METHODIST

COLLEGE OF ENGINEERING ANDTECHNOLOGY

Estd:2008

(Affiliated to Osmania University & Approved by AICTE, New Delhi)

LABORATORY MANUAL

COMPUTER NETWORKS

LABORATORY

BE IV Semester (AICTE Model Curriculum): 2022-23

NAME: _____

ROLL NO: _____

BRANCH: _____ **SEM:** _____

DEPARTMENT OF COMPUTERSCIENCE & ENGINEERNG

Empower youth- Architects of Future World

METHODIST

Estd:2008 COLLEGE OF ENGINEERING ANDTECHNOLOGY

VISION

To produce ethical, socially conscious and innovative professionals who would contribute to sustainable technological development of the society.

MISSION

To impart quality engineering education with latest technological developments and interdisciplinary skills to make students succeed in professional practice.

To encourage research culture among faculty and students by establishing state of art laboratories and exposing them to modern industrial and organizational practices.

To inculcate humane qualities like environmental consciousness, leadership, social values, professional ethics and engage in independent and lifelong learning for sustainable contribution to the society.

METHODIST

Estd:2008 COLLEGE OF ENGINEERING ANDTECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY MANUAL

COMPUTER NETWORKSLABORATORY

Prepared By

Ms.Sana Mateen

Assistant Professor, Dept. of CSE

Estd:2008

METHODIST

COLLEGE OF ENGINEERING ANDTECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION & MISSION

VISION

To become a leader in providing Computer Science & Engineering education with emphasis on knowledge and innovation.

MISSION

- To offer flexible programs of study with collaborations to suit industry needs.
- To provide quality education and training through novel pedagogical practices.
- To expedite high performance of excellence in teaching, research and innovations.
- To impart moral, ethical values and education with social responsibility.

Estd:2008

METHODIST

COLLEGE OF ENGINEERING ANDTECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND

PROGRAM EDUCATIONAL OBJECTIVES

After 3-5 years of graduation, the graduates will be able to

PEO1: Apply technical concepts, Analyze, synthesize data to Design and create novel products and solutions for the real life problems.

PEO2: Apply the knowledge of Computer Science Engineering to pursue higher education with due consideration to environment and society.

PEO3: Promote collaborative learning and spirit of team work through multidisciplinary projects

PEO4: Engage in life-long learning and develop entrepreneurial skills.

METHODIST

Estd:2008

COLLEGE OF ENGINEERING ANDTECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND

PROGRAM OUTCOMES

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 modelling 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.

PROGRAM SPECIFIC OUTCOMES

At the end of 4 years, Computer Science and Engineering graduates at MCET will be able to:

PSO1: Apply the knowledge of Computer Science and Engineering in various domains like networking and data mining to manage projects in multidisciplinary environments.

PSO2: Develop software applications with open-ended programming environments**.**

PSO3: Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms

COMPUTER NETWORKSLAB

Semester–IV	L	T	P	Credits
Subjectcode-3PC456CS	0	0	2	1

Course Objectives:	Course Outcomes:
➢ Learn to communicate between two desktops ➢ Learn to implement different protocols ➢ Be familiar with socket programming ➢ Be familiar with various routing algorithms ➢ Be familiar with different simulation tools ➢ Use simulation tools to analyze performance of various network Protocols	1. Demonstrate a broad knowledge of the area of computer networking and its terminology 2. Understand to configure intermediary network devices 3. Program using sockets 4. Use simulation tools to analyze the performance of various network protocols 5. Implement and analyze various routing algorithms.

List of Programs

1. Running and using services/commands like tcp dump, netstat, ifconfig, nslookup, FTP, TELNET and trace route. Capture ping and trace route PDUs using network protocol analyzer andexamine

2. Implement the data link layer framing methods such as character-stuffing and bit stuffing

3. Implement on a dataset of characters using CRC polynomials CRC12 and CRC16

4. Initial Configuration of router and switch (using real devices or simulators)

5. Designandimplementthefollowingexperimentsusingpackettracersoftware
   I. Simulation of network topologies
   II. Configuration of network using different routing protocols

6. Do the following using NS2/NS3/NetSim or any other equivalent tool
   I. Simulation of Congestion Control Algorithms
   II. Simulation of Routing Algorithms

7. Socket programming using UDP and TCP (e.g simple DNS, date & time client/server, echo client/server, iterative & concurrent servers)

8. Programming using RPC

METHODIST

Estd:2008

COLLEGE OF ENGINEERING ANDTECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Outcomes (CO's):

SUBJECT NAME: COMPUTER NETWORKS LAB

CODE: 3PC456CS

SEMESTER: IV

CO No.	Course Outcome	Taxonomy Level
3PC456CS.1	Understand the network services/commands for network troubleshooting and management..	Understand
3PC456CS.2	Apply appropriate network analysis techniques to capture and examine network traffic using a network protocol analyzer	Apply
3PC456CS.3	Implement datalink layer framing methods and evaluate data integrity using CRC polynomials.	Apply
3PC456CS.4	Design and simulate network topologies using software tools like Packet Tracer to analyze and optimize network configurations.	Create
3PC456CS.5	Simulate and analyze the performance of various congestion control algorithms using tools like NS2/NS3/NetSim.	Analyze
3PC456CS.6	design and implement client-server applications using socket programming, enabling communication and data exchange between clients and servers	Create

METHODIST

Estd:2008

COLLEGE OF ENGINEERING ANDTECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the program / experiment details.
3. Student should enter into the laboratory with:
    a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
    b. Laboratory Record updated up to the last session experiments.
    c. Formal dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out. If anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

METHODIST

Estd:2008

COLLEGE OF ENGINEERING ANDTECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CODE OF CONDUCT FOR THE LABORATORY

- All students must observe the dress code while in the laboratory
- Footwear is NOT allowed
- Foods, drinks and smoking are NOT allowed
- All bags must be left at the indicated place
- The lab timetable must be strictly followed
- Be PUNCTUAL for your laboratory session
- All programs must be completed within the given time
- Noise must be kept to a minimum
- Workspace must be kept clean and tidy at all time
- All students are liable for any damage to system due to their own negligence
- Students are strictly PROHIBITED from taking out any items from the laboratory
- Report immediately to the lab programmer if any damages to equipment

BEFORE LEAVING LAB:

- Arrange all the equipment and chairs properly.
- Turn off / shut down the systems before leaving.
- Please check the laboratory notice board regularly for updates.

Lab in – charge

METHODIST

Estd:2008

COLLEGE OF ENGINEERING ANDTECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LIST OF EXPERIMENTS

Sl. No.	Name of the Experiment	Date of Experiment	Date of Submission	Page No	Faculty Signature
1.	Running and using services/commands like tcp dump, netstat, ifconfig, nslookup, FTP,TELNET and trace route.				
2.	Implement the data link layer framing methods such as character-stuffing and bit stuffing				
3.	Implement on a dataset of characters using CRC polynomials CRC12 and CRC16				
4.	Initial Configuration of router and switch(using real devices or simulators)				
5.	Designandimplementthefollowingexperimentsusingpackettracersoftware I. Simulation of network topologies II. Configuration of network using different routing protocols				
6.	Do the following using NS2/NS3/NetSim or any other equivalent tool I. Simulation of Congestion Control Algorithms II. Simulation of Routing Algorithms				
7.	Socket programming using UDP and TCP (e.g simple DNS, date & time client/server, echo client/server, iterative & concurrent servers)				
8	Programming using RPC				

1. Running and using services/commands like tcpdump, netstat, ifconfig, nslookup, FTP,TELNET and trace route. Capture ping and trace route PDUs using network protocol analyzer and examine

tcpdump

The tcpdump utility allows you to capture packets that flow within your network to assist in network troubleshooting. The following are several examples of using tcpdump with different options. Traffic is captured based on a specified filter.

```
sana@sana-virtual-machine:~$ sudo tcpdump host 192.168.0.111 > tcphost.txt
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens33, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C21 packets captured
21 packets received by filter
0 packets dropped by kernel
```

```
sana@sana-virtual-machine:~$ sudo tcpdump src 192.168.217.129 > tcpsrc.txt
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens33, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C3646 packets captured
3665 packets received by filter
```

```
sana@sana-virtual-machine:~$ sudo tcpdump tcp > tcpdtcp.txt
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens33, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C231 packets captured
231 packets received by filter
0 packets dropped by kernel
```

```
sana@sana-virtual-machine:~$ sudo tcpdump src port 22  > sshtcpd.txt
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens33, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C42 packets captured
42 packets received by filter
0 packets dropped by kernel
```

```
sana@sana-virtual-machine:~$ sudo tcpdump tcp > tcpdtcp.txt
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on ens33, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C125 packets captured
133 packets received by filter
0 packets dropped by kernel
```

netstat

Netstat command displays various network related information such as network connections,

routing tables, interface statistics, masquerade connections, multicast memberships etc.

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

netstat -a | more : To show both listening and

non-listening sockets.

```
sana@sana-virtual-machine:~$ netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp        0      0 sana-virtual-mach:50478 36.75.98.34.bc.go:https TIME_WAIT
tcp        0      0 sana-virtual-mach:60676 ec2-52-35-3-113.u:https ESTABLISHED
tcp        0      0 sana-virtual-mach:36528 kazooie.canonical.:http TIME_WAIT
tcp        0      0 sana-virtual-mach:35748 150.9.241.35.bc.g:https ESTABLISHED
tcp        0      0 sana-virtual-mach:48556 191.144.160.34.bc:https TIME_WAIT
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN
udp        0      0 0.0.0.0:34520           0.0.0.0:*
udp        0      0 localhost:domain        0.0.0.0:*
udp        0      0 localhost:domain        0.0.0.0:*
udp        0      0 sana-virtual-mac:bootpc 192.168.217.254:bootps  ESTABLISHED
udp        0      0 0.0.0.0:mdns            0.0.0.0:*
udp        0      0 0.0.0.0:631             0.0.0.0:*
udp6       0      0 [::]:mdns               [::]:*
udp6       0      0 [::]:44681              [::]:*
raw6       0      0 [::]:ipv6-icmp          [::]:*                  7
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ACC ]     STREAM     LISTENING     31487    @/tmp/.ICE-unix/1169
unix  2      [ ACC ]     STREAM     LISTENING     28081    /run/systemd/resolve/io.systemd.Resolve
unix  2      [ ACC ]     STREAM     LISTENING     48748    @/tmp/dbus-xF4RI2VC
unix  2      [ ACC ]     STREAM     LISTENING     32923    @/tmp/.X11-unix/X0
unix  2      [ ACC ]     STREAM     LISTENING     32925    @/tmp/.X11-unix/X1
unix  2      [ ACC ]     STREAM     LISTENING     31488    /tmp/.ICE-unix/1169
unix  2      [ ACC ]     STREAM     LISTENING     32924    /tmp/.X11-unix/X0
unix  2      [ ACC ]     STREAM     LISTENING     32926    /tmp/.X11-unix/X1
unix  2      [ ]         DGRAM                    31239    /run/user/1000/systemd/notify
unix  2      [ ACC ]     STREAM     LISTENING     31242    /run/user/1000/systemd/private
```

2. Listing TCP Ports connections

Listing only **TCP** (**Transmission Control Protocol**) port connections using **netstat -at**.

 # netstat -at : To list all tcp ports.

```
sana@sana-virtual-machine:~$ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp        0      0 sana-virtual-mach:60676 ec2-52-35-3-113.u:https ESTABLISHED
tcp        0      0 sana-virtual-mach:58862 ec2-34-210-148-25:https ESTABLISHED
tcp        0      0 sana-virtual-mach:58850 ec2-34-210-148-25:https ESTABLISHED
tcp        0      0 sana-virtual-mach:43530 123.208.120.34.bc:https ESTABLISHED
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN
sana@sana-virtual-machine:~$
```

3. Listing UDP Ports connections

Listing only **UDP** (**User Datagram Protocol**) port connections using **netstat -au**.

 # netstat -au : To list all udp ports.

```
sana@sana-virtual-machine:~$ netstat -au
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 0.0.0.0:34520           0.0.0.0:*
udp        0      0 localhost:domain        0.0.0.0:*
udp        0      0 localhost:domain        0.0.0.0:*
udp        0      0 sana-virtual-mac:bootpc 192.168.217.254:bootps  ESTABLISHED
udp        0      0 0.0.0.0:mdns            0.0.0.0:*
udp        0      0 0.0.0.0:631             0.0.0.0:*
udp6       0      0 [::]:mdns               [::]:*
udp6       0      0 [::]:44681              [::]:*
sana@sana-virtual-machine:~$
```

4.Listing all LISTENING Connections

Listing all active listening ports connections with netstat -l.

```
sana@sana-virtual-machine:~$ sudo netstat -l
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN
udp        0      0 0.0.0.0:34520           0.0.0.0:*
udp        0      0 localhost:domain        0.0.0.0:*
udp        0      0 localhost:domain        0.0.0.0:*
udp        0      0 0.0.0.0:mdns            0.0.0.0:*
udp        0      0 0.0.0.0:631             0.0.0.0:*
udp6       0      0 [::]:mdns               [::]:*
udp6       0      0 [::]:44681              [::]:*
raw6       0      0 [::]:ipv6-icmp          [::]:*                  7
Active UNIX domain sockets (only servers)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ACC ]     STREAM     LISTENING     31487    @/tmp/.ICE-unix/1169
unix  2      [ ACC ]     STREAM     LISTENING     28081    /run/systemd/resolve/io.systemd.Resolve
unix  2      [ ACC ]     STREAM     LISTENING     32923    @/tmp/.X11-unix/X0
unix  2      [ ACC ]     STREAM     LISTENING     32925    @/tmp/.X11-unix/X1
unix  2      [ ACC ]     STREAM     LISTENING     31488    /tmp/.ICE-unix/1169
unix  2      [ ACC ]     STREAM     LISTENING     32924    /tmp/.X11-unix/X0
unix  2      [ ACC ]     STREAM     LISTENING     32926    /tmp/.X11-unix/X1
unix  2      [ ACC ]     STREAM     LISTENING     31242    /run/user/1000/systemd/private
unix  2      [ ACC ]     STREAM     LISTENING     31775    /run/user/1000/bus
unix  2      [ ACC ]     STREAM     LISTENING     31777    /run/user/1000/gnupg/S.dirmngr
unix  2      [ ACC ]     STREAM     LISTENING     50637    @/tmp/dbus-d397I9cN
```

ifconfig

```
sana@sana-virtual-machine:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.217.141  netmask 255.255.255.0  broadcast 192.168.217.255
        inet6 fe80::e95:816:ca44:915d  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:51:d3:bd  txqueuelen 1000  (Ethernet)
        RX packets 20091  bytes 23025216 (23.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6522  bytes 732852 (732.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 869  bytes 109000 (109.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 869  bytes 109000 (109.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

nslookup

Nslookup (stands for "Name Server Lookup") is a useful command for getting information from the DNS server. It is a network administration tool for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or any other specific DNS record. It is also used to troubleshoot DNS-related problems.

Syntax:

nslookup [option]

```
sana@sana-virtual-machine:~$ nslookup google.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   google.com
Address: 142.250.196.14
Name:   google.com
Address: 2404:6800:4007:829::200e
```

- **nslookup 192.168.217.141:** Reverse DNS lookup
  You can also do the reverse DNS look-up by providing the IP Address as an argument to nslookup.

```
sana@sana-virtual-machine:~$ nslookup  192.168.217.141
141.217.168.192.in-addr.arpa    name = sana-virtual-machine.
141.217.168.192.in-addr.arpa    name = sana-virtual-machine.local.
```

nslookup -type=any google.com : Lookup for any record

We can also view all the available DNS records using the -type=any option.

```
sana@sana-virtual-machine:~$ nslookup -type=any google.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   google.com
Address: 2404:6800:4007:829::200e
google.com      nameserver = ns1.google.com.
google.com      nameserver = ns2.google.com.
google.com      nameserver = ns4.google.com.
google.com      nameserver = ns3.google.com.
google.com
        origin = ns1.google.com
        mail addr = dns-admin.google.com
        serial = 513783434
        refresh = 900
        retry = 900
        expire = 1800
        minimum = 60
Name:   google.com
Address: 142.250.196.14
google.com      rdata_65 = 1 . alpn="h2,h3"
google.com      mail exchanger = 10 smtp.google.com.

Authoritative answers can be found from:
ns3.google.com  internet address = 216.239.36.10
ns3.google.com  has AAAA address 2001:4860:4802:36::a
ns1.google.com  internet address = 216.239.32.10
ns1.google.com  has AAAA address 2001:4860:4802:32::a
ns2.google.com  internet address = 216.239.34.10
ns2.google.com  has AAAA address 2001:4860:4802:34::a
ns4.google.com  internet address = 216.239.38.10
```

nslookup -type=txt google.com : Lookup for an txt record

TXT records are useful for multiple types of records like DKIM, SPF, etc. You can find all TXT records configured for any domain using the below command.

