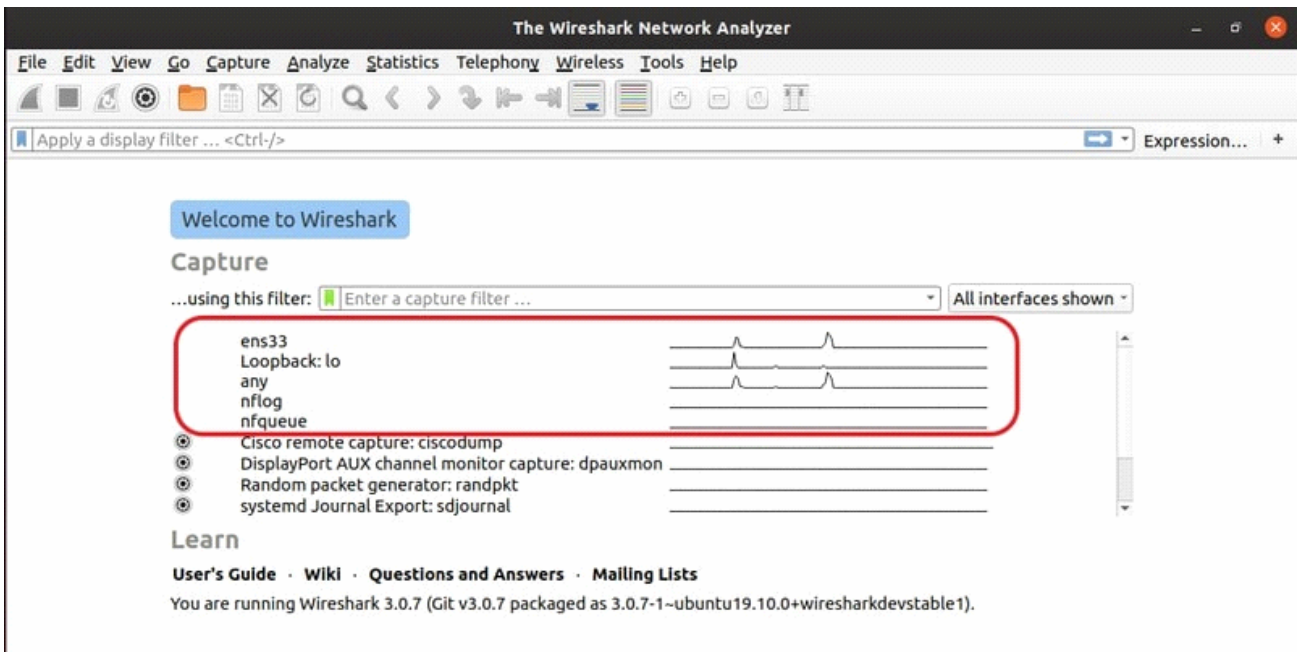
Finally, Start the wireshark program from the terminal:

```
Sudo  wireshark
```

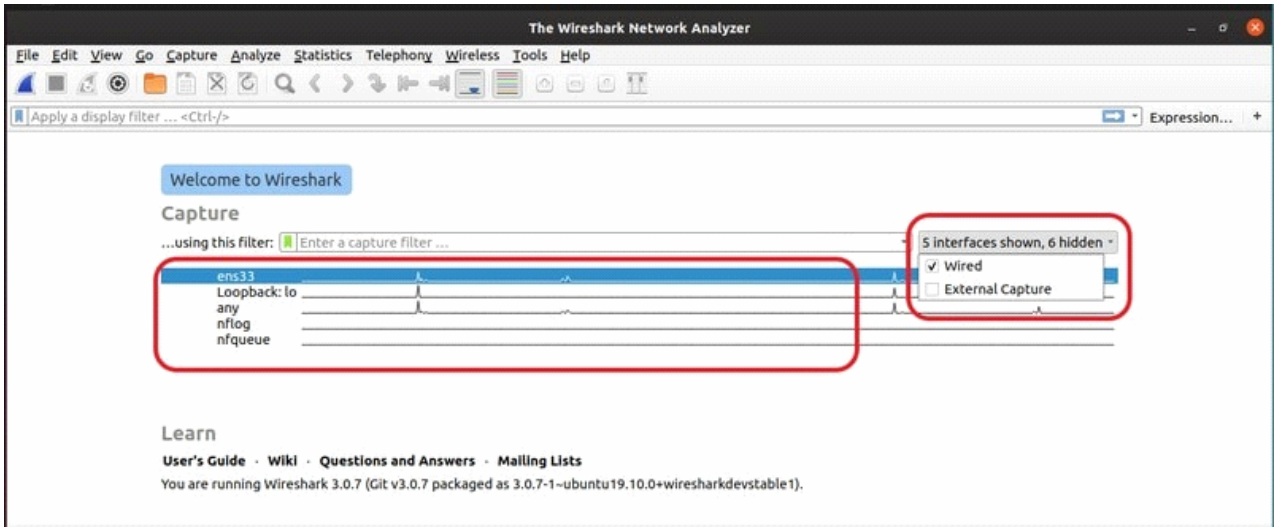
Capturing packets using Wireshark

When you start Wireshark, you will see a list of interfaces that you can use to capture packets to and from.

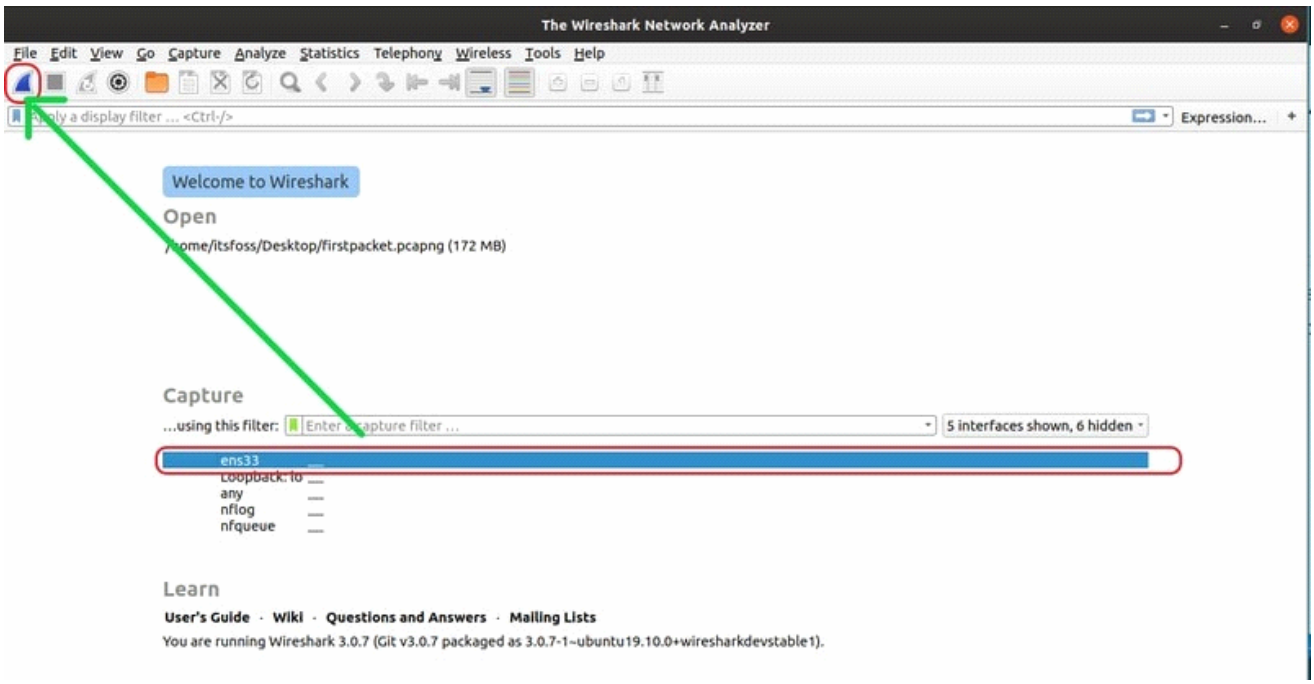
There are many types of interfaces available which you can monitor using Wireshark such as, Wired, External devices, etc. According to your preference, you can choose to show specific types of interfaces in the welcome screen from the marked area in the given image below.