```
sana@sana-virtual-machine:~$ nslookup -type=txt google.com
;; Truncated, retrying in TCP mode.
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
google.com      text = "v=spf1 include:_spf.google.com ~all"
google.com      text = "onetrust-domain-verification=de01ed21f2fa4d8781cbc3ffb89cf4ef"
google.com      text = "atlassian-domain-verification=5YjTmWmjI92ewqkx2oXmBaD60Td9zWon9r6eakvHX6B77zzkFQto8PQ9QsKnbf4I"
google.com      text = "facebook-domain-verification=22rm551cu4k0ab0bxsw536tlds4h95"
google.com      text = "docusign=1b0a6754-49b1-4db5-8540-d2c12664b289"
google.com      text = "globalsign-smime-dv=CDYX+XFHUw2wml6/Gb8+59BsH31KzUr6c1l2BPvqKX8="
google.com      text = "google-site-verification=TV9-DBe4R80X4v0M4U_bd_J9cpOJM0nikft0jAgjmsQ"
google.com      text = "apple-domain-verification=30afIBcvSuDV2PLX"
google.com      text = "webexdomainverification.8YX6G=6e6922db-e3e6-4a36-904e-a805c28087fa"
google.com      text = "google-site-verification=wD8N7i1JTNTkezJ49swvWW48f8_9xveREV4oB-0Hf5o"
google.com      text = "MS=E4A68B9AB2BB9670BCE15412F62916164C0B20BB"
google.com      text = "docusign=05958488-4752-4ef2-95eb-aa7ba8a3bd0e"
```

nslookup -type=mx google.com : Lookup for an mx record

MX (Mail Exchange) record maps a domain name to a list of mail exchange servers for that domain. The MX record tells that all the mails sent to “google.com” should be routed to the Mail server in that domain.

```
sana@sana-virtual-machine:~$ nslookup -type=mx google.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
google.com      mail exchanger = 10 smtp.google.com.

Authoritative answers can be found from:
```

FTP

FTP (File Transfer Protocol) is a network protocol for transmitting files between computers over Transmission Control Protocol/Internet Protocol (TCP/IP) connections. Within the TCP/IP suite, FTP is considered an application layer protocol.

Linux ftp Command Syntax

The Linux ftp command uses the following basic syntax:

ftp [options] [IP]

The IP is the IP address of the system you are connecting to.

install ftp on ubuntu

```
sana@sana-virtual-machine:~$ sudo apt install vsftpd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  vsftpd
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 123 kB of archives.
After this operation, 326 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu kinetic/main amd64 vsftpd amd64 3.0.5-0ubuntu1 [123 kB]
Fetched 123 kB in 2s (65.0 kB/s)
Preconfiguring packages ...
Selecting previously unselected package vsftpd.
(Reading database ... 260731 files and directories currently installed.)
Preparing to unpack .../vsftpd_3.0.5-0ubuntu1_amd64.deb ...
Unpacking vsftpd (3.0.5-0ubuntu1) ...
Setting up vsftpd (3.0.5-0ubuntu1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/vsftpd.service → /lib/systemd/system/vsftpd.service.
Processing triggers for man-db (2.10.2-2) ...
```

```
sana@sana-virtual-machine:~$ ftp 192.168.0.112
Connected to 192.168.0.112.
220 (vsFTPd 3.0.5)
Name (192.168.0.112:sana): sana
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

```
sana@sana-virtual-machine:~$ sudo systemctl start vsftpd
sana@sana-virtual-machine:~$ sudo systemctl enable vsftpd
Synchronizing state of vsftpd.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable vsftpd
```

Log into the FTP Server

Once you initiate a connection to a remote system using the ftp command, the FTP interface requires you to enter a username and password to log in:

```
sana@sana-virtual-machine:~$ ftp 192.168.0.112
Connected to 192.168.0.112.
220 (vsFTPd 3.0.5)
Name (192.168.0.112:sana): sana
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

List Directories

The FTP interface allows you to list the contents of a directory on a remote system using the ls command:

ls

Using the command without any options displays the content of the remote system's current

working directory. In this example, that is the Home directory:

```
ftp> ls
229 Entering Extended Passive Mode (|||34974|)
150 Here comes the directory listing.
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Desktop
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Documents
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Downloads
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Music
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Pictures
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Public
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Templates
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Videos
-rw-rw-r--    1 1000     1000        58511 Feb 21 16:09 netstat.txt
drwxrwxr-x    6 1000     1000         4096 Feb 21 09:28 ns-allinone-3.33
-rw-rw-r--    1 1000     1000     28412785 Apr 27  2022 ns-allinone-3.33.tar.bz2
drwxrwxr-x    7 1000     1000         4096 Feb 21 10:06 ns-allinone-3.36.1
-rw-rw-r--    1 1000     1000     43615559 May 24  2022 ns-allinone-3.36.1.tar.b
z2
drwx------    4 1000     1000         4096 Feb 21 15:59 snap
-rw-r--r--    1 109      118       1253841 Feb 21 16:01 tcpdtcp
-rw-rw-r--    1 1000     1000        17433 Feb 21 16:02 tcpdtcp.txt
-rw-rw-r--    1 1000     1000            1 Feb 21 15:59 tcpdumptcp.txt
226 Directory send OK.
```

Transferring a file from a specific directory requires you to move into that directory:

```
ftp> cd Documents
250 Directory successfully changed.
ftp> get f.txt fs.txt
local: fs.txt remote: f.txt
229 Entering Extended Passive Mode (|||34253|)
150 Opening BINARY mode data connection for f.txt (3 bytes).
100% |***********************************************************************|     3       12.30 KiB/s    00:00 ETA
226 Transfer complete.
3 bytes received in 00:00 (0.06 KiB/s)
```

```
ftp> ls
229 Entering Extended Passive Mode (|||56078|)
150 Here comes the directory listing.
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Desktop
drwxr-xr-x    2 1000     1000         4096 Mar 04 21:54 Documents
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Downloads
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Music
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Pictures
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Public
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Templates
drwxr-xr-x    2 1000     1000         4096 Feb 21 00:53 Videos
-rw-rw-r--    1 1000     1000            3 Mar 05  2023 fs.txt
-rw-rw-r--    1 1000     1000        58511 Feb 21 16:09 netstat.txt
drwxrwxr-x    6 1000     1000         4096 Feb 21 09:28 ns-allinone-3.33
-rw-rw-r--    1 1000     1000     28412785 Apr 27  2022 ns-allinone-3.33.tar.bz2
drwxrwxr-x    7 1000     1000         4096 Feb 21 10:06 ns-allinone-3.36.1
-rw-rw-r--    1 1000     1000     43615559 May 24  2022 ns-allinone-3.36.1.tar.bz2
drwx------    4 1000     1000         4096 Feb 21 15:59 snap
-rw-r--r--    1 109      118       1253841 Feb 21 16:01 tcpdtcp
-rw-rw-r--    1 1000     1000        17433 Feb 21 16:02 tcpdtcp.txt
-rw-rw-r--    1 1000     1000            1 Feb 21 15:59 tcpdumptcp.txt
226 Directory send OK.
ftp>
```

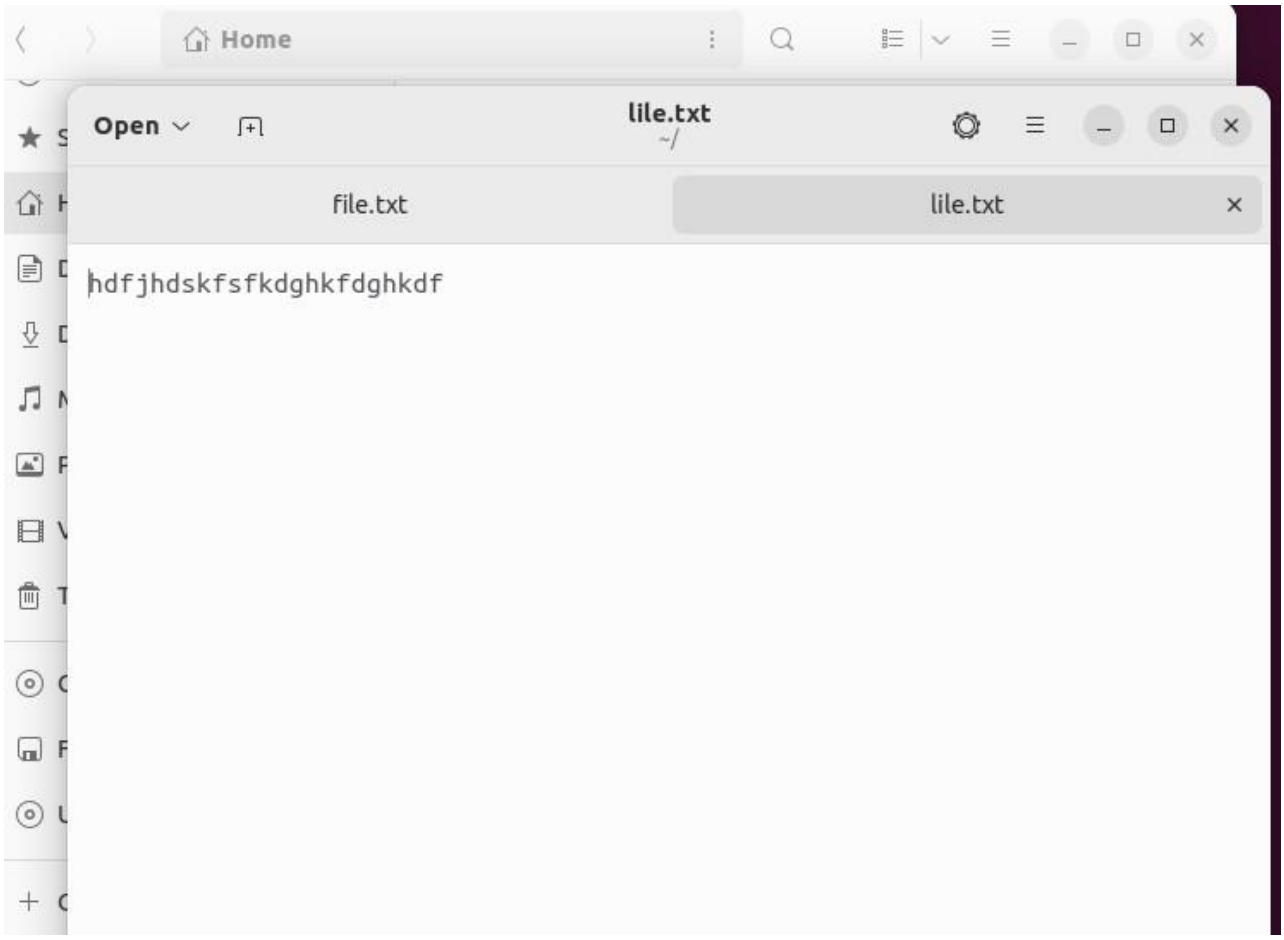
```
sana@sana-virtual-machine:~/Documents$ sudo adduser mateen
[sudo] password for sana:
Adding user `mateen' ...
Adding new group `mateen' (1001) ...
Adding new user `mateen' (1001) with group `mateen' ...
Creating home directory `/home/mateen' ...
Copying files from `/etc/skel' ...
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: password updated successfully
Changing the user information for mateen
Enter the new value, or press ENTER for the default
        Full Name []: mateen
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
```

```
sana@sana-virtual-machine:~/Documents$ sudo nano /etc/vsftpd.conf
sana@sana-virtual-machine:~/Documents$ sudo service vsftpd restart
```

```
sana@sana-virtual-machine: ~/Documents
  GNU nano 6.4                                    /etc/vsftpd.conf *
# This directive enables listening on IPv6 sockets. By default, listening
# on the IPv6 "any" address (::) will accept connections from both IPv6
# and IPv4 clients. It is not necessary to listen on *both* IPv4 and IPv6
# sockets. If you want that (perhaps because you want to listen on specific
# addresses) then you must run two copies of vsftpd with two configuration
# files.
listen_ipv6=YES
#
# Allow anonymous FTP? (Disabled by default).
anonymous_enable=NO
#
# Uncomment this to allow local users to log in.
local_enable=YES
#
# Uncomment this to enable any form of FTP write command.
write_enable=YES
#
# Default umask for local users is 077. You may wish to change this to 022,
# if your users expect that (022 is used by most other ftpd's)
#local_umask=022
#
# Uncomment this to allow the anonymous FTP user to upload files. This only
# has an effect if the above global write enable is activated. Also, you will
# obviously need to create a directory writable by the FTP user.
#anon_upload_enable=YES
#
# Uncomment this if you want the anonymous FTP user to be able to create
# new directories.
#anon_mkdir_write_enable=YES
```

```
mateen@sana-virtual-machine:~$ ftp 192.168.0.112
Connected to 192.168.0.112.
220 (vsFTPd 3.0.5)
Name (192.168.0.112:mateen): mateen
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put file.txt
local: file.txt remote: file.txt
229 Entering Extended Passive Mode (|||63791|)
150 Ok to send data.
  0% |                                     |     0        0.00 KiB/s    --:-- ETA
226 Transfer complete.
ftp>
```

```
mateen@sana-virtual-machine:~$ ls
file.txt  snap
mateen@sana-virtual-machine:~$ cat file.txt
mateen@sana-virtual-machine:~$ nano file.txt
mateen@sana-virtual-machine:~$ cat file.txt
hdfjhdskfsfkdghkfdghkdf
mateen@sana-virtual-machine:~$ ftp 192.168.0.112
Connected to 192.168.0.112.
220 (vsFTPd 3.0.5)
Name (192.168.0.112:mateen): sana
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put file.txt lile.txt
local: file.txt remote: lile.txt
229 Entering Extended Passive Mode (|||15710|)
150 Ok to send data.
100% |**********************************|    24      152.19 KiB/s    00:00 ETA
226 Transfer complete.
24 bytes sent in 00:00 (16.03 KiB/s)
```

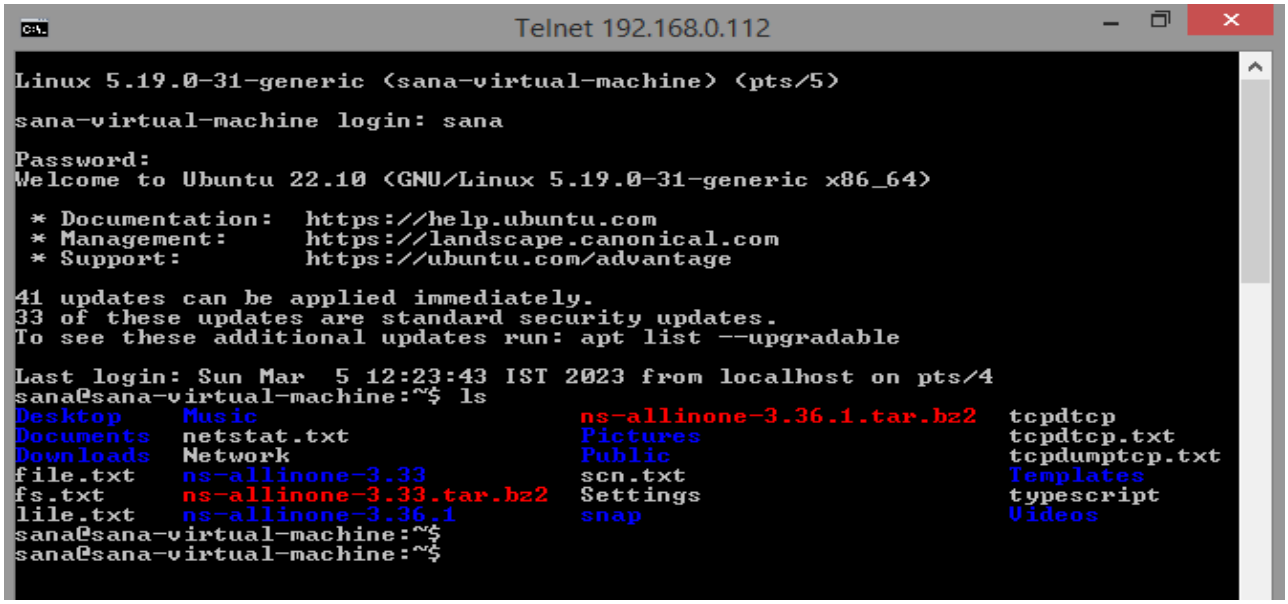


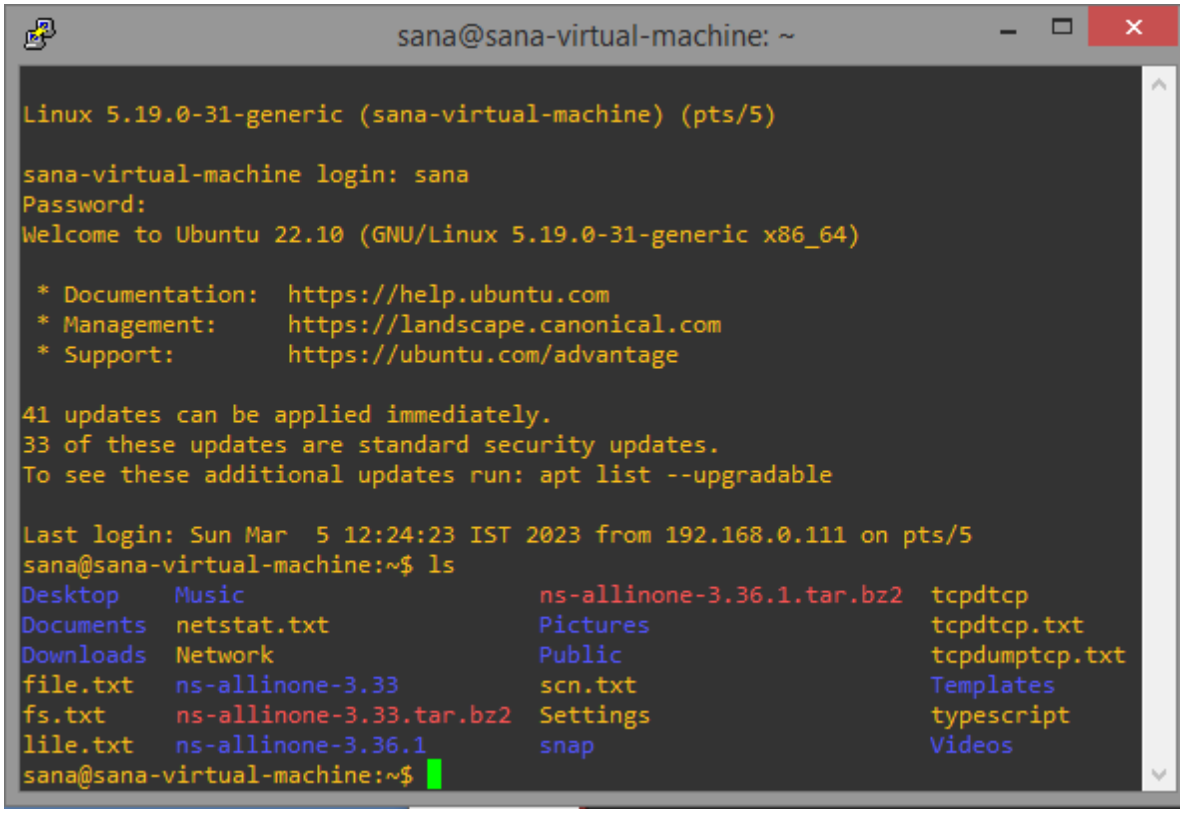
```
ftp> get lile.txt -
remote: lile.txt
229 Entering Extended Passive Mode (|||56845|)
150 Opening BINARY mode data connection for lile.txt (24 bytes).
hdfjhdskfsfkdghkfdghkdf
226 Transfer complete.
24 bytes received in 00:00 (23.15 KiB/s)
```

TELNET

Telnet is a terminal emulation program for TCP/IP networks that allows you to access another computer on the Internet or on a local network by logging on to the remote system. Telnet is a client-server protocol that connects to port 23 of the Transmission Control Protocol. You can also use Telnet to check open ports on a remote system.Telnet is an unencrypted and therefore insecure protocol.

```
sana@sana-virtual-machine:~$ sudo /etc/init.d/xinetd restart

[sudo] password for sana:
Restarting xinetd (via systemctl): xinetd.service.
sana@sana-virtual-machine:~$
```





traceroute

Traceroute is a widely used command-line utility available in almost all operating systems. It shows you the complete route to a destination address. It also shows the time is taken (or delays) between intermediate routers.

The syntax of the traceroute command is mentioned below:

traceroute [options] host_address [path_length]

```
sana@sana-virtual-machine:~$ sudo apt install traceroute
[sudo] password for sana:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  tcpd
Use 'sudo apt autoremove' to remove it.
The following NEW packages will be installed:
  traceroute
0 upgraded, 1 newly installed, 0 to remove and 37 not upgraded.
Need to get 50.2 kB of archives.
After this operation, 160 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu kinetic/universe amd64 traceroute amd64 1:2.1.0-3 [50.2 kB]
Fetched 50.2 kB in 1s (52.9 kB/s)
Selecting previously unselected package traceroute.
(Reading database ... 260870 files and directories currently installed.)
Preparing to unpack .../traceroute_1%3a2.1.0-3_amd64.deb ...
Unpacking traceroute (1:2.1.0-3) ...
Setting up traceroute (1:2.1.0-3) ...
update-alternatives: using /usr/bin/traceroute.db to provide /usr/bin/traceroute (traceroute) in auto mode
update-alternatives: using /usr/bin/traceroute6.db to provide /usr/bin/traceroute6 (traceroute6) in auto mode
update-alternatives: using /usr/bin/lft.db to provide /usr/bin/lft (lft) in auto mode
update-alternatives: using /usr/bin/traceproto.db to provide /usr/bin/traceproto (traceproto) in auto mode
update-alternatives: using /usr/sbin/tcptraceroute.db to provide /usr/sbin/tcptraceroute (tcptraceroute) in auto mode
Processing triggers for man-db (2.10.2-2) ...
sana@sana-virtual-machine:~$
```

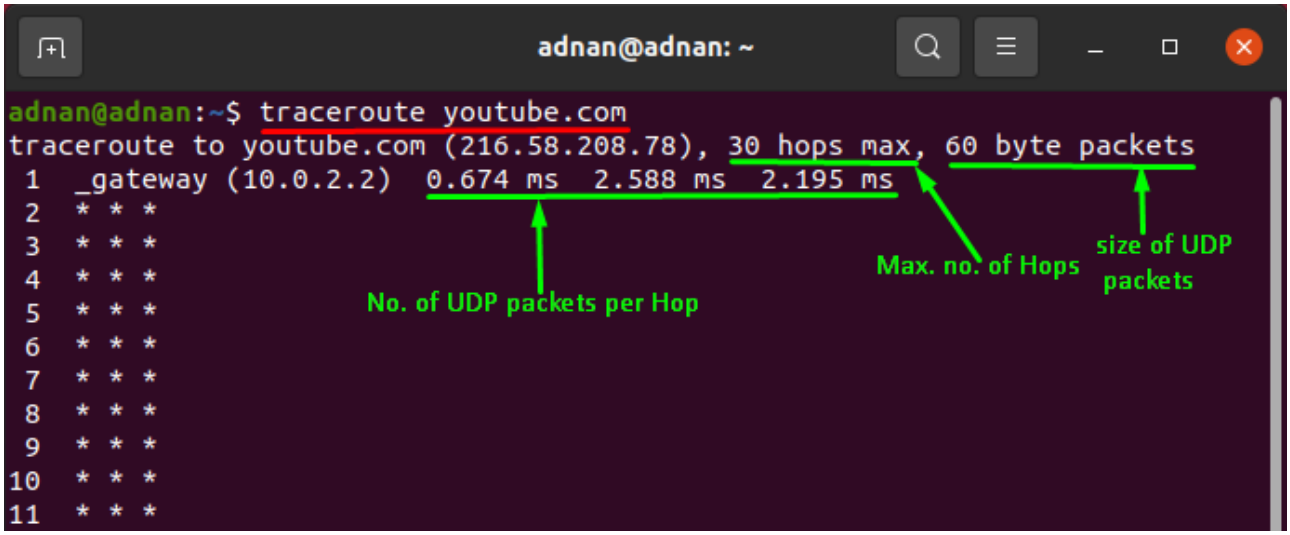
```
sana@sana-virtual-machine:~$ traceroute www.youtube.com
traceroute to www.youtube.com (142.250.182.110), 30 hops max, 60 byte packets
 1  _gateway (192.168.0.1)  1.686 ms  1.390 ms  1.172 ms
 2  * * *
 3  * * *
 4  broadband.actcorp.in (183.82.14.78)  16.252 ms  16.227 ms *
 5  72.14.243.242 (72.14.243.242)  16.496 ms broadband.actcorp.in (183.82.12.70)  2.699 ms  2.775 ms
 6  * * broadband.actcorp.in (183.82.14.78)  15.692 ms
 7  108.170.253.97 (108.170.253.97)  17.432 ms 72.14.243.242 (72.14.243.242)  16.458 ms  16.244 ms
 8  108.170.253.120 (108.170.253.120)  16.270 ms 142.251.55.241 (142.251.55.241)  15.508 ms 108.170.253.122 (108.170.253.122)  20.768 ms
 9  216.239.43.234 (216.239.43.234)  16.178 ms 142.251.55.246 (142.251.55.246)  17.753 ms 74.125.242.129 (74.125.242.129)  16.497 ms
10  maa05s21-in-f14.1e100.net (142.250.182.110)  16.285 ms 142.251.55.241 (142.251.55.241)  16.029 ms 108.170.253.120 (108.170.253.120)  16.22
4 ms
sana@sana-virtual-machine:~$
```

The IP address of the destination

Number of Hops: it is a numeric value and shows how much time the traceroute will try to reach the destination (the default value is 30)

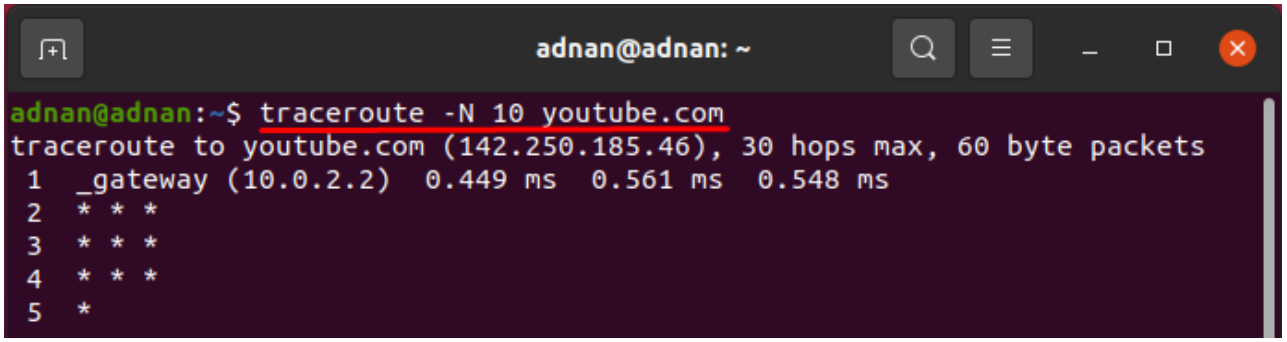
Number of Probes you are sending per Hop or number of packets per Hop (default value is 3)

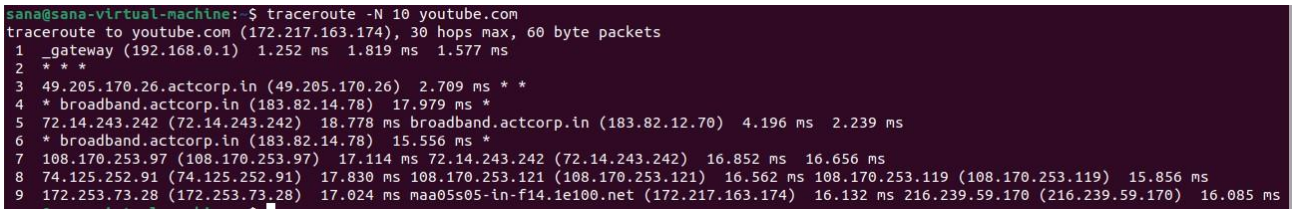
The last major point is about the size of packets you are sending. (its default value is 60bytes)



You can set the number of Probes that are being passed: by default, 16 probes are passed simultaneously; you can adjust them using the "-N" option: the command given below will set the number to 10:

$ traceroute -N 10 youtube.com





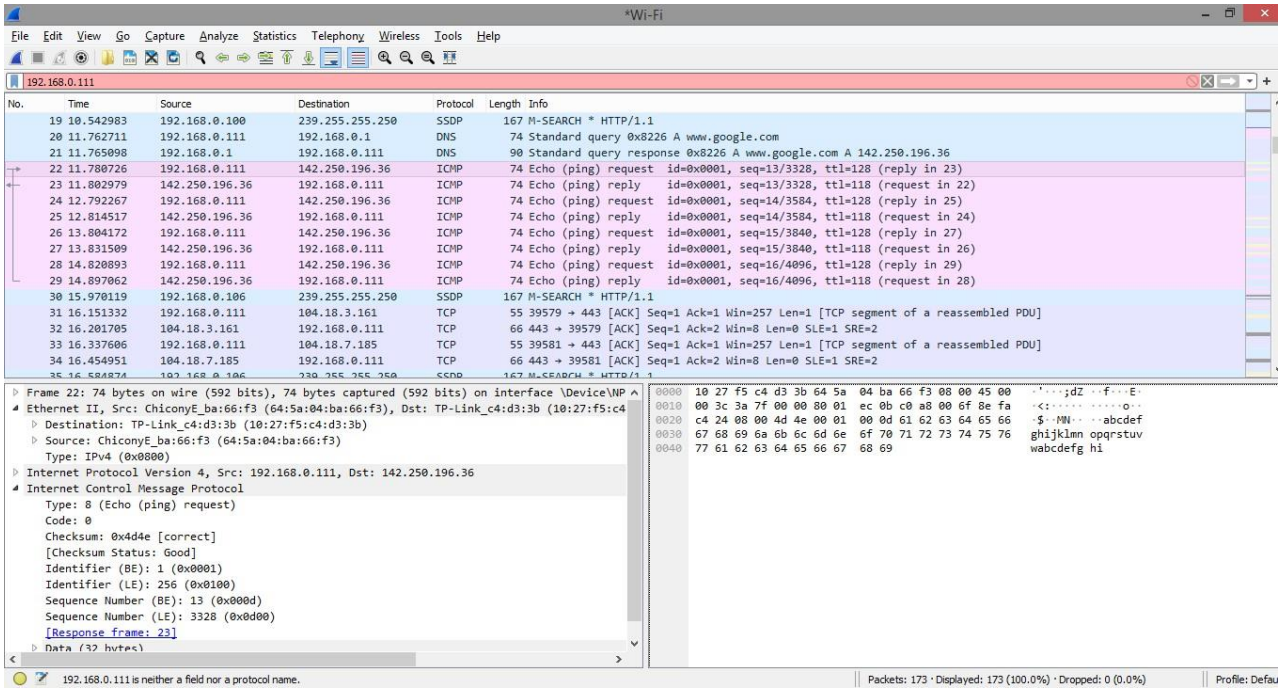
With the help of the "-w" option, you can increase the number of seconds that each Hop must wait to show the result. It gets only float values, so you have to pass a floating number (6.0, 6.5); and it is foreseen that increasing the number of Hop times will show a better response: the

command given below will set the wait time of 5.5ms for each Hop:

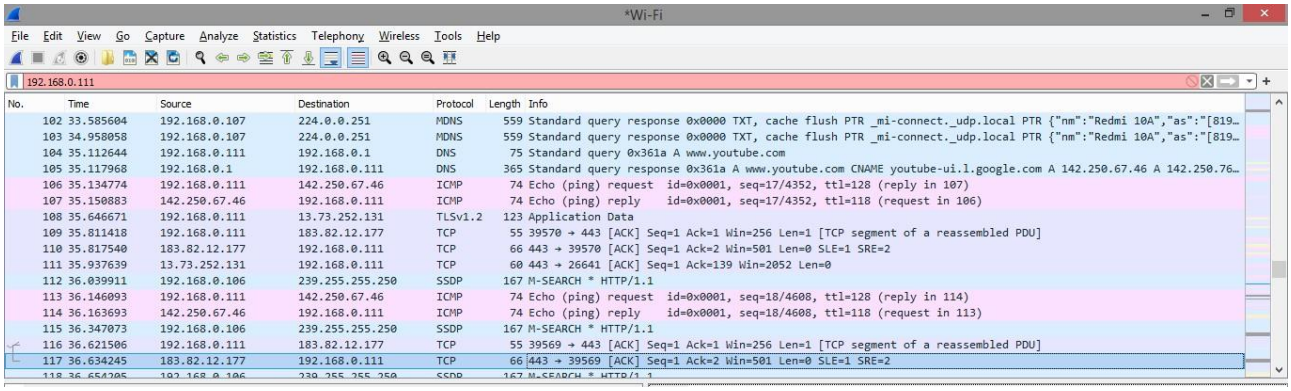
```
sana@sana-virtual-machine:~$ traceroute -w 5.5 youtube.com
traceroute to youtube.com (172.217.163.206), 30 hops max, 60 byte packets
 1  _gateway (192.168.0.1)  7.739 ms  11.502 ms  10.888 ms
 2  * * *
 3  * 49.205.170.26.actcorp.in (49.205.170.26)  9.783 ms *
 4  broadband.actcorp.in (183.82.14.78)  20.850 ms * *
 5  72.14.243.242 (72.14.243.242)  19.904 ms  19.965 ms  19.730 ms
 6  * * *
 7  72.14.243.242 (72.14.243.242)  16.407 ms  16.322 ms 108.170.253.97 (108.170.253.97)  22.056 ms
 8  108.170.253.120 (108.170.253.120)  21.513 ms 108.170.253.104 (108.170.253.104)  21.100 ms 108.170.253.122 (108.170.253.122)  20.483 ms
 9  maa05s06-in-f14.1e100.net (172.217.163.206)  21.338 ms 74.125.242.129 (74.125.242.129)  21.063 ms 142.251.55.230 (142.251.55.230)  18.190
ms
sana@sana-virtual-machine:~$
```

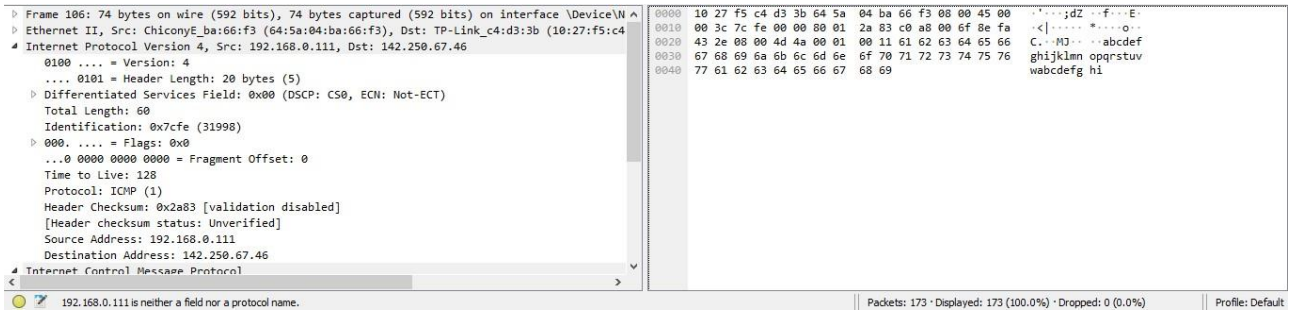
A network protocol analyzer such as Wireshark can be used to capture and examine Ping and Traceroute PDUs. The analyzer can display the PDU fields and decode the data in the payloads, allowing the user to see the details of the communication between the sender and target hosts.