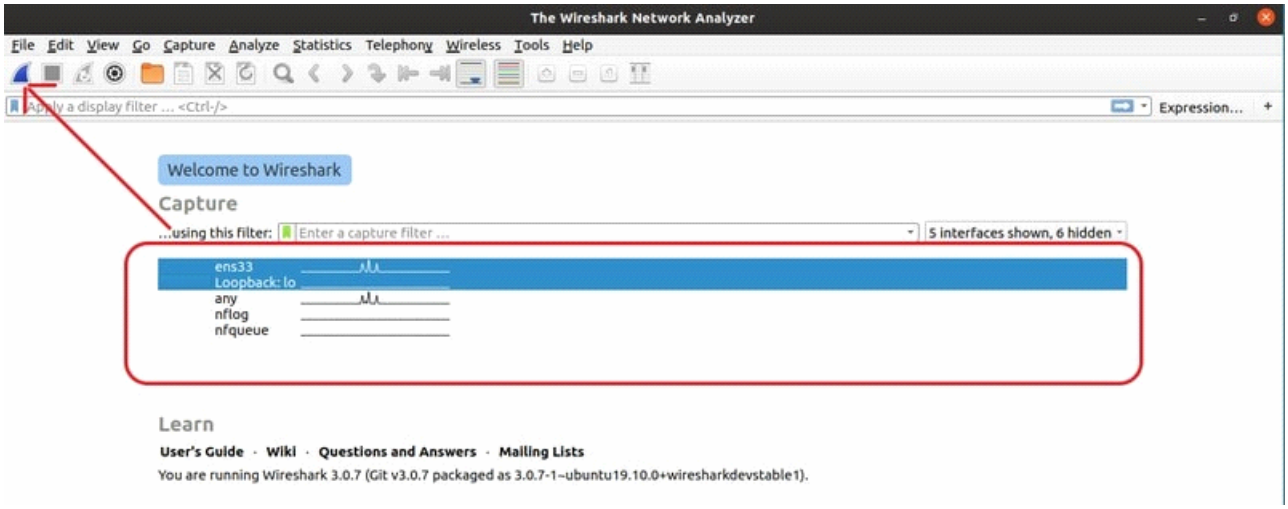
For instance, I listed only the **Wired** network interfaces.



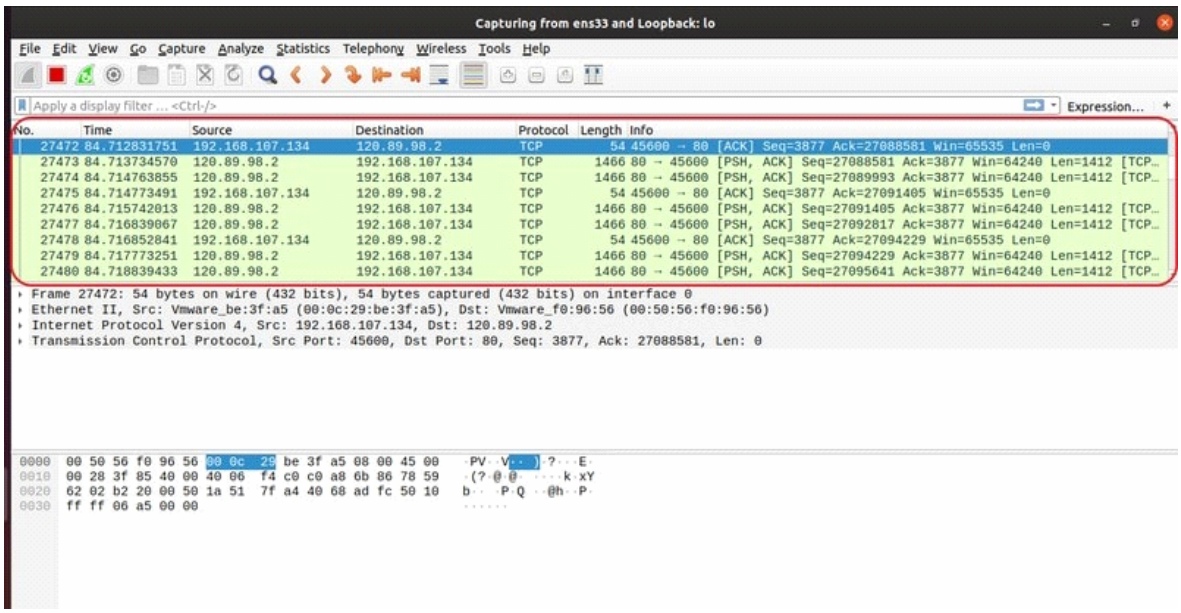
Next, to start capturing packets, you have to select the and click on the **Start capturing packets** icon as marked in the image below.



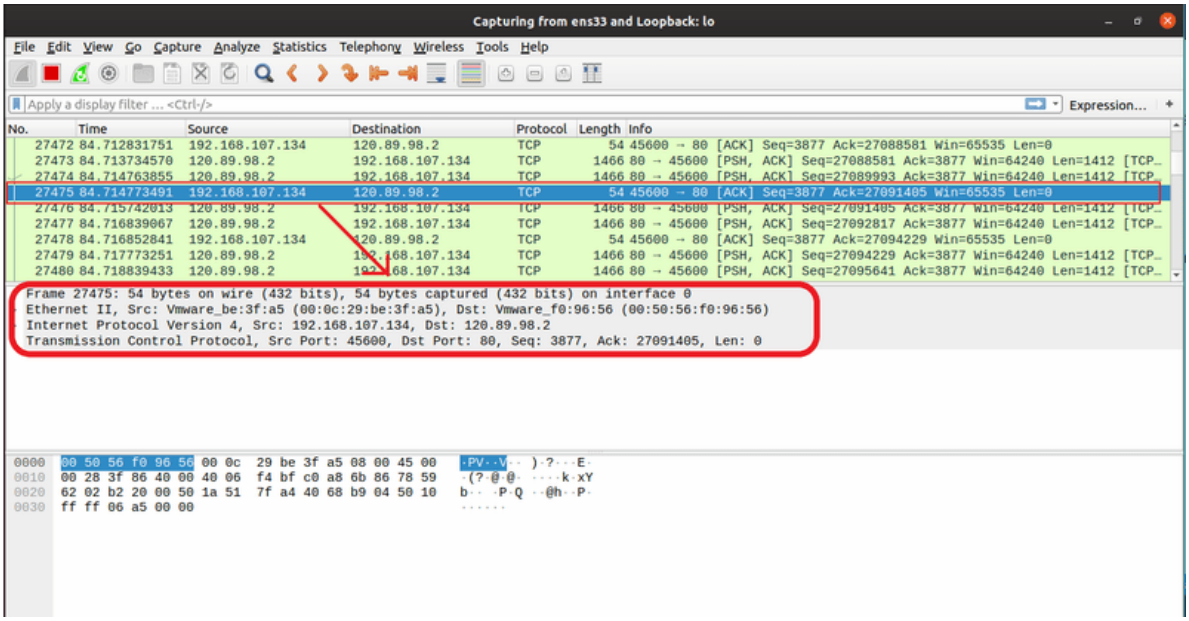
You can also capture packets to and from multiple interfaces at the same time. Just press and hold the **CTRL** button while clicking on the interfaces that you want to capture to and from and then hit the **Start capturing packets** icon as marked in the image below.



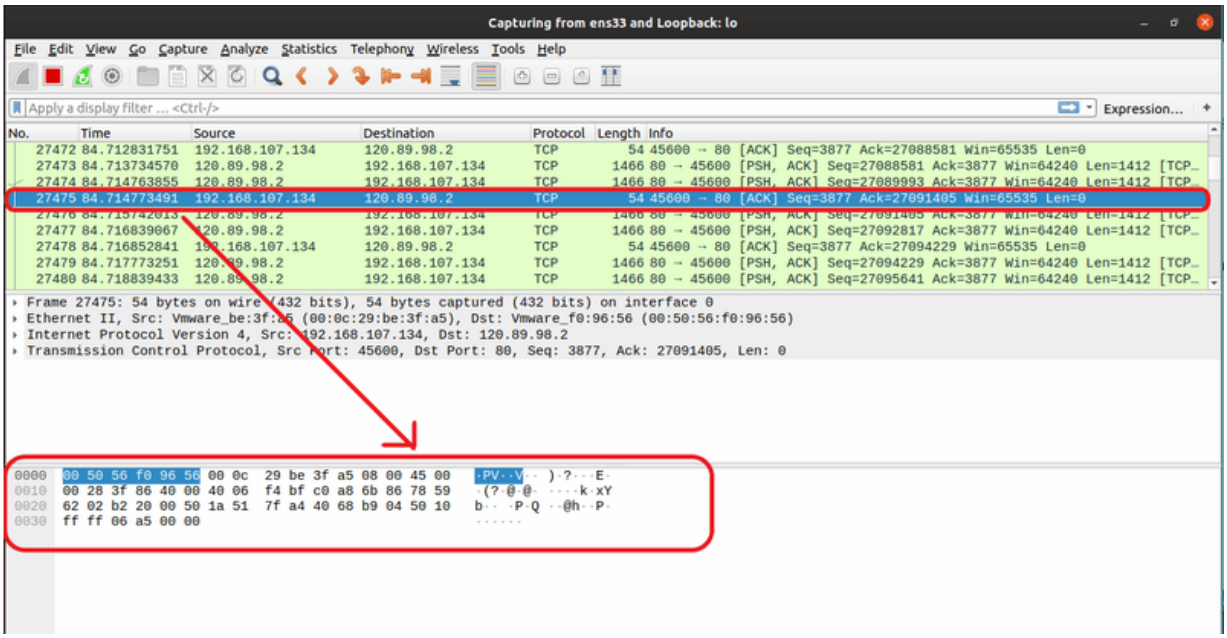
Next using **ping google.com** command in the terminal and as you can see, many packets were captured.



Now you can select on any packet to check that particular packet. After clicking on a particular packet you can see the information about different layers of TCP/IP Protocol associated with it.

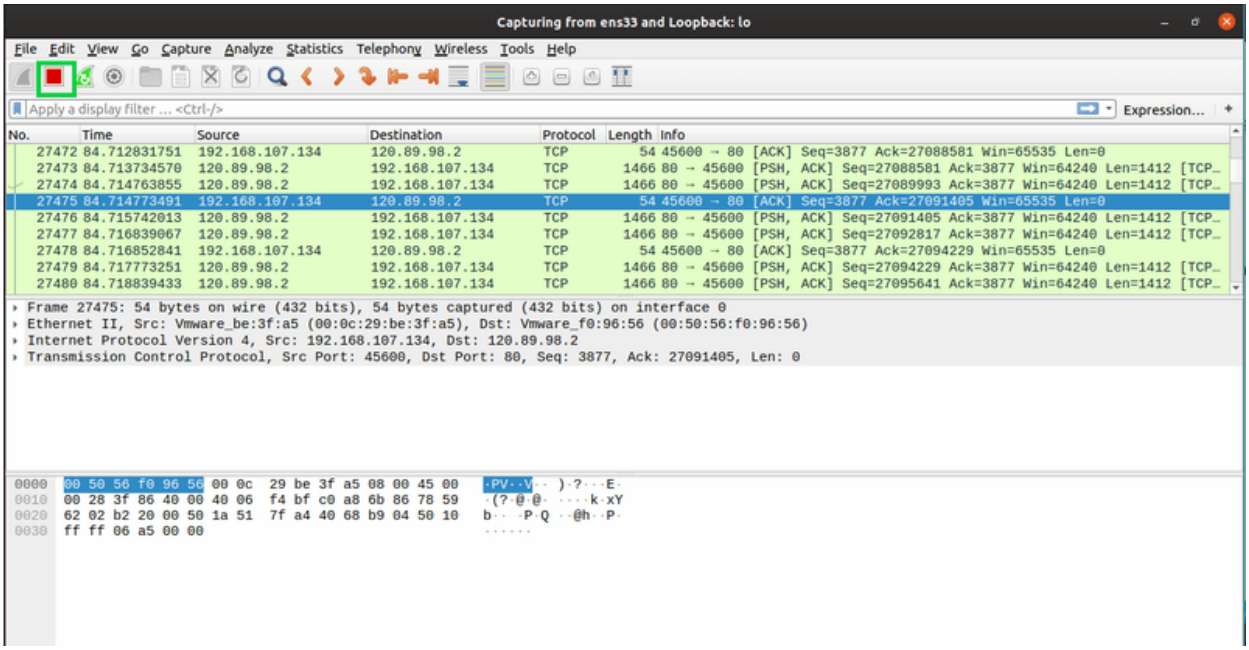


You can also see the RAW data of that particular packet at the bottom as shown in the image below.



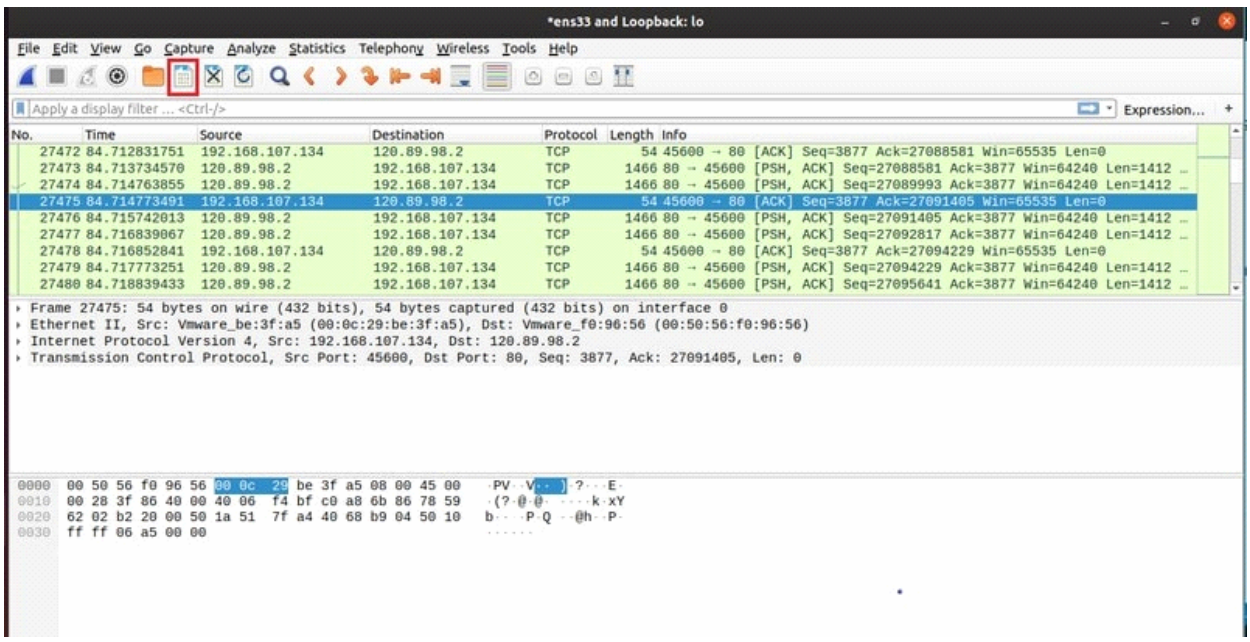
Stopping packet capture in Wireshark

You can click on the red icon as marked in the given image to stop capturing Wireshark packets.



Save captured packets to a file

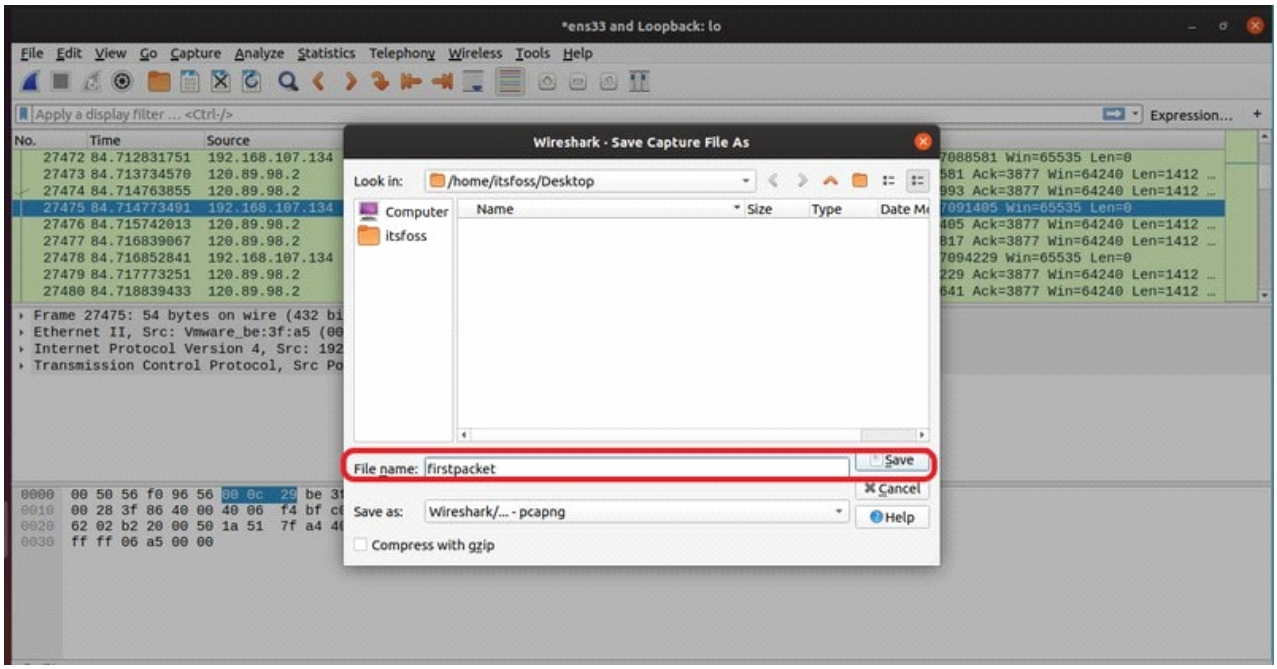
You can click on the marked icon in the image below to save captured packets to a file for future use.