```
C:\Users\Sana>ping www.google.com

Pinging www.google.com [142.250.196.36] with 32 bytes of data:
Reply from 142.250.196.36: bytes=32 time=22ms TTL=118
Reply from 142.250.196.36: bytes=32 time=22ms TTL=118
Reply from 142.250.196.36: bytes=32 time=27ms TTL=118
Reply from 142.250.196.36: bytes=32 time=76ms TTL=118

Ping statistics for 142.250.196.36:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 22ms, Maximum = 76ms, Average = 36ms
```

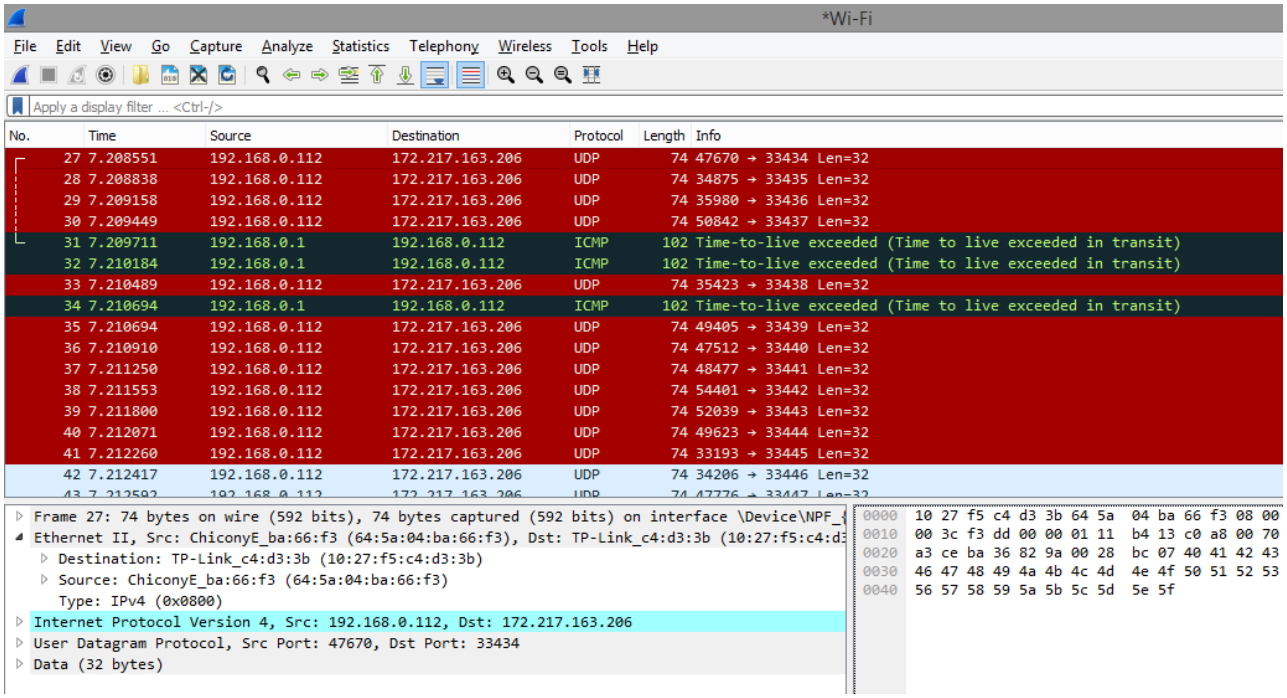



```
C:\Users\Sana>ping www.youtube.com

Pinging youtube-ui.l.google.com [142.250.67.46] with 32 bytes of data:
Reply from 142.250.67.46: bytes=32 time=16ms TTL=118
Reply from 142.250.67.46: bytes=32 time=17ms TTL=118
Reply from 142.250.67.46: bytes=32 time=16ms TTL=118
Reply from 142.250.67.46: bytes=32 time=17ms TTL=118

Ping statistics for 142.250.67.46:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 16ms, Maximum = 17ms, Average = 16ms
```

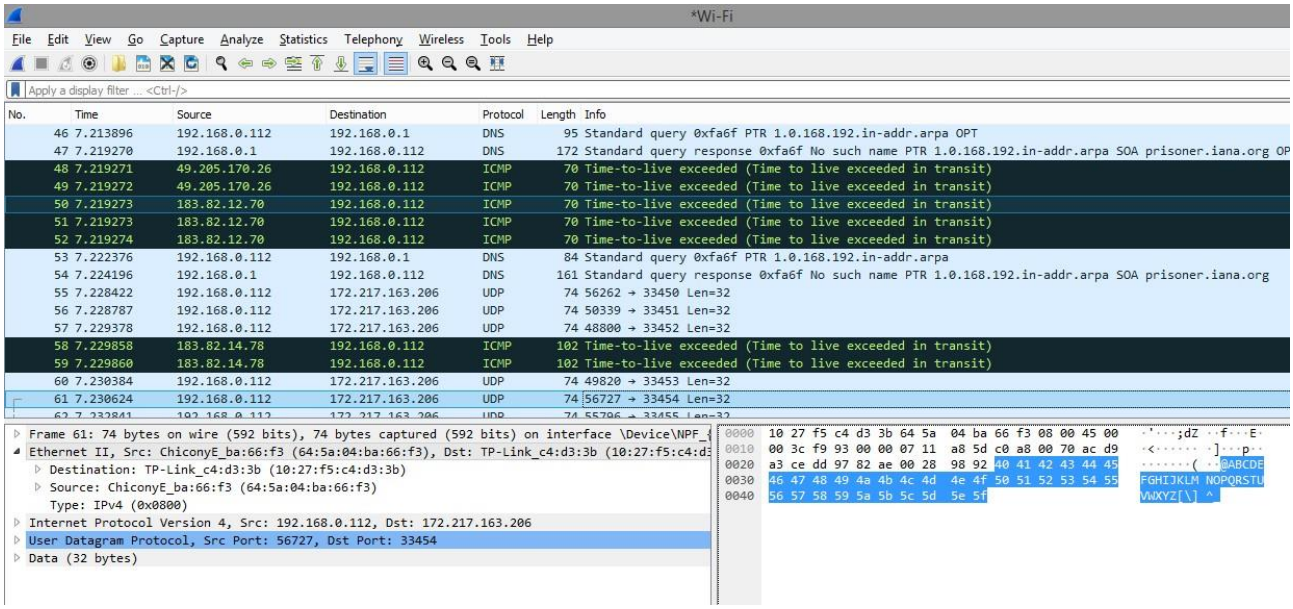





```
sana@sana-virtual-machine: $ traceroute -w 5.5 youtube.com
traceroute to youtube.com (172.217.163.206), 30 hops max, 60 byte packets
 1  _gateway (192.168.0.1)  1.709 ms  1.569 ms  1.772 ms
 2  * * *
 3  * 49.205.170.26.actcorp.in (49.205.170.26)  8.394 ms  8.052 ms
 4  broadband.actcorp.in (183.82.14.78)  18.490 ms  18.244 ms *
 5  broadband.actcorp.in (183.82.12.70)  7.153 ms  6.979 ms  6.822 ms
 6  * * *
 7  142.251.55.234 (142.251.55.234)  20.129 ms 209.85.248.180 (209.85.248.180)  19.572 ms 72.14.243.242 (72.14.243.242)  18.762 ms
 8  209.85.248.219 (209.85.248.219)  17.410 ms * 108.170.253.103 (108.170.253.103)  16.387 ms
 9  74.125.242.129 (74.125.242.129)  16.421 ms 142.251.55.228 (142.251.55.228)  15.384 ms 74.125.253.12 (74.125.253.12)  16.327 ms
10  maa05s06-in-f14.1e100.net (172.217.163.206)  15.900 ms  15.634 ms 108.170.253.105 (108.170.253.105)  18.290 ms
```



```
sana@sana-virtual-machine: $ traceroute -N 10 youtube.com
traceroute to youtube.com (172.217.163.206), 30 hops max, 60 byte packets
 1  _gateway (192.168.0.1)  15.059 ms  15.063 ms  15.458 ms
 2  * * *
 3  * * *
 4  * broadband.actcorp.in (183.82.14.78)  301.246 ms  300.387 ms
 5  broadband.actcorp.in (183.82.12.70)  299.985 ms 72.14.243.242 (72.14.243.242)  188.103 ms broadband.actcorp.in (183.82.12.70)  187.343 ms
 6  * * broadband.actcorp.in (183.82.14.78)  262.739 ms
 7  108.170.253.97 (108.170.253.97)  257.001 ms 72.14.243.242 (72.14.243.242)  328.940 ms  189.098 ms
 8  * 108.170.253.105 (108.170.253.105)  38.970 ms 209.85.248.219 (209.85.248.219)  38.704 ms
 9  74.125.242.129 (74.125.242.129)  38.454 ms 74.125.242.145 (74.125.242.145)  38.244 ms  38.044 ms
10  209.85.248.219 (209.85.248.219)  37.832 ms 209.85.248.211 (209.85.248.211)  326.995 ms maa05s06-in-f14.1e100.net (172.217.163.206)  326.79
1 ms
```



2. Implement the datalink layer framing methods such as byte-stuffing and bit stuffing

Security and Error detection are the most prominent features that are to be provided by any application which transfers data from one end to the other end. One of such a mechanism in tracking errors which may add up to the original data during transfer is known as Stuffing. It is of two types namely Bit Stuffing and the other Character Stuffing. Coming to the Bit Stuffing, 01111110 is appended within the original data while transfer of it. The following program describes how it is stuffed at the sender end and de-stuffed at the reciever end.

program

```
#include<stdio.h>
#include<string.h>
void main()
{
        int a[20],b[30],i,j,k,count,n;
```

```
printf("Enter frame length:");
scanf("%d",&n);
printf("Enter input frame (0's & 1's only):");
for(i=0;i<n;i++)
  scanf("%d",&a[i]);
i=0; count=1; j=0;
while(i<n)
{
  if(a[i]==1)
  {
          b[j]=a[i];
          for(k=i+1;a[k]==1 && k<n && count<5;k++)
          {
                  j++;
                  b[j]=a[k];
                  count++;
                  if(count==5)
                  {
                          j++;
                          b[j]=0;
                  }
                  i=k;
          }
  }
  else
  {
          b[j]=a[i];
  }
  i++;
  j++;
```

```
        }
        printf("After stuffing the frame is:");
        for(i=0;i<j;i++)
          printf("%d",b[i]);
}
```

Output

```
sana@sana-virtual-machine:~/cnlab$ gcc bitstuff.c
sana@sana-virtual-machine:~/cnlab$ ls
a.out  bitstuff.c
sana@sana-virtual-machine:~/cnlab$ ./a.out
Enter frame length:5
Enter input frame (0's & 1's only):1 1 1 1 1
After stuffing the frame is:111110sana@sana-virtual-machine:~/cnlab$
```

Byte stuffing (or character stuffing) is a method for converting a message formed of a sequence of bytes that may contain reserved values such as frame delimiters into another byte sequence that does not contain the reserved values.

When there is a character with the same pattern as the flag, a byte usually called the escape character [ESC], with a preset bit pattern is appended to the data part of the frame. When the ESC character is encountered, the receiver removes it from the data section and interprets the following character as data rather than a flag.

The issue arises when the text contains one or more escape characters followed by a flag. To fix this problem, the escape characters that are part of the text are indicated by another escape character, i.e., if the escape character is part of the text, an extra one is added to signify that the second one is part of the text.

How does the Byte Stuffing Program in C works?

The physical layer's stream of bits is split into data frames at the Data Link layer. The data frames can be of constant or variable length. In variable-length framing, the size of each frame to be sent can vary.

In order to distinguish between one frame and the next, a pattern of bits is utilized as a

delimiter. However, if the pattern appears in the message, then mechanisms must be included to ensure that this circumstance is prevented.

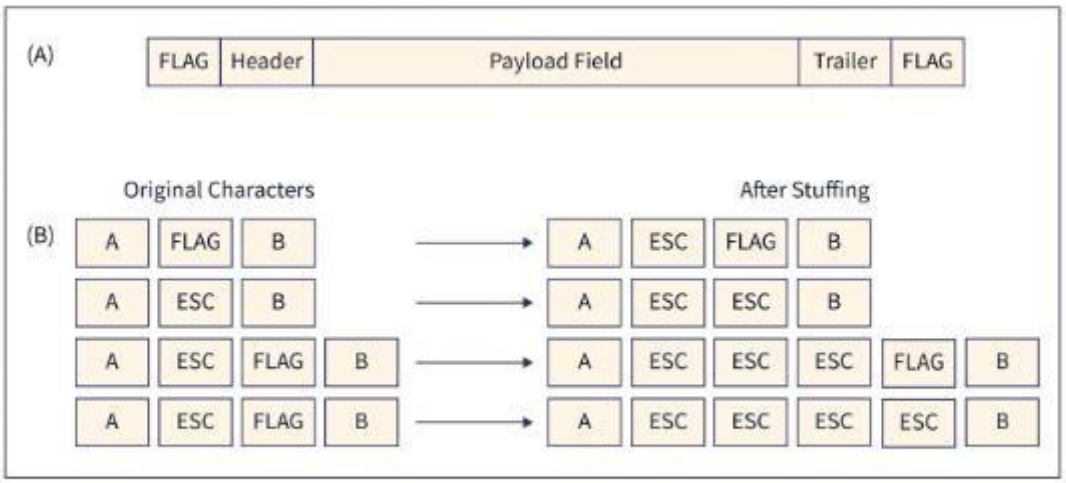
Every byte in the message is prefixed with an escape character [ESC], which follows the same pattern as the flag byte. A second ESC byte is inserted before the message byte if the ESC sequence is present there. When a receiver encounters the ESC, it removes it from the data section and considers the next character to be data rather than a delimiting flag.

What happens if there are one or more escape characters followed by a flag in the data?

The receiver discards the escape character but retains the flag, which is misinterpreted as the end of the frame. To resolve this issue, the escape characters that are a part of the data must also be identified by another escape character.

A data link frame has the following parts −

1. Frame Header − It includes the source and the destination addresses of the frame.
2. Payload field − It includes the intended message. In Byte stuffing, it is a variable sequence of data bytes.
3. Trailer − It contains the error detection and error correction bits.
4. Flags − Flags are frame delimiters that indicate the beginning and end of a frame. In Byte stuffing, a flag is 1- byte denoting a protocol-dependent unique character.



A Character Stuffing
(A) A Character delimited by flag bytes
(B) Four examples of byte sequences before and after byte stuffing

BYTE STUFFING MECHANISM

Four examples of byte sequences before and after stuffing:

PAYLOAD FIELD

Example of Byte Stuffing Program in C

```
#include<stdio.h>
#include<string.h>
int main()
{
    char a[30], fs[50] = " ", t[3], sd, ed, x[3], s[3], d[3], y[3];
    int i, j, p = 0, q = 0;
    printf("Enter characters to be stuffed:");
    scanf("%s", a);
    printf("\nEnter a character that represents starting delimiter:");
    scanf(" %c", &sd);
    printf("\nEnter a character that represents ending delimiter:");
    scanf(" %c", &ed);
    x[0] = s[0] = s[1] = sd;
```

```
    x[1] = s[2] = '\0';
    y[0] = d[0] = d[1] = ed;
    d[2] = y[1] = '\0';
    strcat(fs, x);
    for(i = 0; i < strlen(a); i++)
    {
       t[0] = a[i];
       t[1] = '\0';
       if(t[0] == sd)
          strcat(fs, s);
       else if(t[0] == ed)
          strcat(fs, d);
       else
          strcat(fs, t);
    }
    strcat(fs, y);
    printf("\n After stuffing:%s", fs);
    return 0;
  }
```

Output:

```
sana@sana-virtual-machine:~/cnlab$ gedit charstuff.c
sana@sana-virtual-machine:~/cnlab$ gcc  charstuff.c
sana@sana-virtual-machine:~/cnlab$ ./a.out
Enter characters to be stuffed:goodday

Enter a character that represents starting delimiter:d

Enter a character that represents ending delimiter:g

 After stuffing: dggooddddaygsana@sana-virtual-machine:~/cnlab$
```

3. Implement on a dataset of characters using CRC polynomials CRC12 and CRC16

Cyclic Redundancy Check is an error detection algorithm used to check the validity of the data sent by the sender. In CRC, in addition to the data to be transmitted, the algorithm requires a **generator polynomial** that is used to compute the **check value** using binary division. The check value or the CRC is sent along with the data to the receiver to check the validity of the data.

The data that is to be sent to the receiver can be represented in the polynomial form with the degree of the polynomial as the bit positions. For example, the binary data 1010101 of length 7 can be represented as,

$$x^7 + x^5 + x^3 + 1$$

The bit value of 0 is not represented as the value of that representation is also 0.

The generator polynomial can also be represented in binary. The degree of the generator polynomial must be greater than 0 and lesser than the degree of the data. The CRC can be classified into different standards based on the degree of the generator polynomial. The CRC-8 standard use a generator polynomial of degree 8 and CRC-16 uses a generator polynomial of degree 16. A simple representation of the generator polynomial is given as follows.

$$x^5 + x^4 + x^2$$

The above generator polynomial is represented in binary data as 00011010.

Note : Cyclic Redundancy Check can also be used as a hashing function and in such cases, the CRC-8 standard is not used as it can produce only 256(282_8) values.

The **steps involved** in CRC are as follows, In the **sender side**,

- The data of length, n, and the generator polynomial of length, l is prepared.
- The data to be sent is appended with (l-1) number of zeros.
- Binary division is performed with the data as dividend and the binary equivalent of the generator polynomial as the divisor. The remainder of the binary division is the check value.
- The signal is sent with the check value appended to the end of the data.

The **mathematical representation** of check value or CRC is,

$$CRC = remainder\ of\left[data * \frac{x^n}{generatorpolynomial}\right]$$

Here, the n represents the number of bits in the generator polynomial On the receiver side,

- The data received again proceeds with binary division with the data as dividend and the binary equivalent of the generator polynomial as the divisor.
- If the remainder of the binary division is zero then the data transmitted from the sender has no error. If the remainder is not zero, then the signal has been corrupted with an error.
- The process of binary division follows the following steps and is similar to normal polynomial division,

- Each step of the division involves XOR of divisor and dividend. The first n bits of the divisor are only used for this operation where n is the number of bits in the dividend.
- The quotient will be 1 or 0 based on the n-bit data. If the last bit of the data is 1, then the quotient is 1 and if the last bit of the data is 0, then the quotient is 0.
- Then the bit at the position n+1 is taken from the data and appended with the remainder of the above operation. This remainder becomes the divisor for the next operation.
- The operation is repeated until all the bits in the data are used in the calculation.

The above procedure can be one of the interesting properties followed by the Cyclic Redundancy Check is, CRC(x^y^z) == crc(x) ^ crc(y) ^ crc(z).

The ^ symbol represents the XOR function. If we perform an XOR on 3 data and perform a CRC on the result, then the result of the CRC will be equivalent to CRC on the individual data and XOR with the results. CRC can be used as

Note : The XOR operator returns 0 when both the inputs are the same and returns 1 in other cases.

program for crc 12

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

int copy();
int check();

int i, j, k, t, count = 0, num = 0;
int gen[10], frame[30], rem[30], temp[30];

int main() {
   char c, plym[50];
   char ch[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};

   printf("\t\t\t***CYCLIC REDUNDANCY CHECK-12***\n\n\n");

   for (i = 0; i < 50; i++)
      plym[i] = '\0';

   for (i = 0; i < 30; i++)
      temp[i] = rem[i] = frame[i] = '\0';

   for (i = 0; i < 10; i++)
      gen[i] = '\0';

   printf("Enter the polynomial: ");
```

```
fgets(plym, 50, stdin);
plym[strcspn(plym, "\n")] = '\0';
strcat(plym, " ");

for (i = 0; i < strlen(plym); i++) {
   if (plym[i] == 'x') {
      i++;
      for (; plym[i] != ' '; count++, i++) {
      }
      if (count == 3) {
         for (i = i - 1, j = 3; j <= 9; j++)
            if (plym[i] == ch[j]) {
               printf("Enter the polynomial's degree is high");
               exit(0);
            }
         for (j = 0, num = 10; j <= 2; j++)
            if (plym[i] == ch[j]) {
               num = num + j;
               frame[num] = 1;
            }
      } else if (count == 2) {
         for (i = i - 1, j = 1, num = 0; j <= 9; j++)
            if (plym[i] == ch[j])
               num = j;
         frame[num] = 1;
      } else if (count == 0) {
         frame[1] = 1;
      }
      count = 0;
   } else if (plym[i] == '1') {
```

```
        frame[0] = 1;
    }
}

printf("FRAME is: ");
for (i = 12, j = 0; i >= 0; i--, j++) {
    temp[j] = frame[i];
    printf("%d", frame[i]);
}

printf("\n\n\n>>>>both high & low orders bits of GENERATOR must be 1<<<<");
printf("\nEnter the generator: ");

for (num = i = 0; (c = getchar()) != '\n'; i++, num++) {
    if (c == '1') {
        gen[i] = 1;
    } else if (c == '0') {
        gen[i] = 0;
    } else {
        printf("\nEnter the GENERATOR is other than 0 or 1");
        exit(0);
    }
}

for (j = 13, i = i - 1; i > 0; i--, j++) {
    temp[j] = 0;
}

printf("\n\n FRAME after appending 0's: ");
copy();
```

```
    check();

  printf("\n The REMAINDER is: ");

    for(i = 13; i < j; i++)
    {
       temp[i] = rem[i];
       printf("%d", rem[i]);
    }
    printf("\n\n\n Transmitting FRAME ..... ");
    //delay(10000);
    printf("\n\n\n Transmitted FRAME is: ");
    copy();
    check();
    printf("\n frame recieved");
    printf("\n\n\n checking for errors ..... ");
    //delay(10000);
    printf("\n\n\n recieved frame is: ");
    copy();
    check();
    printf("\n the remainder is: ");
    for(i = 13; i < j; i++)
    printf("%d", rem[i]);
    printf("\n DATA SENT SUCCESSFULLY");
    return 0;
}
int check()
{
    for(i = 0; i <= 12; i++)
    {
```

```
        if(rem[i] == 0)
        continue;
        else
        {
           for(k = 0, t = i; k < num; k++, t++)
           {
              if(rem[t] == 1 && gen[k] == 1)
              rem[t] = 0;
              else if(rem[t] == 0 && gen[k] == 0)
              rem[t] = 0;
              else if(rem[t] == 1 && gen[k] == 0)
              rem[t] = 1;
              else if(rem[t] == 0 && gen[k] == 1)
              rem[t] = 1;
           }
        }
    }
    return 0;
}
int copy()
{
    for(i = 0; i < j; i++)
    {
       printf("%d", temp[i]);
       rem[i] = temp[i];
    }
    return 0;
    }
```

```
sana@sana-virtual-machine:~/cnlab$ gcc crc12.c
sana@sana-virtual-machine:~/cnlab$ ./a.out
```

```
                        ***CYCLIC REDUNDANCY CHECK-12***


Enter the polynomial: x^8 x^7 x^3 x^2 1
FRAME is: 0000110001101


>>>>both high & low orders bits of GENERATOR must be 1<<<<
Enter the generator: 10011


 FRAME after appending 0's: 00001100011010000
 The REMAINDER is: 0101


 Transmitting FRAME......


 Transmitted FRAME is: 00001100011010101
 frame recieved


 checking for errors......
```

```
recieved frame is: 00001100011010101
the remainder is: 0000
```

program for crc 16

```
#include<conio.h>
#include<string.h>
#include<dos.h>
#include<stdlib.h>
int copy();
int check();
int i, j, k, t, count = 0, num = 0;
int gen[10], frame[30], rem[30], temp[30];
void main()
{
   char c, plym[50];
   char ch[]={'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
   clrscr();
   printf("\t\t\t***CYCLIC REDUNDANCY CHECK-12***\n\n\n");
   for(i = 0; i < 50; i++)
```

```
plym[i] = '\0';
for(i = 0; i < 30; i++)
temp[i] = rem[i] = frame[i] = '\0';
for(i = 0; i < 10; i++)
gen[i] = '\0';
printf("enter the polynomial: ");
gets(plym);
plym[strlen(plym)] = ' ';
for(i = 0; i < strlen(plym); i++)
{
   if(plym[i] == 'x')
   {
      i++;
      for(;plym[i] != '+'; count, i++)
      {
      }
      if(count == 3)
      {
         for(i = i - 1, j = 3; j <= 9; j++)
         if(plym[i] == ch[j])
         {
            printf("\Enter the polynomial's");
            printf("degree is high");
            getch(); exit(0);
         }
         for(j = 0, num = 10; j <= 2; j++)
      if(plym[i] == ch[j])
      {
         num=num + j;
         frame[num] = 1;
      }
   }
   if(count == 2)
   {
      for(i = i - 1, j = 1, num = 0; j <= 9; j++)
      if(plym[i] == ch[j])
      num = j;
      frame[num] = 1;
   }
   if(count == 0)
   frame[1] = 1;
   count = 0;
}
else if(plym[i] == '1')
frame[0] = 1;
```

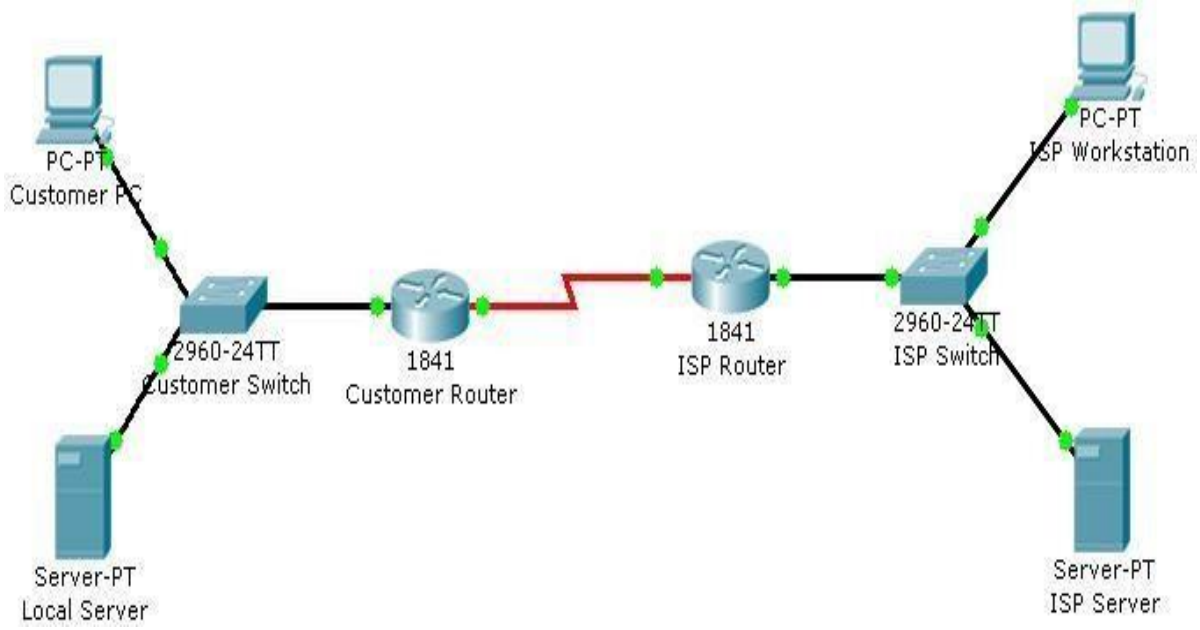
```
}
printf("FRAME is: ");
for(i = 12, j = 0; i >= 0; i--, j++)
{
   temp[j] = frame[i];
   printf("%d", frame[i]);
}
printf("\n\n\n>>>>both high & low orders");
printf("bits of GENERATOR must be 1<<<<");
printf("\n enter the generator: ");
for(num = i = 0; (c = getchar()) != '\n'; i++, num++)
{
   if(c == '1')
   gen[i] =  1;
   else if(c == '0')
   gen[i] =  0;
   else
   {
      printf("\n Enter the GENERATOR");
      printf("is other then 0 or 1");
      getch(); exit(0);
   }
}
for(j = 13,i = i - 1; i > 0; i--, j++)
temp[j] = 0;
printf("\n\n FRAME after appending 0's: ");
copy(); check();
printf("\n The REMAINDER is: ");
for(i = 13; i < j; i++)
{
   temp[i] = rem[i];
   printf("%d", rem[i]);
}
printf("\n\n\n Transmitting FRAME..... ");
delay(10000);
printf("\n\n\n Transmitted FRAME is: ");
copy(); check();
printf("\n frame recieved");
printf("\n\n\n checking for errors..... ");
delay(10000);
printf("\n\n\n recieved frame is: ");
copy(); check();
printf("\n the remainder is: ");
for(i = 13; i < j; i++)
printf("%d", rem[i]);
```

```
    printf("\n DATA SENT SUCCESSFULLY");
    getch();
}
check()
{
    for(i = 0; i <= 12; i++)
    {
        if(rem[i] == 0)
        continue;
        else
        {
            for(k = 0, t = i; k < num; k++, t++)
            {
                if(rem[t] == 1 && gen[k] == 1)
                rem[t] = 0;
                else if(rem[t] == 0 && gen[k] == 0)
                rem[t] = 0;
                else if(rem[t] == 1 && gen[k] == 0)
                rem[t] = 1;
                else if(rem[t] == 0 && gen[k] == 1)
                rem[t] = 1;
            }
        }
    }
    return 0;
}
copy()
{
    for(i = 0; i < j; i++)
    {
        printf("%d", temp[i]);
        rem[i] = temp[i];
    }
    return 0;
}
```

4. Initial Configuration of router and switch(using real devices or simulators)

(a) Performing an Initial Switch Configuration

Topology Diagram



Objectives

- Perform an initial configuration of a Cisco Catalyst 2960 switch.

Background / Preparation

In this activity, you will configure these settings on the customer Cisco Catalyst 2960 switch:

- Host name
- Console password
- vty password
- Privileged EXEC mode password
- Privileged EXEC mode secret
- IP address on VLAN1 interface
- Default gateway

Note: Not all commands are graded by Packet Tracer.

Step 1: Configure the switch host name.

a. From the Customer PC, use a console cable and terminal emulation software to connect to the consoleof the customer Cisco Catalyst 2960 switch.

b. Set the host name on the switch to **CustomerSwitch** using these commands.

Switch>**enable**
Switch#**configu**
re terminal
Switch(config)#**hostname CustomerSwitch**

Step 2: Configure the privileged mode password and secret.

a. From global configuration mode, configure the password as **cisco**.

CustomerSwitch(config)#**enable password cisco**

b. From global configuration mode, configure the secret as **cisco123**.

CustomerSwitch(config)#**enable secret cisco123**

Step 3: Configure the console password.

a. From global configuration mode, switch to configuration mode to configure the console line.

CustomerSwitch(config)#**line console 0**

b. From line configuration mode, set the password to **cisco** and require the password to be entered atlogin.

CustomerSwitch(config-line)#**password cisco**
CustomerSwitch(config-line)#**login**
CustomerSwitch(config-line)#**exit**

Step 4: Configure the vty password.

a. From global configuration mode, switch to the configuration mode for the vty lines 0 through 15.

CustomerSwitch(config)#**line vty 0 15**

b. From line configuration mode, set the password to **cisco** and require the password to be entered atlogin.

```
CustomerSwitch(config-
line)#password cisco
CustomerSwitch(config-
line)#login
CustomerSwitch(config-
line)#exit
```

Step 5: Configure an IP address on interface VLAN1.

From global configuration mode, switch to interface configuration mode for VLAN1, and assign the IP address192.168.1.5 with the subnet mask of 255.255.255.0.

```
CustomerSwitch(config)#interface vlan 1
CustomerSwitch(config-if)#ip address 192.168.1.5 255.255.255.0
CustomerSwitch(config-if)#no shutdown
CustomerSwitch(config-if)#exit
```

Step 6: Configure the default gateway.

a. From global configuration mode, assign the default gateway to 192.168.1.1.

```
CustomerSwitch(config)#ip default-gateway 192.168.1.1
```

b. Click the **Check Results** button at the bottom of this instruction window to check your work.

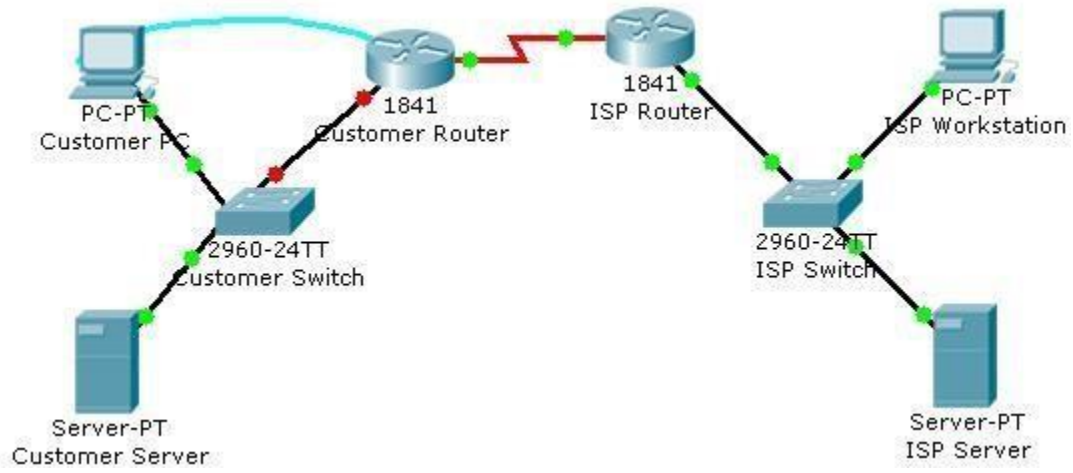
Step 7: Verify the configuration.

The Customer Switch should now be able to ping the ISP Server at 209.165.201.10. The first one or two pingsmay fail while ARP converges.

```
CustomerSwitch(config)
#end
CustomerSwitch#ping
209.165.201.10
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 209.165.201.10, timeout is 2 seconds:
..!!!
Success rate is 60 percent (3/5), round-trip min/avg/max = 181/189/197 ms
CustomerSwitch#
```

(b) Performing an Initial Router Configuration

Topology Diagram



Objectives

- Configure the router host name.
- Configure passwords.
- Configure banner messages.
- Verify the router configuration.

Background / Preparation

In this activity, you will use the Cisco IOS CLI to apply an initial configuration to a router, including hostname, passwords, a message-of-the-day (MOTD) banner, and other basic settings.

Note: Some of the steps are not graded by Packet Tracer.

Step 1: Configure the router host name.

a. On Customer PC, use the terminal emulation software to connect to the console of the customer Cisco 1841 ISR.

Set the host name on the router to **CustomerRouter** by using these commands.

```
Router>enable
Router#configu
re terminal
Router(config)#hostname CustomerRouter
```

Step 2: Configure the privileged mode and secret passwords.

a. In global configuration mode, set the password to **cisco**.

CustomerRouter(config)#**enable password cisco**

Set an encrypted privileged password to **cisco123** using the **secret** command.