Note: *Output can be exported to XML, PostScript®, CSV, or plain text.*

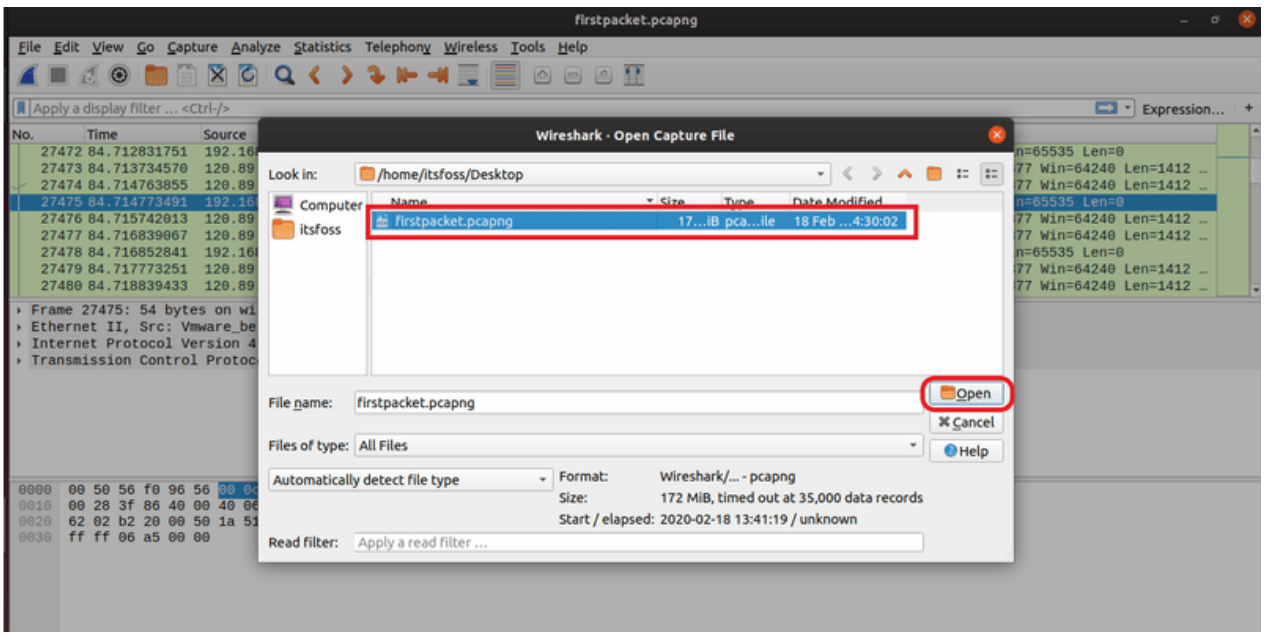
Next, select a destination folder, and type the file name and click on **Save**.

Then select the file and click on **Open**.



Now you can open and analyze the saved packets anytime. To open the file, press \ + **o** or go to **File > Open** from Wireshark.

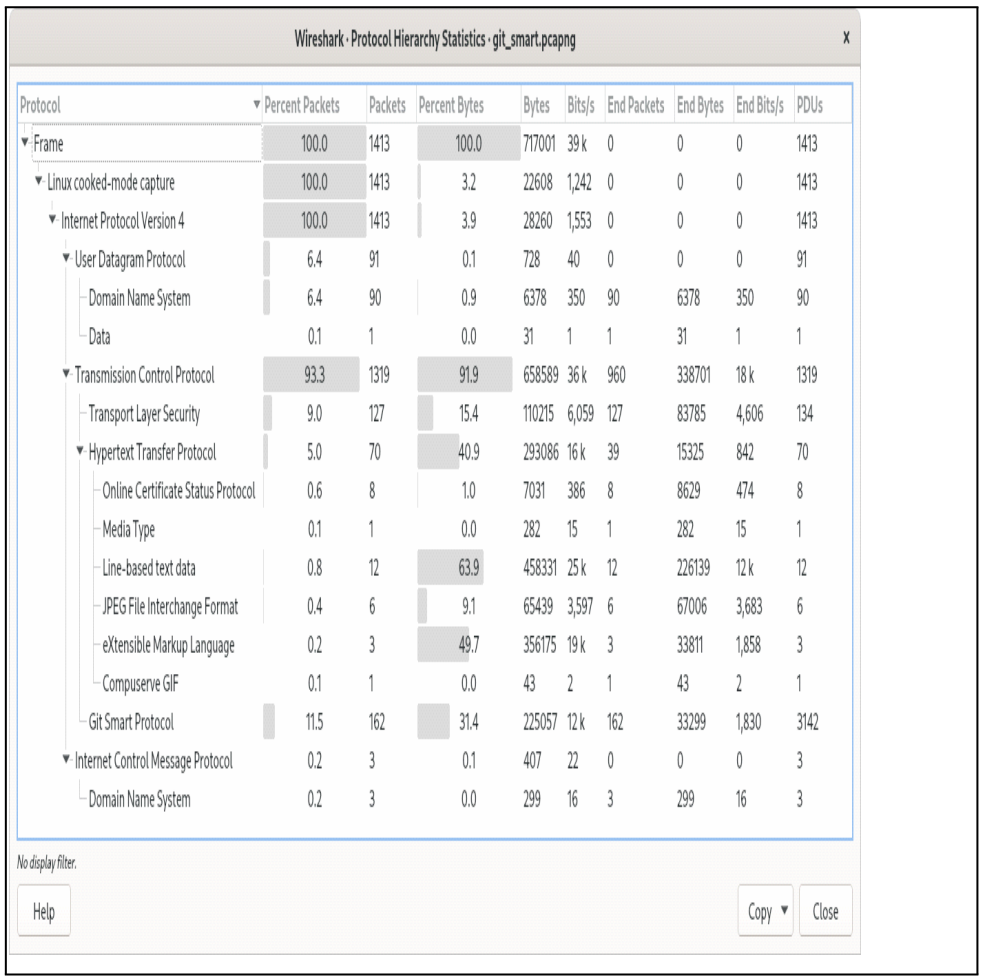
The captured packets should be loaded from the file.



The "Protocol Hierarchy"

The protocol hierarchy of the captured packets.

The "Protocol Hierarchy" Window



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	1413	100.0	717001	39 k	0	0	0	1413
Linux cooked-mode capture	100.0	1413	3.2	22608	1,242	0	0	0	1413
Internet Protocol Version 4	100.0	1413	3.9	28260	1,553	0	0	0	1413
User Datagram Protocol	6.4	91	0.1	728	40	0	0	0	91
Domain Name System	6.4	90	0.9	6378	350	90	6378	350	90
Data	0.1	1	0.0	31	1	1	31	1	1
Transmission Control Protocol	93.3	1319	91.9	658589	36 k	960	338701	18 k	1319
Transport Layer Security	9.0	127	15.4	110215	6,059	127	83785	4,606	134
Hypertext Transfer Protocol	5.0	70	40.9	293086	16 k	39	15325	842	70
Online Certificate Status Protocol	0.6	8	1.0	7031	386	8	8629	474	8
Media Type	0.1	1	0.0	282	15	1	282	15	1
Line-based text data	0.8	12	63.9	458331	25 k	12	226139	12 k	12
JPEG File Interchange Format	0.4	6	9.1	65439	3,597	6	67006	3,683	6
eXtensible Markup Language	0.2	3	49.7	356175	19 k	3	33811	1,858	3
Compuserve GIF	0.1	1	0.0	43	2	1	43	2	1
Git Smart Protocol	11.5	162	31.4	225057	12 k	162	33299	1,830	3142
Internet Control Message Protocol	0.2	3	0.1	407	22	0	0	0	3
Domain Name System	0.2	3	0.0	299	16	3	299	16	3

This is a tree of all the protocols in the capture. Each row contains the statistical values of one protocol. Two of the columns (*Percent Packets* and *Percent Bytes*) serve double duty as bar graphs. If a display filter is set it will be shown at the bottom.