CustomerRouter(config)#**enable secret cisco123**

Step 3: Configure the console password.

a. In global configuration mode, switch to line configuration mode to specify the console line.

CustomerRouter(config)#**line console 0**

Set the password to **cisco123**, require that the password be entered at login, and then exit line configuration mode.

```
CustomerRouter(config-
line)#password cisco123
CustomerRouter(config-
line)#login
CustomerRouter(config-line)#exit
CustomerRouter(config)#
```

Step 4: Configure the vty password to allow Telnet access to the router.

a. In global configuration mode, switch to line configuration mode to specify the vtylines.

CustomerRouter(config)#**line vty 0 4**

Set the password to **cisco123**, require that the password be entered at login, exit line configuration mode, and then
exit the configuration session.

```
CustomerRouter(config-
line)#password cisco123
CustomerRouter(config-
line)#login
CustomerRouter(config-line)#exit
CustomerRouter(config)#
```

Step 5: Configure password encryption, a MOTD banner, and turn off domain server lookup.

a. Currently, the line passwords and the enable password are shown in clear text when you show therunning configuration. Verify this now by entering

the **show running-config** command.

To avoid the security risk of someone looking over your shoulder and reading the passwords, encryptall clear text passwords.

CustomerRouter(config)#**service password-encryption**

Use the **show running-config** command again to verify that the passwords are encrypted.

To provide a warning when someone attempts to log in to the router, configure a MOTD banner.

CustomerRouter(config)#**banner motd $Authorized Access Only!$**

Test the banner and passwords. Log out of the router by typing the **exit** command twice. The banner displays beforethe prompt for a password. Enter the password to log back into the router.

You may have noticed that when you enter a command incorrectly at the user or privileged EXEC prompt, the routerpauses while trying to locate an IP address for the mistyped word you entered. For example, this output shows what happens when the **enable** command is mistyped.

CustomerRouter>**emable**
Translating "emable"...domain server (255.255.255.255)

To prevent this from happening, use the following command to stop all DNS lookups from the routerCLI.

CustomerRouter(config)#**no ip domain-lookup**

Save the running configuration to the startup configuration.

CustomerRouter(co
nfig)#**end**
CustomerRouter#**co**
py run start

Step 6: Verify the configuration.

a. Log out of your terminal session with the Cisco 1841 customer router.
b. Log in to the Cisco 1841 Customer Router. Enter the console password when prompted.
c. Navigate to privileged EXEC mode. Enter the privileged EXEC password when prompted.
d. Click the **Check Results** button at the bottom of this instruction window to check your work.

5. Design and implement the following experiments using packet tracer software

I. Simulation of network topologies

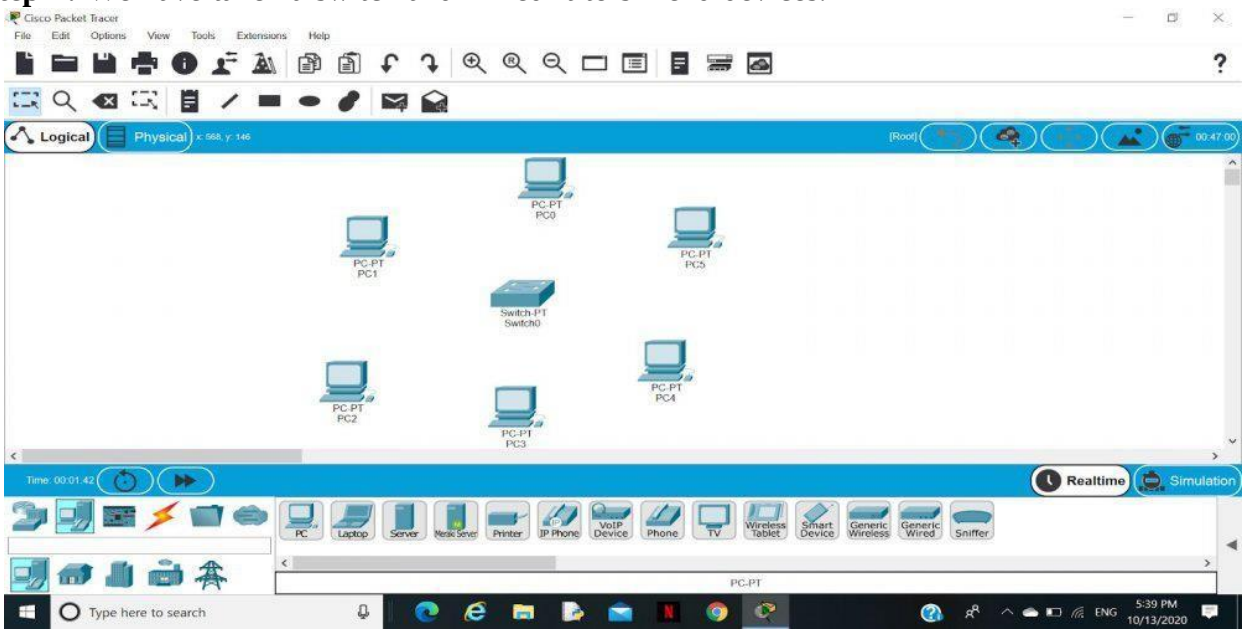
II. Configuration of network using different routing protocols

A star topology for a Local Area Network (LAN) is one in which each node is connected to a central connection point, such as a hub or switch. Whenever a node tries to connect with another node then the transmission of the message must be happening with the help of the central node. The best part of star topology is the addition and removal of the node in the network but too many nodes can cause suffering to the network.

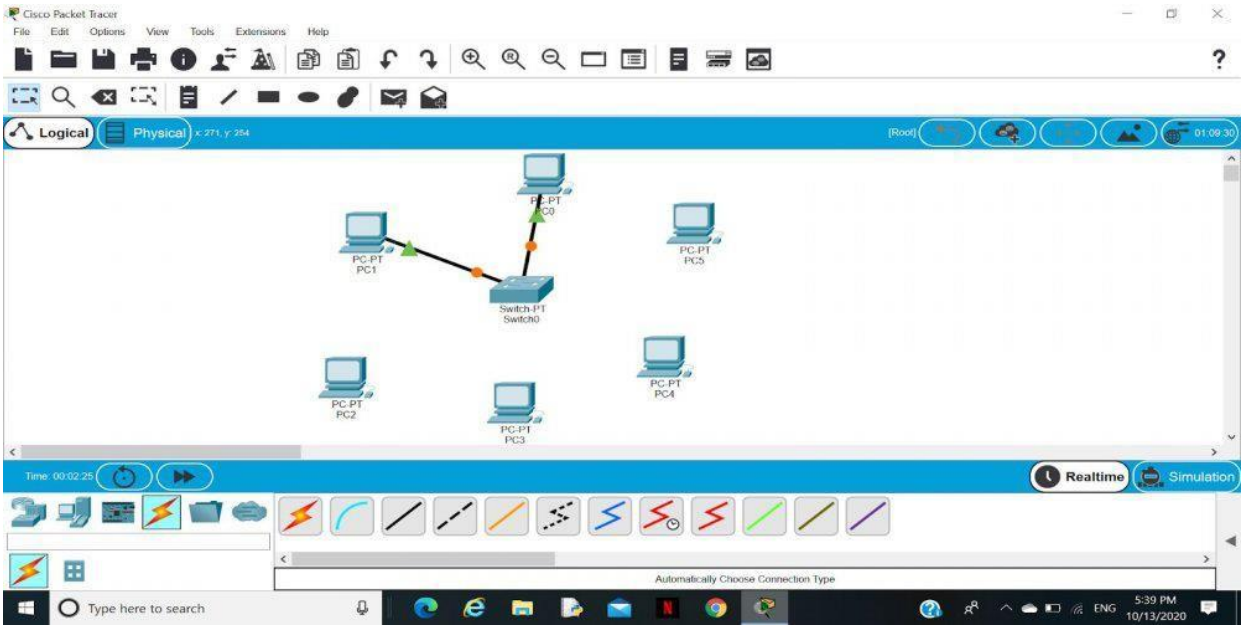
A Cisco packet tracer is a simulation tool that is used for understanding the networks. The best part of the Cisco packet tracer is its visualization you can see the actual flow of the message and understand the workflow of the network devices. Implementation of Star Topology using Cisco Packet Tracer is done using Switch.

Steps Implementing Star Topology using Cisco Packet Tracer:

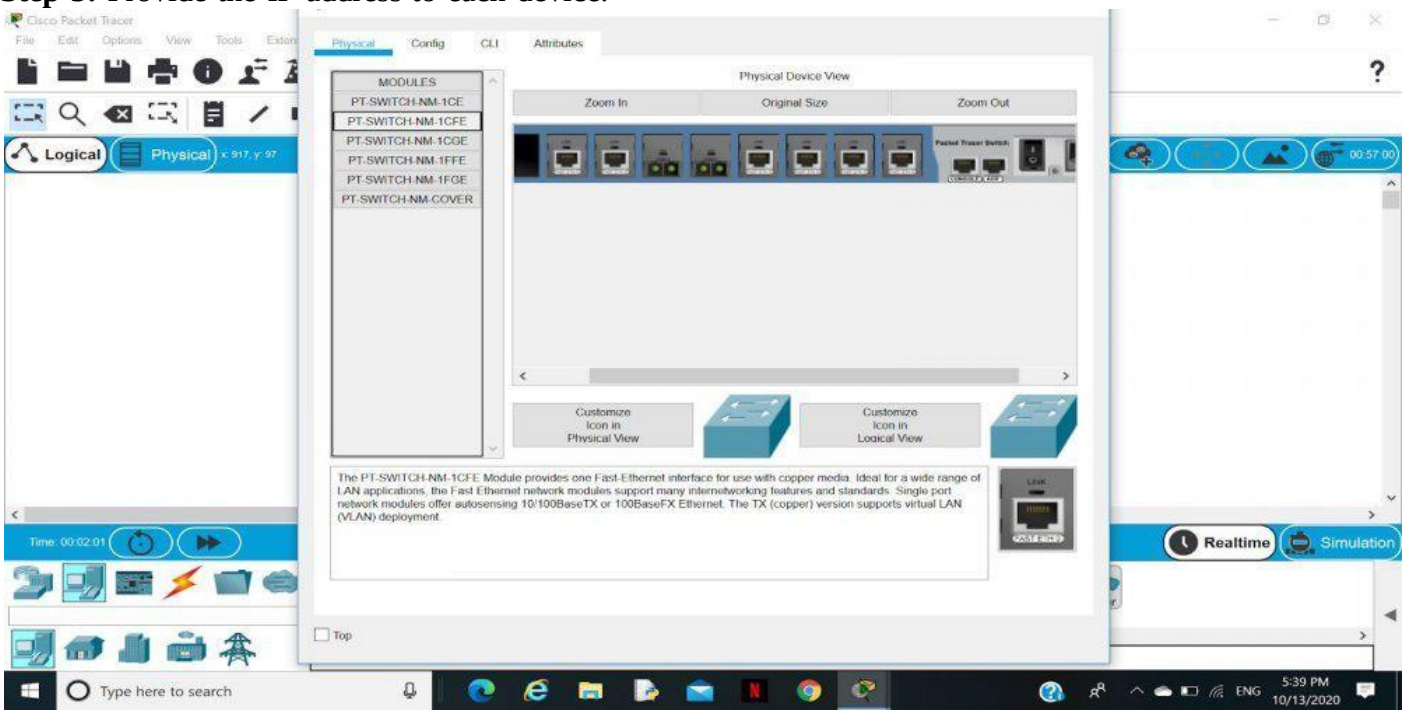
Step 1: We have taken a switch and linked it to six end devices.



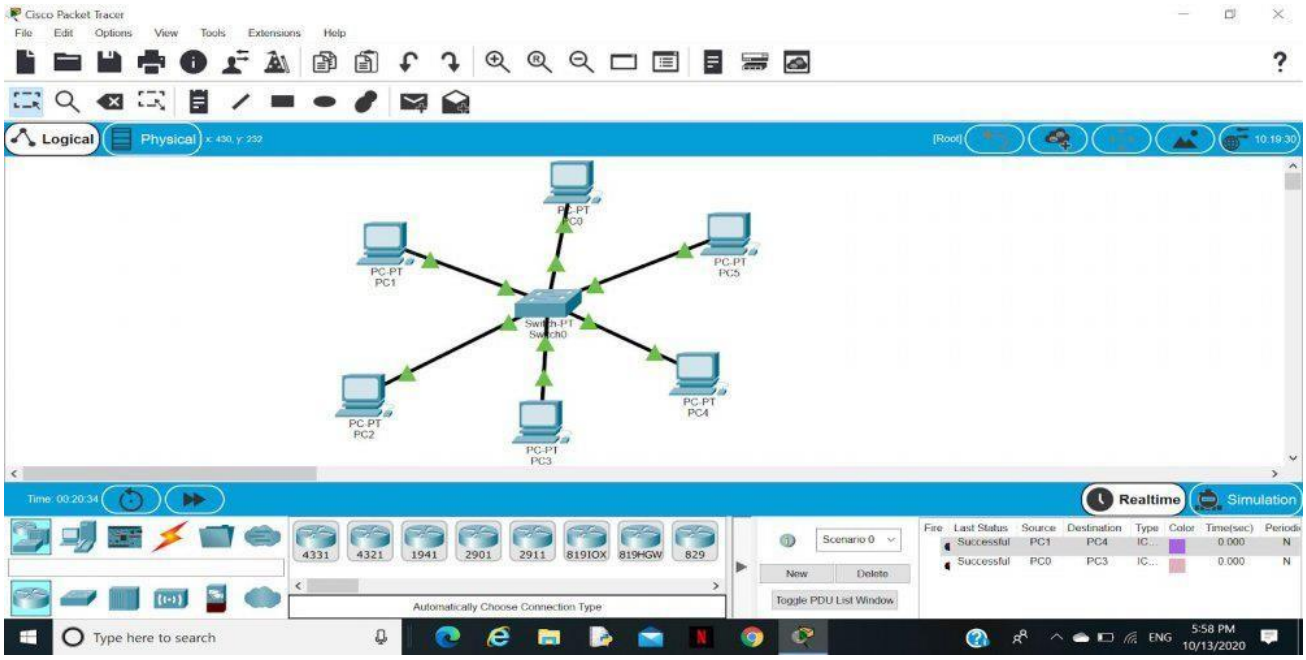
Step 2: Link every device with the switch.



Step 3: Provide the IP address to each device.



Step 4: Transfer message from one device to another and check the Table for Validation.



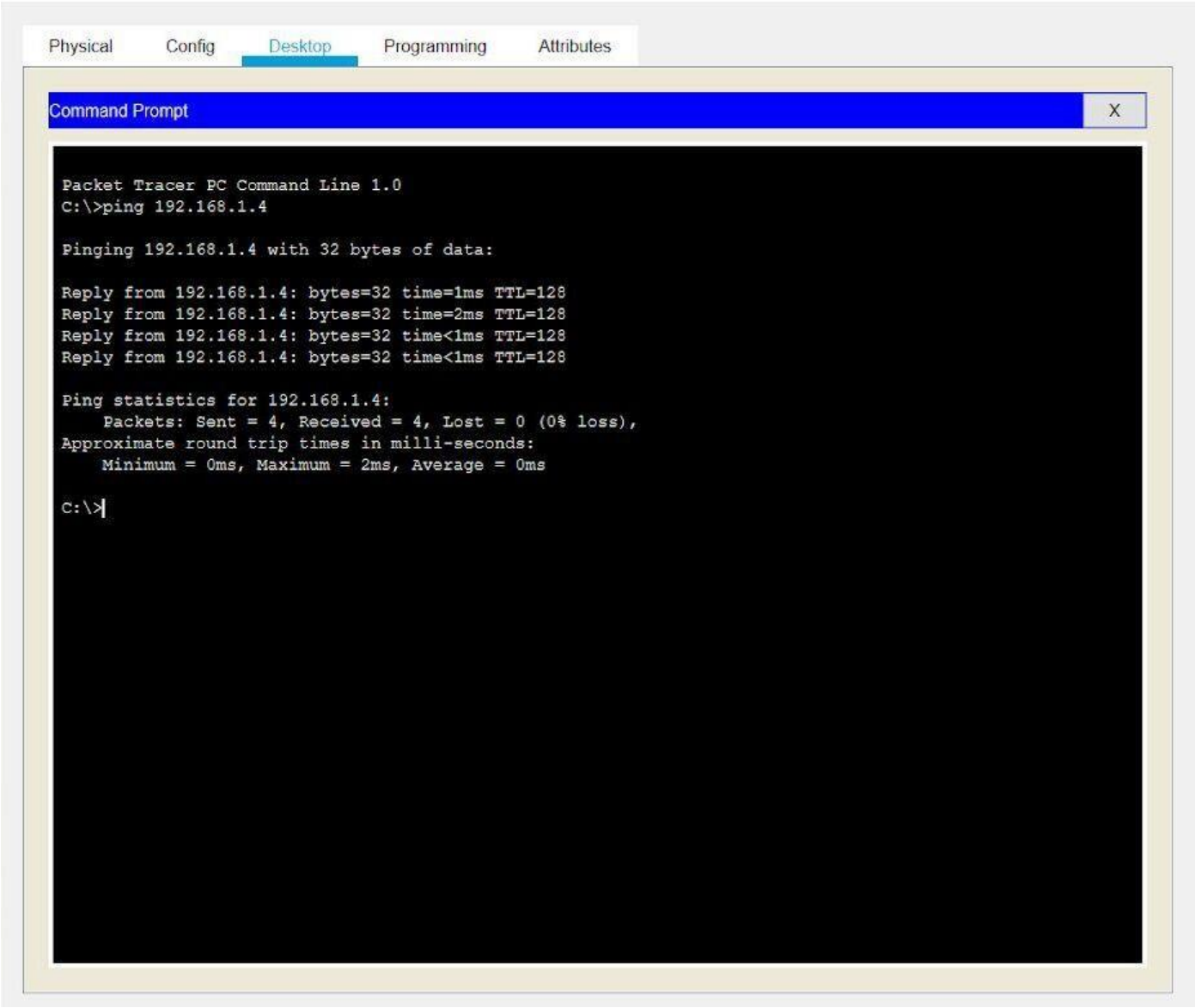
Now to check whether the connections are correct or not try to ping any device and the image below is doing the same.

To do ping one terminal of one device and run the following command:

Command:
"ping ip_address_of _any_device"

Example:
ping 192.168.1.4

Note:If the connections are correct then you will receive the response.

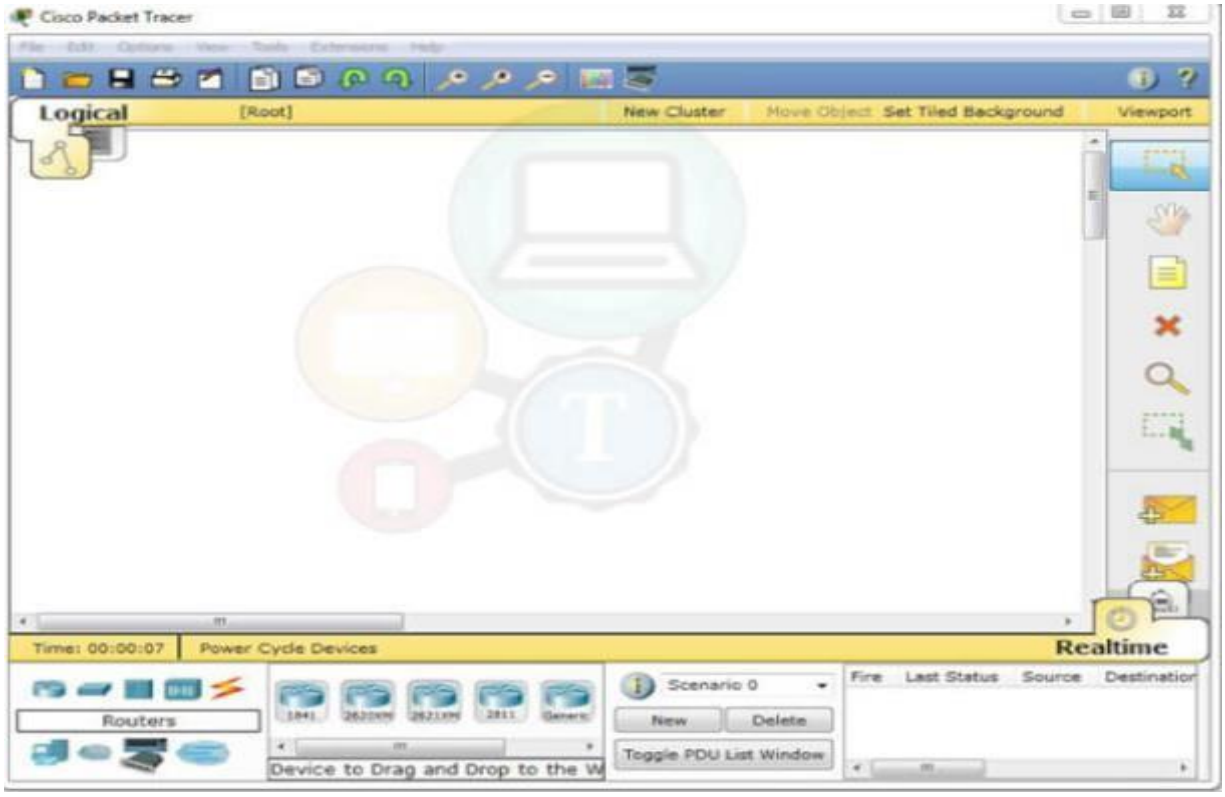


Mesh Topology

Mesh Topology is one of the most reliable, efficient, fast and costly type of network topology. In Mesh Topology Network, each network node or computer or network device has a dedicated and direct connection to every other network device present in computer network.

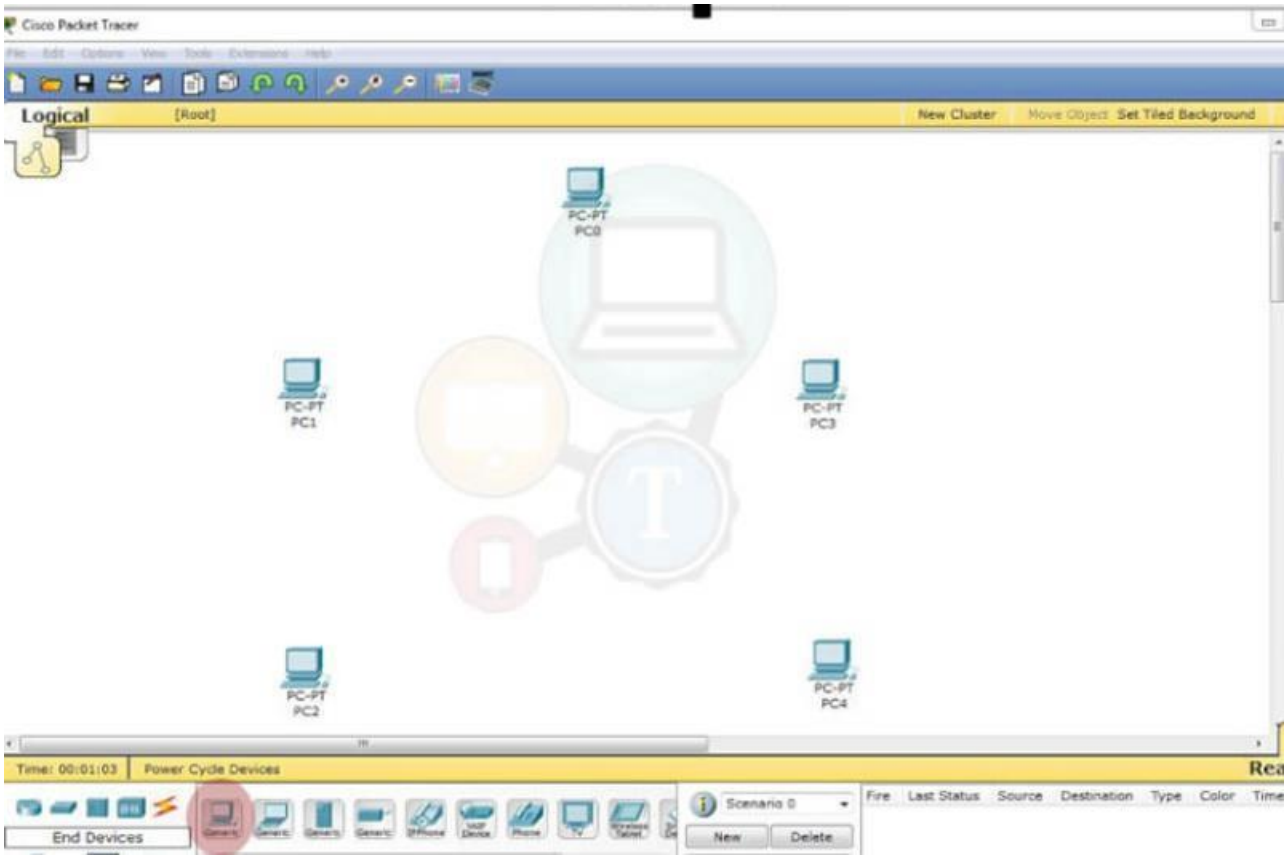
Open Cisco Packet Tracer (Step # 1)

After fulfilling the prerequisites you are now good to go for this tutorial. In first step, you will need to just find out Cisco Packet Tracer icon and double click it. we will see the user interface as given in following picture.



Select and Draw End Devices of Your Mesh Network (Step # 2)

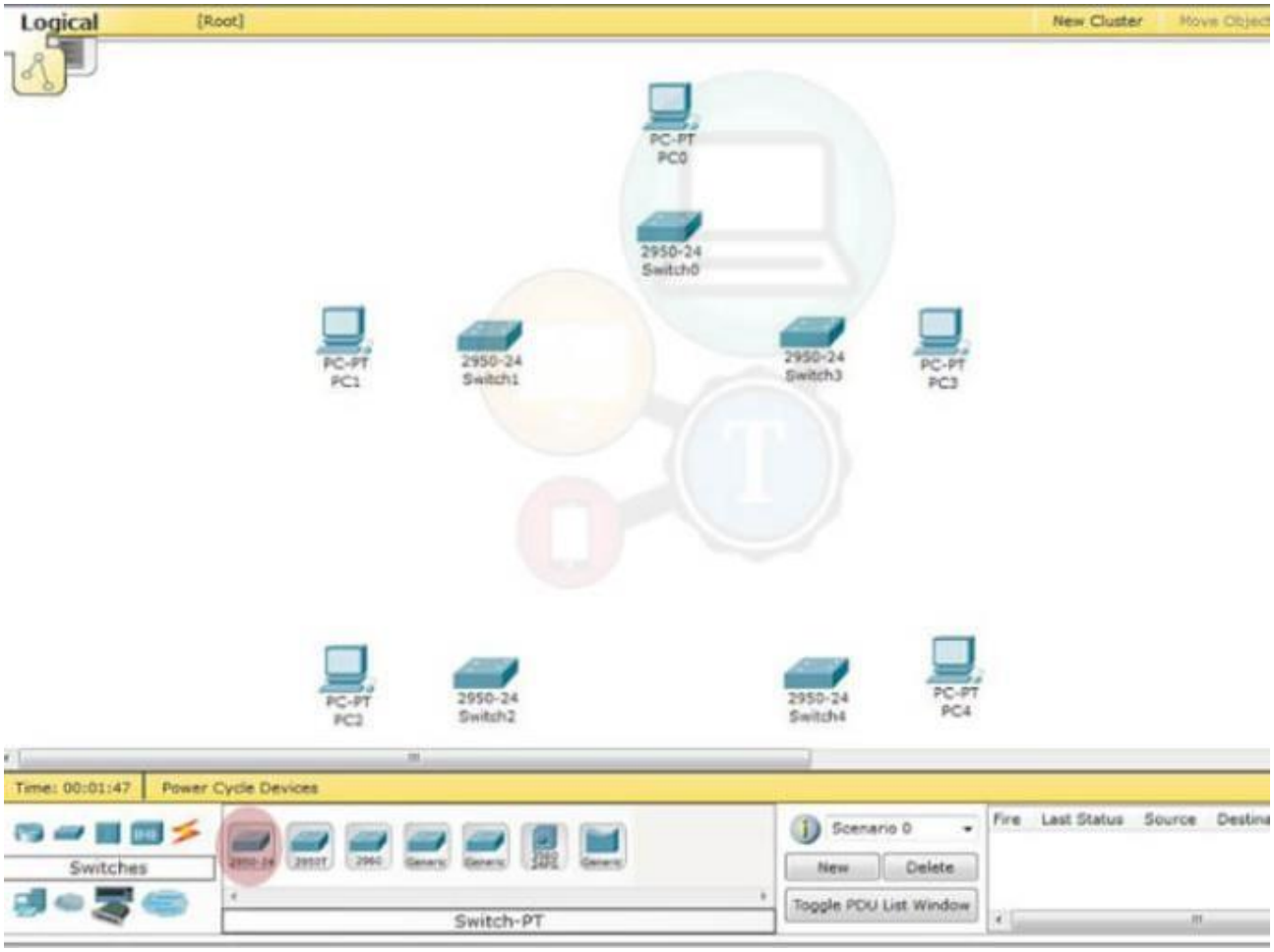
First thing that you will need for creating mesh network in Cisco packet tracer is obviously the end devices. For this go to the end devices menu in cisco packet tracer. we'll find a lot of devices there. Use simple desktop PC as end devices.create mesh topology using any number of end devices. A total of five end devices are used.



Select and Draw Switches for each End Device (Step # 3)

You can't connect end devices directly with each other. For this we will need a communication device like hub or switch. For the sake of simplicity, I am using switch for connecting end devices. You can find communication devices from tools menu that can be found in bottom left corner.

Simply go to the toolbox and select switches. We can find different models of switches,use any of them. The number of switches must be equal to the number of end devices.

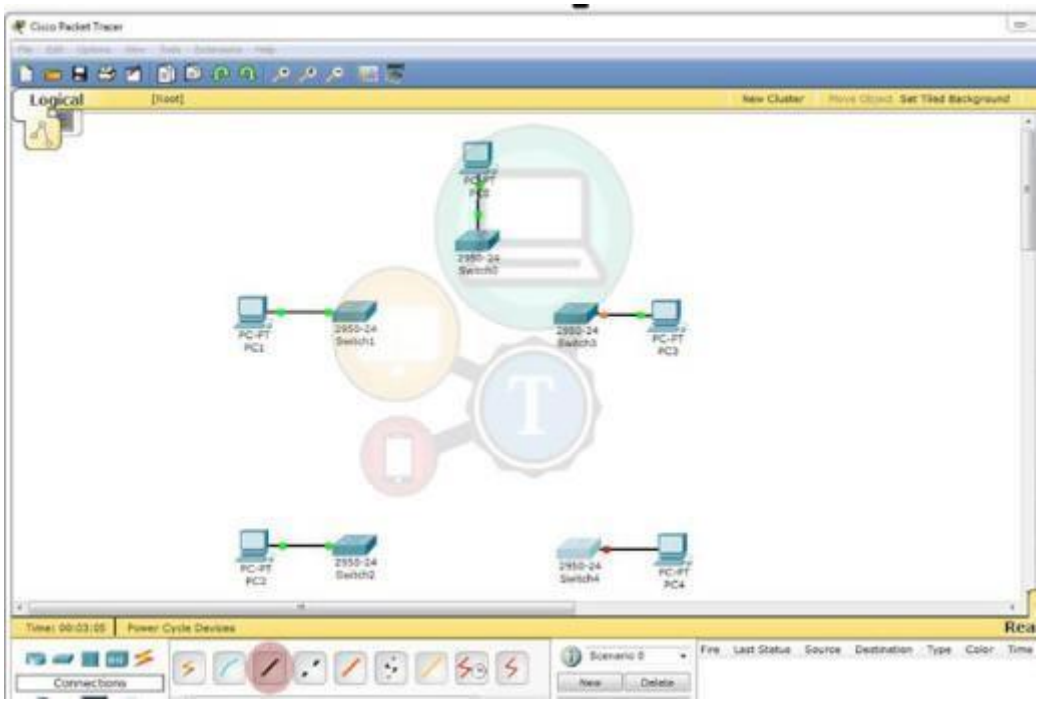


Connect Each Pair of Computer and Switch (Step # 4)

Here comes the connection setup. Go to the toolbox (as used earlier) and select the connections menu item. There are different sort of connecting media in there. Select copper-straight through cable for connecting each pair of switch and end device together. Connect each of the end device with corresponding switch turn by turn.

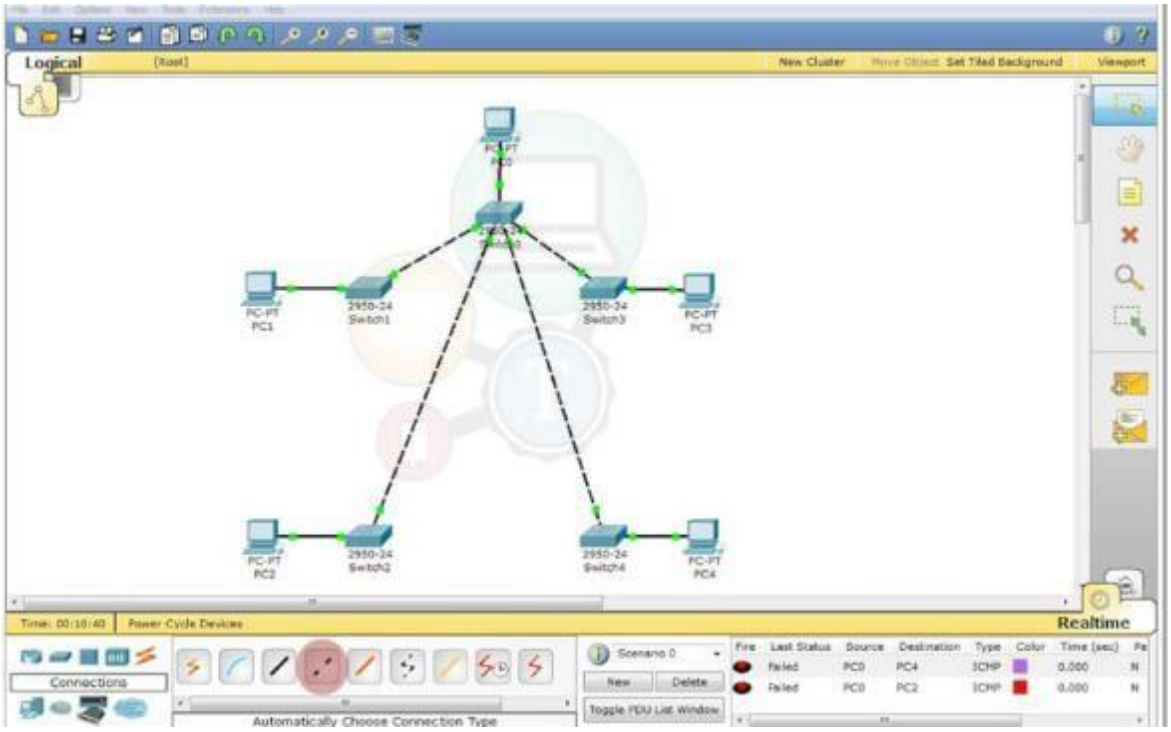
NOTE:

We are using copper straight through UTP cable for making connections. So, use 'Fast Ethernet' connecting option while connecting end device to the switch.