The Copy button will let you copy the window contents as CSV or YAML.

Protocol hierarchy columns

Protocol

This protocol's name.

Percent Packets

The percentage of protocol packets relative to all packets in the capture.

Packets

The total number of packets that contain this protocol.

Percent Bytes

The percentage of protocol bytes relative to the total bytes in the capture.

Bytes

The total number of bytes of this protocol.

Bits/s

The bandwidth of this protocol relative to the capture time.

End Packets

The absolute number of packets of this protocol where it was the highest protocol in the stack (last dissected).

End Bytes

The absolute number of bytes of this protocol where it was the highest protocol in the stack (last dissected).

End Bits/s

The bandwidth of this protocol relative to the capture time where was the highest protocol in the stack (last dissected).

PDUs

The total number of PDUs of this protocol.

Packets usually contain multiple protocols. As a result, more than one protocol will be counted for each packet. Example: In the screenshot 100% of packets are IP and 99.3% are TCP (which is together much more than 100%).

Protocol layers can consist of packets that won't contain any higher layer protocol, so the sum of all higher layer packets may not sum to the protocol's packet count. This can be caused by segments and fragments reassembled in other frames, TCP protocol overhead, and other undissected data. Example: In the screenshot 99.3% of the packets are TCP but the sum of the subprotocols (TLS, HTTP, Git, etc.) is much less.

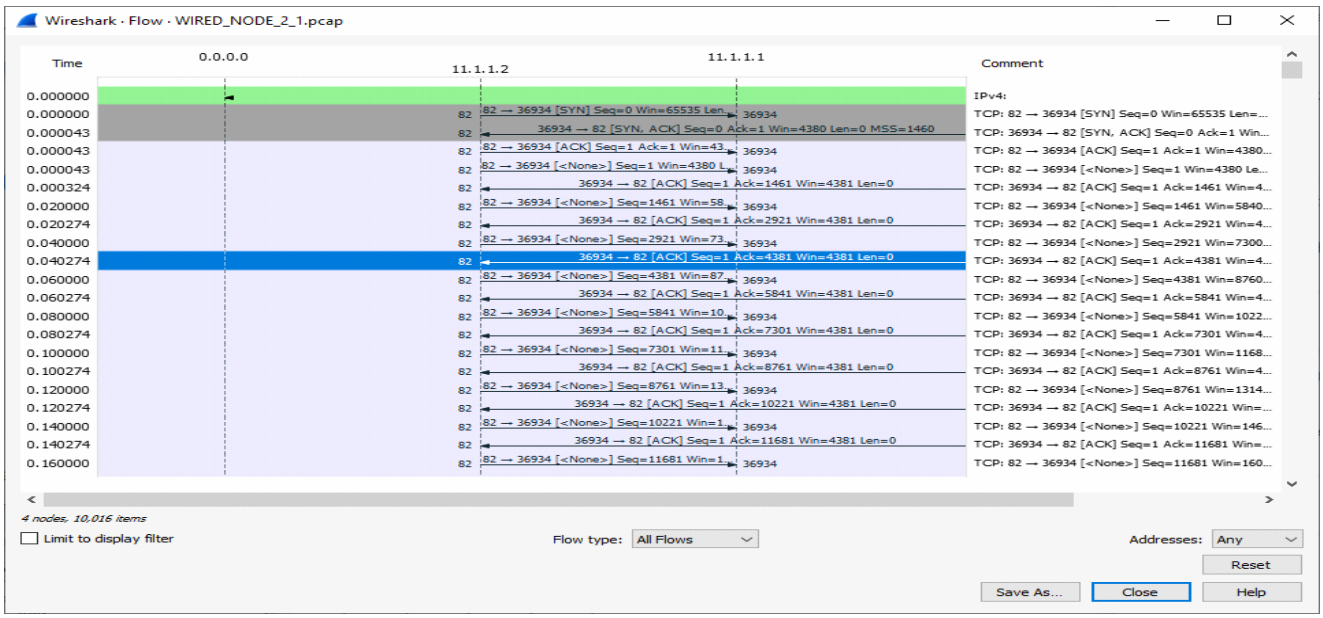
A single packet can contain the same protocol more than once. In this case, the entry in the PDUs column will be greater than that of Packets. Example: In the screenshot there are many more TLS and Git PDUs than there are packets.

Flow Graph

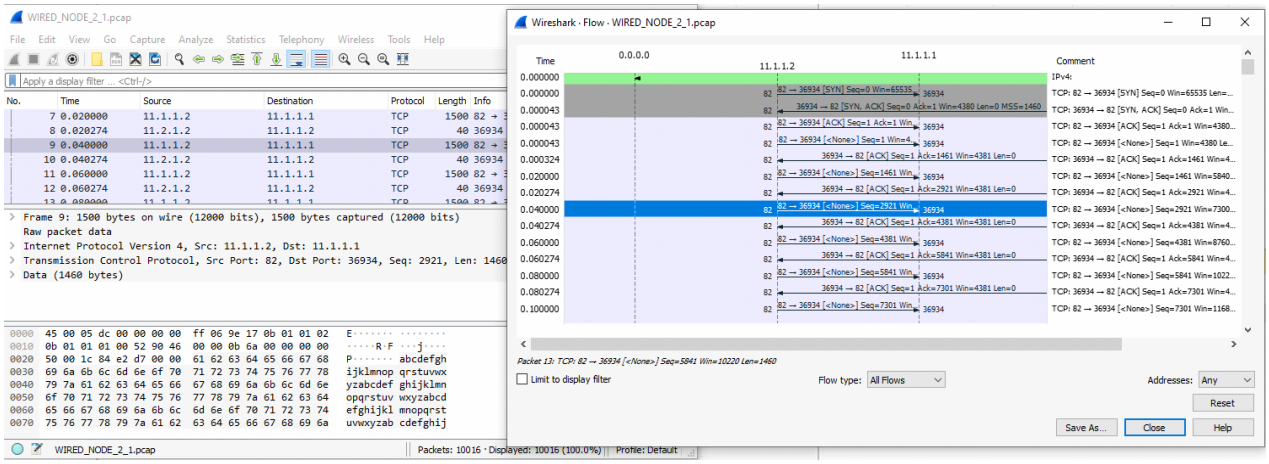
The Flow Graph window shows connections between hosts. It displays the packet time, direction, ports and comments for each captured connection. You can filter all connections by ICMP Flows, ICMPv6 Flows, UIM Flows and TCP Flows. Flow Graph window is used for showing multiple different topics. Based on it, it offers different controls.

Go to Statistics from the WireShark Window -> Choose Flow Graph -> Select flow type either All flows or TCP flows.

Mouse over any packet user will get the packet information in the lower panel

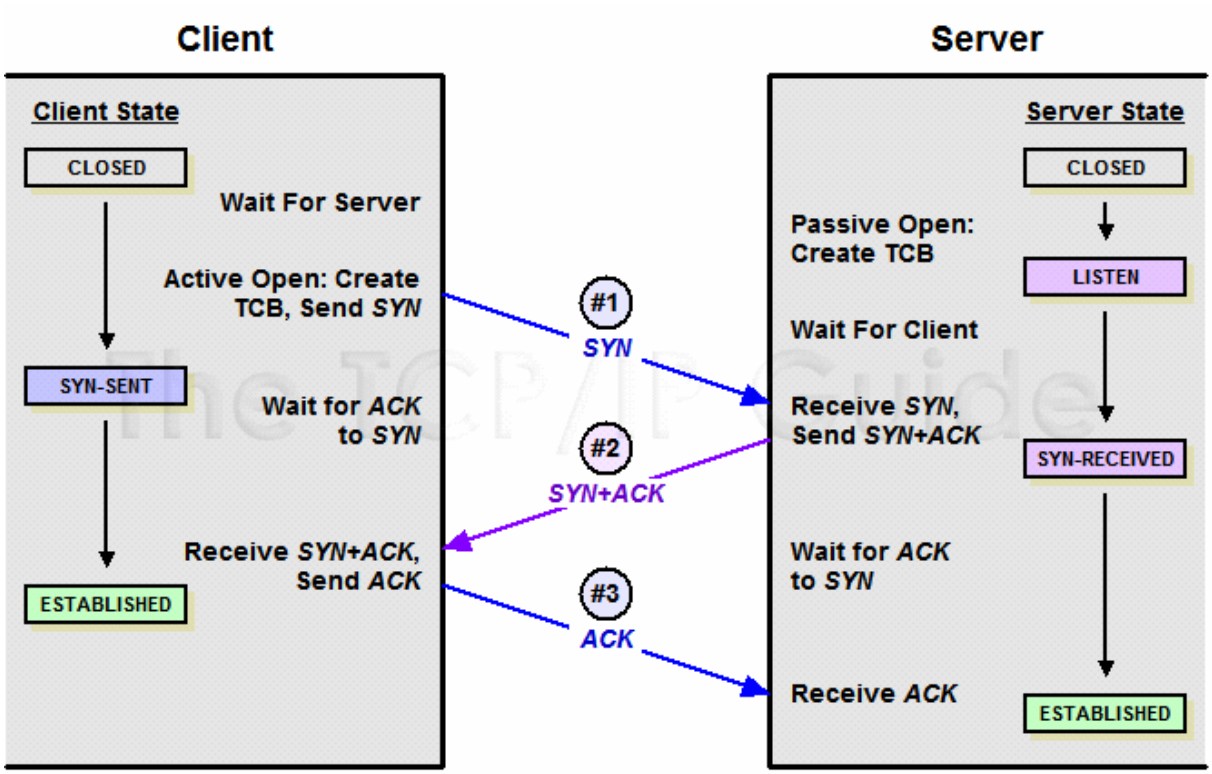


Select any packet that packet is automatically highlighted in the main wireshark window



TCP_3_way_handshaking

We assume that both client and server side start from CLOSED status.



1. The server process create a *TCB* [1] and use *TCB* prepares to accept the clients request. After **TCB** born the server change status to LISTEN.

2. The host does the same thing, create a *TCB* and use this *TCB* to send request, set the "SYN=1" in the request header, and initiates a arbitrary sequence number, *seq=x*. SYN packet (which means SYN=1) can not take any data content, but it will **consume a sequence number**. After request sent, the client goes into SYN-SENT status.

3. After receiving the clients request:

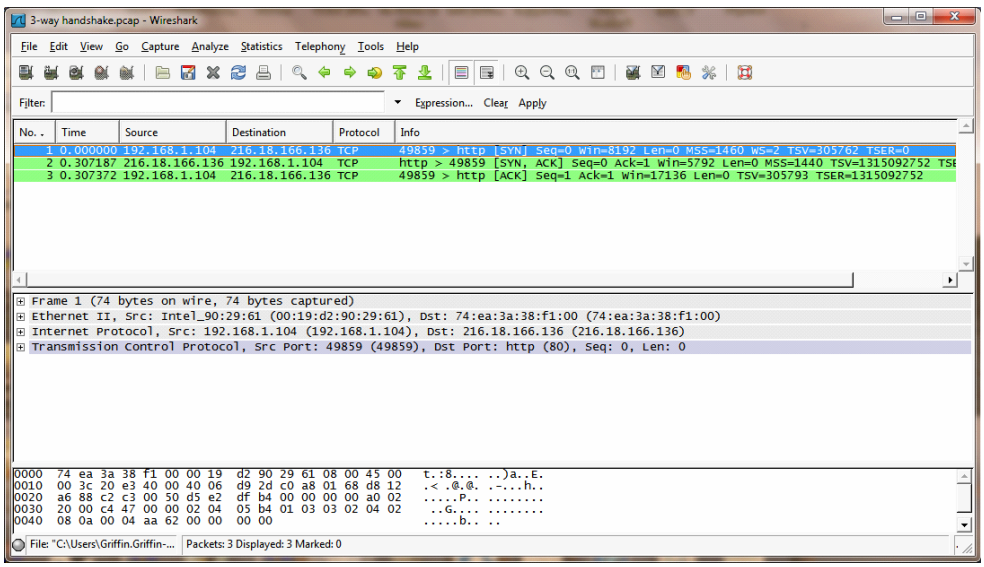
   i. If the server accept to this connection, it will send back a confirm response. In the response both SYN and ACK bits should be '1', and server side also initiates a SEQ number, *seq=y*. The server will send its sequence number within packet which is used to be acknowledged to the client's SYN packet. This packet can not take any data content either, but it **consumes a sequence number**. So in this packet *seq=y, ack=x+1*. And the server goes into SYN-RCVD status.

   ii. If the server rejects the connection, it just responses a RST packet to reset the connection.

4. After the client received the server's response, it will send back also a confirm packet with ACK bit sets to '1' and **seq=x+1, ack=y+1**. [2]

After that, both side goes into ESTABLISHED status. This is what we called three-way handshake.

Example



EXPERIMENT :12

Study of NS2 simulator

AIM:-

To study NS2 simulator

Introduction to Network Simulator 2 (NS2)

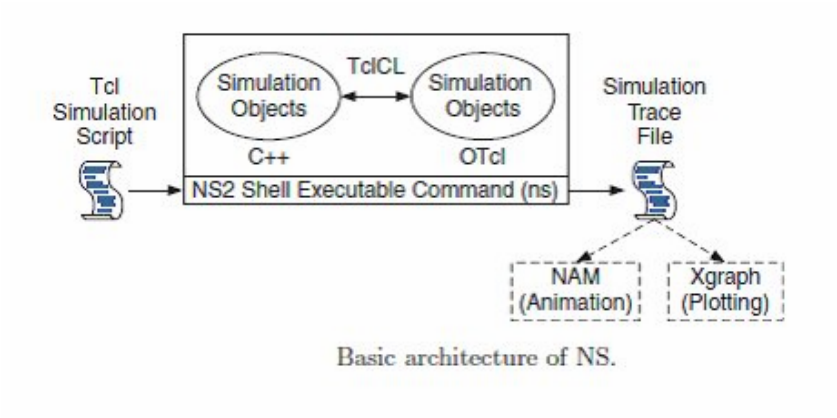
Network Simulator (Version 2), widely known as NS2, is simply an event-driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors.

Features of NS2

1. It is a discrete event simulator for networking research.
2. It provides substantial support to simulate bunch of protocols like TCP, FTP, UDP, https and DSR.
3. It simulates wired and wireless network.
4. It is primarily Unix based.
5. Uses TCL as its scripting language.
6. Otcl: Object oriented support
7. Tclcl: C++ and otcl linkage
8. Discrete event scheduler

3. Basic Architecture

NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events. The C++ and the OTcl are linked together using TclCL



Basic architecture of NS.

Nam is also needed to install. Nam (**Network Animator**) is an animation tool to graphically represent the network and packet traces.

To install NS-2 using following commands:

sudo apt-get install ns2

sudo apt-get install nam

sudo apt install tclsh

ns

nam

NS2 simulation using Distance Vector routing protocol

dist2.tcl

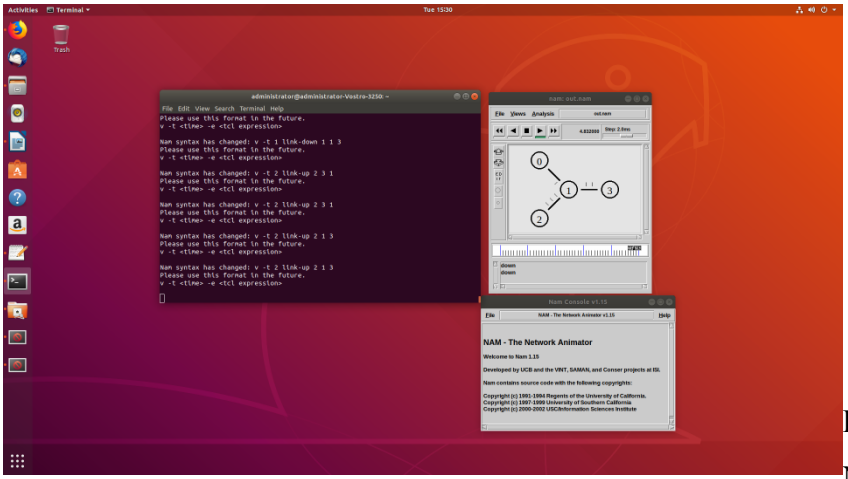
```
set ns [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf

set tr [open out.tr w]
$ns trace-all $tr
```

```
proc finish {} {
global nf ns tr
$ns flush-trace
close $tr
exec nam out.nam &
 exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n1 orient right-up

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink

set udp [new Agent/UDP]
```

```
$ns attach-agent $n2 $udp

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp

set null [new Agent/Null]
$ns attach-agent $n3 $null

$ns connect $tcp $sink
$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto DV

$ns at 0.0 "$ftp start"
$ns at 0.0 "$cbr start"

$ns at 5.0 "finish"
$ns run
```

OUTPUT

```
ns dist2.tcl
```



EXPERIMENT-13

Design and configure a Local Area Network (LAN)

AIM:-

To design and configure a LAN

About Networking

Two or more computers together with the ability to communicate with each other. Networking is to link two or more computing devices together for the purpose of sharing data.

It provides design, programming, development and operational support for LANs, WANs and other networks. A local area network (LAN) is a group of computers and associated devices that share a common communications line or wireless link.

Requirement:

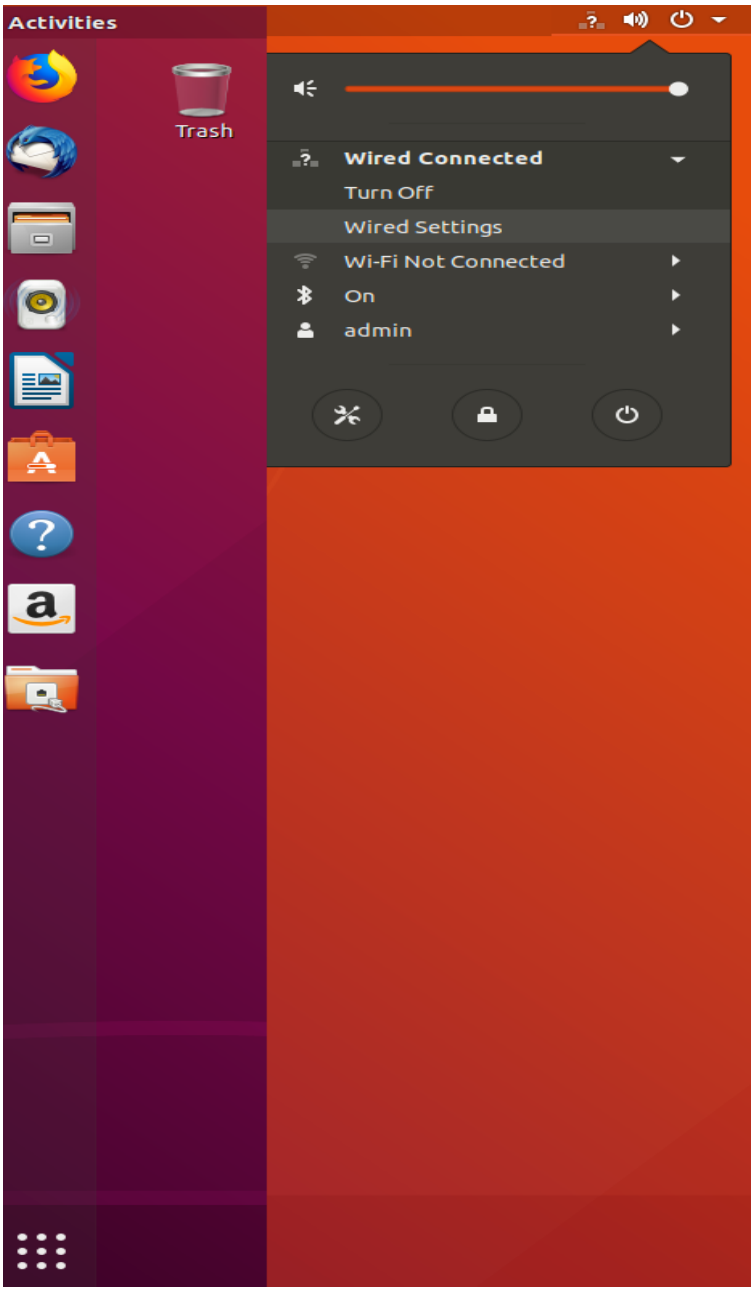
To set up a LAN, you will need:

- A network switch
- Ethernet cables
- Computers

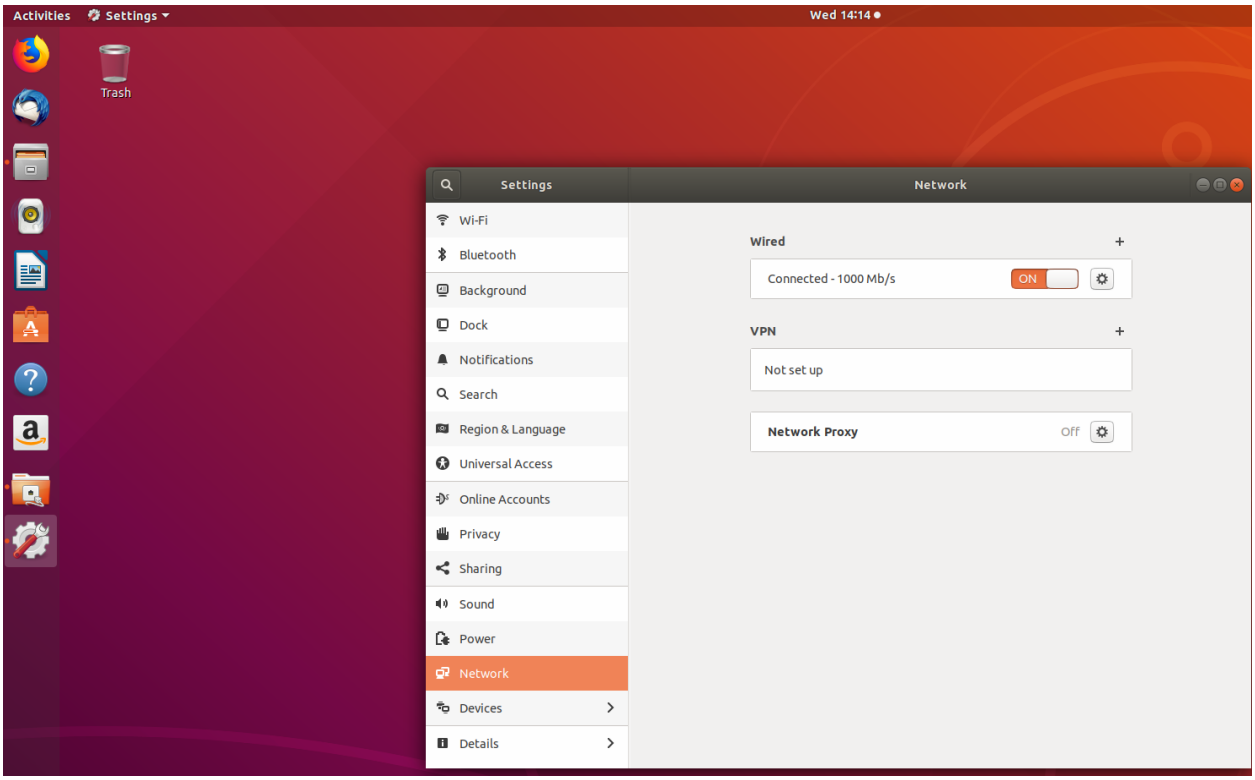
Procedure

Here we contruct a **LAN** with **Ring** toplogy

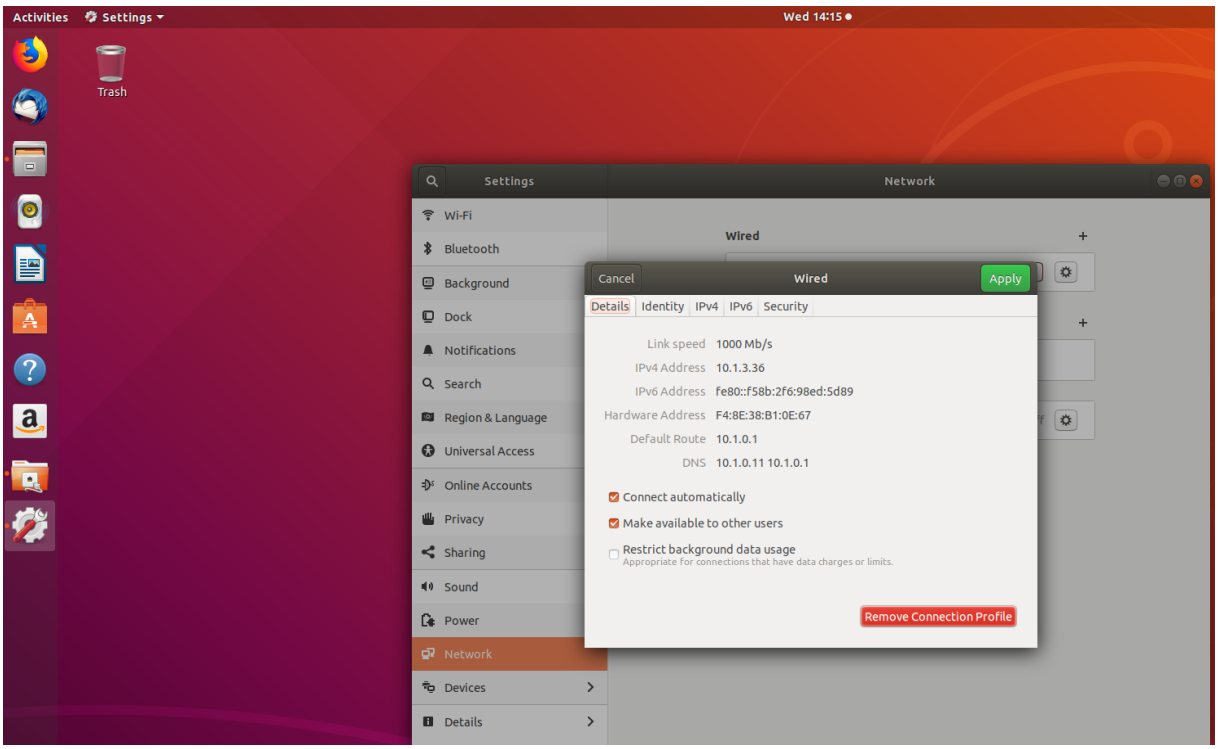
- Connect each Computers with Switch using Ethernet cables
- Power on Switch and Computers
- Power on each computer in Ubuntu
- Log on to Administrator.
- Go to screen right-side top and click on wired settings from wired connections


Activities
Trash
Wired Connected
Turn Off
Wired Settings
Wi-Fi Not Connected
On
admin

Click on the settings button

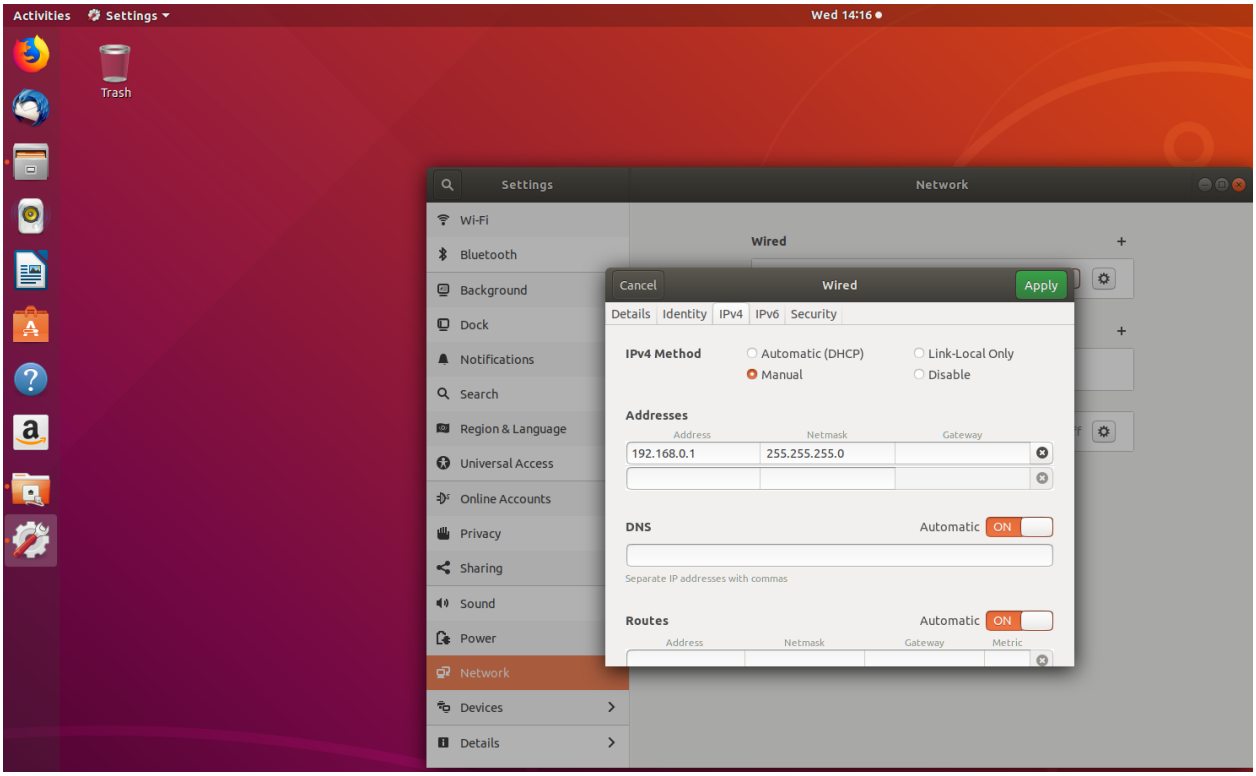


Select IPv4 menu item

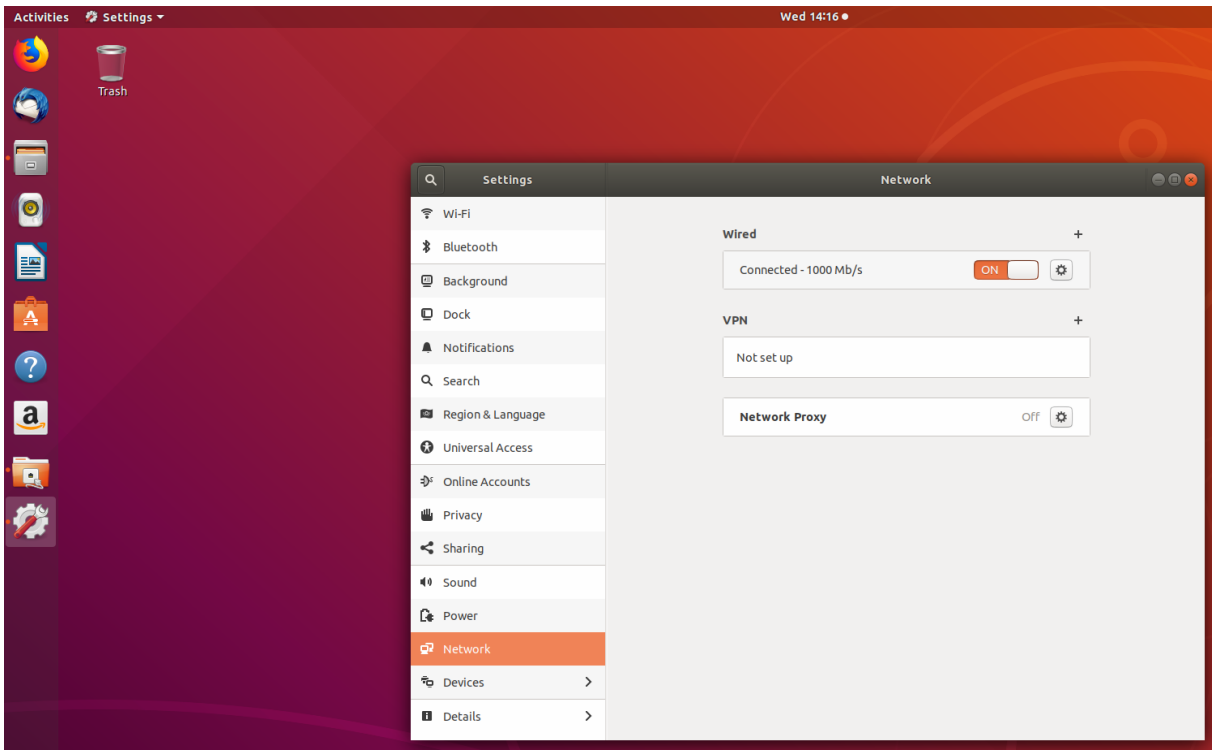


Change IPv4 method as Manual and set address as 192.168.0.x(x may vary 1,2,3,4…..)

Click on Apply



Refresh the wired connection button OFF and ON and close the window



Network is ready. Check it with ping commands from terminal