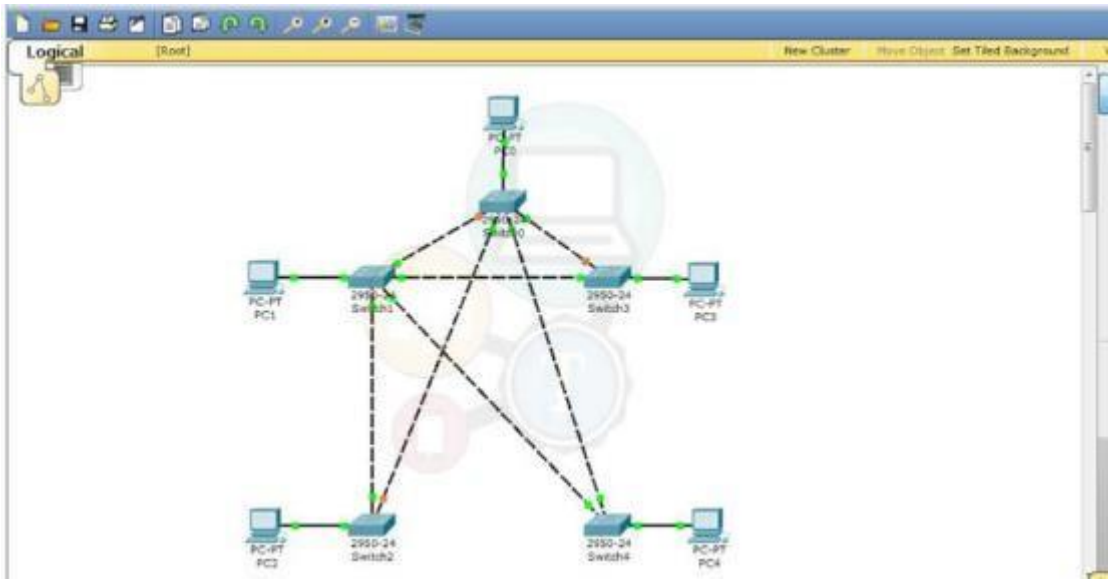
Select First Random Switch and Connect it to Other Remaining Switches (Step # 5)

Now we're done with connecting end devices with switches. So, now its the time to connect switches together. Select any random switch and connect it with each other switch in your mesh network. In this case, I have selected 'Switch 0' first.connect it to other remaining four switches using copper cross over cable. After connections, your network will look like this.



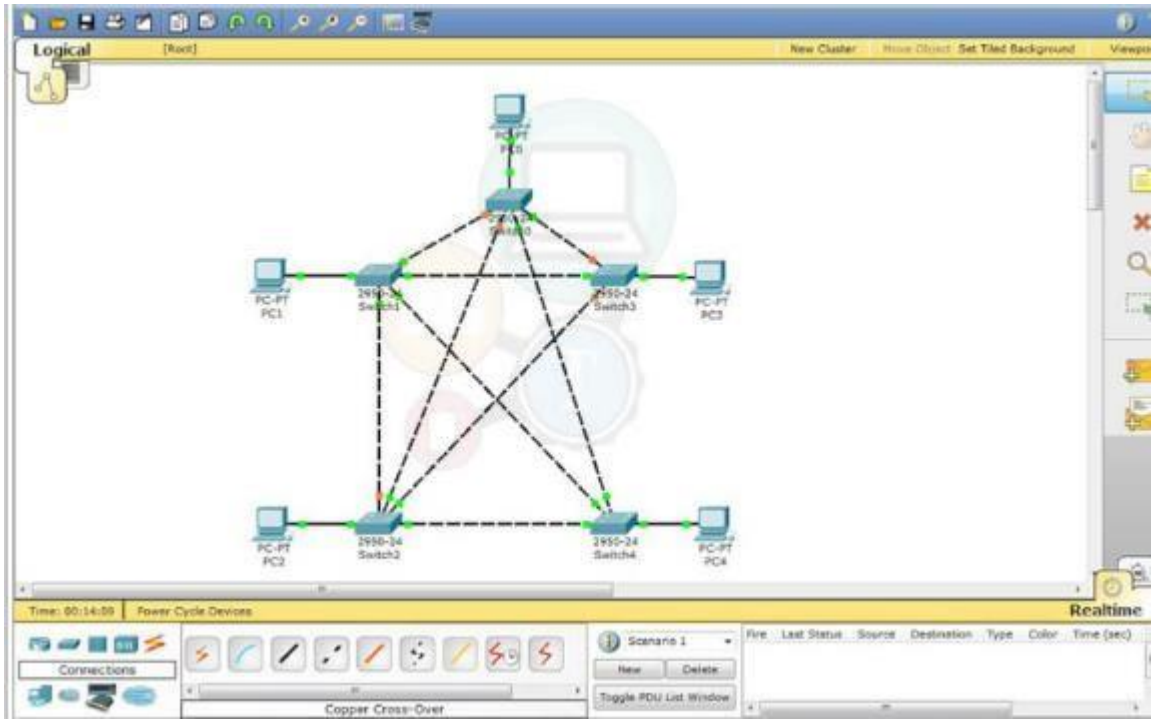
Select Second Random Switch and Connect it to Other Remaining Switches (Step # 6)

Select the second random switch and connect it with each other switch in your mesh network. In this case, I am selecting 'Switch 1'. So, I am connecting it to other remaining four switches using copper cross over cable.



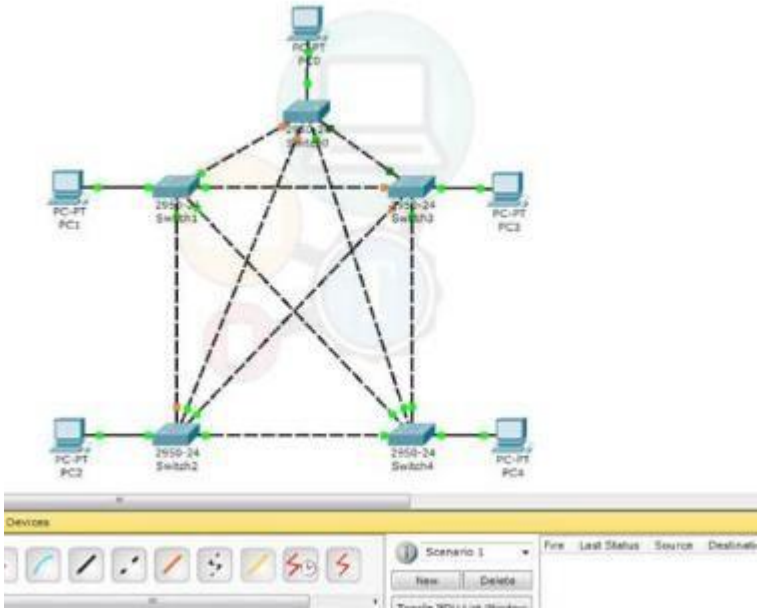
Select Third Random Switch and Connect it to Other Remaining Switches (Step # 7)

Now, select 'Switch 2' as third random switch and connecting it to two of the remaining switches using copper cross over cable. mesh topology network will look like this.



Choose Fourth Switch Randomly and Connect it to Remaining Switches (Step # 8)

In this step, select the fourth random switch and connect it to remaining switches. designate 'Switch 3' as fourth random switch..

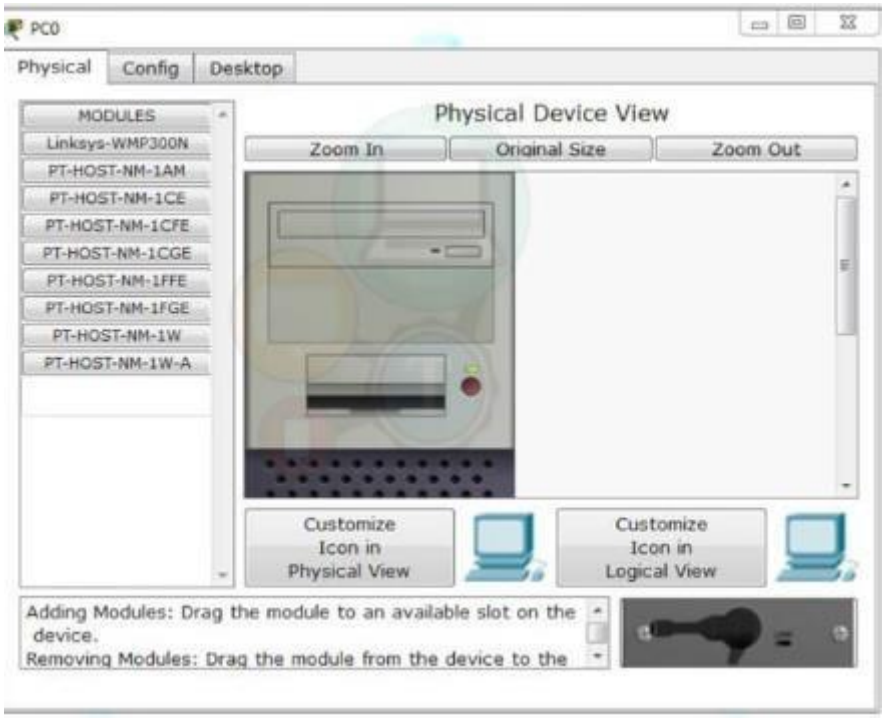


Now, each switch has a dedicated connection to every other one so you don't need to make any further connections. All the devices are connected and you can see the mesh network structure clearly.

In the coming steps, we will need to configure IP addresses of each end device. So, you need to repeat step 9 to step 12 for each individual end device.

Open End Device Configurations (Step # 9)

For configuring IP address, you will need to open end device configuration. This can be done easily by clicking on the end device. Nevertheless, you will see the following configuration screen.



Go To Desktop (Step # 10)

You will need to look for 'IP Configuration' option. This option is present under the 'Desktop' tab. So, go ahead and click on that to open desktop for enhanced configuration settings.

IP Configuration Settings (Step # 11)

This is the main step. In the coming step, you are going to learn about assigning IP address to end device. But before that you will need to open IP configuration settings. For this, please go ahead and click on the 'IP Configuration' option. This will open up a dialog box which is shown in the following picture.

Configure IP Address (Step # 12)

Enter the following configuration data. But please do remember, that for each device you will have to use different IP address. You can use the following list of IP addresses:

Switch	IP Address
Switch 0	192.168.1.1
Switch 1	192.168.1.2
Switch 2	192.168.1.3
Switch 3	192.168.1.4
Switch 4	192.168.1.5
Switch 5	192.168.1.6

6. Do the following using NS2/NS3/NetSim or any other equivalent tool

I. Simulation of Congestion Control Algorithms

II. Simulation of Routing Algorithms

Congestion refers to a network state where a node or link carries so much data that it may deteriorate network service quality, resulting in queuing delay, frame or data packet loss and the blocking of new connections.

what is congestion-control? In Congestion-control, end systems throttle back in order to avoid congesting the network. The mechanism is similar to end-to-end flow controls, but the intention is to reduce congestion in the network, not the receiver.

Network simulator 2 (Ns-2 Program for congestion-control outputs better results).

Techniques available in congestion-control:

Open Loop Technique and closed Loop Technique are utilized in ns-2 program for congestion control.

Open loop

Acknowledgement policy.

Retransmission policy.

Discarding policy.

Window policy.

Admission policy.

Closed loop

Explicit feedback.

Back pressure.

Implicit feedback.

Choke packet.

Architecture-for-control-congestion

Architecture-for-control-congestion

```
# Create a new Tcl script file, e.g., congestion_control.tcl
# Set up network topology
set ns [new Simulator]
# Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
# Create links
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
# Set congestion control algorithm
$ns queue-limit $n0 $n1 10
$ns queue-limit $n1 $n2 10
$ns tcp         $n0 $n1 Reno
$ns tcp         $n1 $n2 Reno
# Define traffic sources and destinations
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
```

```
set sink0 [new Agent/TCPSink]
$ns attach-agent $n2 $sink0
$ns connect $tcp0 $sink0
# Set simulation parameters
$ns tcp-rfc1323 true
$ns trace-all $tcp0
$ns trace-all $sink0
$ns namtrace-all-wireless $ns namtrace-all-mac $ns namtrace-all-queue
# Set simulation duration
set sim_duration 10.0
# Define a procedure to end the simulation
proc finish {} {
   global ns sim_duration
   $ns flush-trace
   $ns nam-end-wireless $sim_duration
   exit 0
}
# Schedule the simulation end
$ns at $sim_duration "finish"
# Run the simulation
$ns run
}
```

Routing Algorithm

Distance Vector Routing is one of the routing algorithm in a Wide Area Network for computing shortest path

between source and destination. The Router is one main devices used in a wide area network. The main task of

the router is Routing. It forms the routing table and delivers the packets depending upon the

routes in the tableeither directly or via an intermediate devices.

Each router initially has information about its all neighbors. Then this information will be shared among nodes.

ALGORITHM:

1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace
file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose distance vector routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

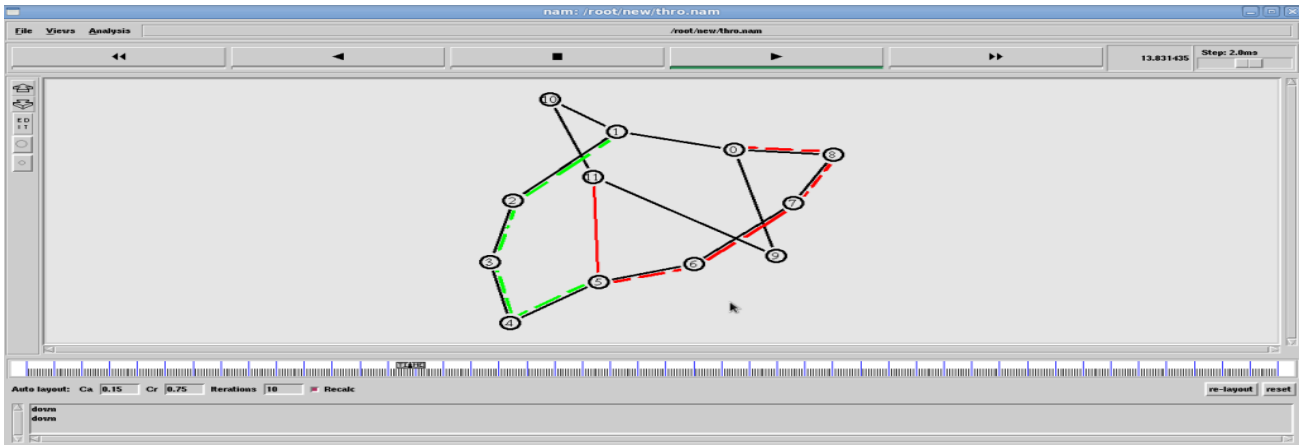
PROGRAM:

```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]
$ns namtrace-all $nf
proc finish { } {
global ns nr nf
 $ns flush-trace
close $nf
close $nr
exec nam thro.nam &
```

```
exit 0
 }
for { set i 0 } { $i < 12} { incr i 1 } {
set n($i) [$ns node]}
for {set i 0} {$i < 8} {incr i} {
$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail }
$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0
set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
```

```
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0
$ns rtproto DV
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)
$udp0 set fid_ 1
$udp1 set fid_ 2
$ns color 1 Red
$ns color 2 Green
$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"
$ns at 45 "finish"
$ns run
```

OUTPUT



7. Socket programming using UDP and TCP (e.g simpleDNS, date&time client/server, echo client/server, iterative & concurrent servers)

(a) Implementation of Domain Name Server.

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

```
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
int main(int argc,char *argv[])
{
struct hostent *h;
if(argc!=2)
{ /*error check the command line*/
     printf(stderr,"usage:getpid address\n");
     exit(1);
}
if((h=gethostbyname(argv[1]))==NULL)
{ /*get the host info*/
      perror("gethostbyname");
      exit(1);
}
printf("Host name : %s\n",h->h_name);
printf("IP Address : %s\n",inet_ntoa(*((struct in_addr *)h->h_addr)));
return 0;
}
```

OUTPUT:

$ cc dns.c

```
dns.c: In function `main':
dns.c:12: warning: passing arg 1 of `printf' from incompatible pointer type
```

$ ./a.out cc5

```
Host name : cc5
IP Address : 127.0.0.1
```

(b) Implementation of Iterative Daytime server using Connection-Oriented(TCP).

Description: This program implements server as a iterative Daytime server, which sends the day and time to the client . This uses the connection oriented concept for implementing, which means first it establishes the connection, uses the connection for communication and closes it after the communication is ended. This uses TCP protocol.

Server side :

i. First creates the socket using socket system call.

ii. Binds the name (i.e. IP address, port number and family) of the server to the socket.
iii. Makes the socket to be ready in passive state by creating queues for storing the client requests using listen system call. In this server is listening for the clients.
iv. Once the connect request comes from the client, it stores in the queue and calls accept system call for establishing the connection. This returns the new connection id, which is used while sending and receiving the messages.
v. Extracts the system time using time and ctime system call.
vi. Sends the day and time to the client.
vii. Close the communication.

Client side :

i. First creates the socket.
ii. Initializes the server address to sockaddress_in structure to which connect request has to send.
iii. Makes a connection request to the server using connection system call. If the connection is established at the server it gets the connection id, which is used for the communication. Otherwise it returns negative number as an error.
iv. If connection is established, receives the day and time message from the server.

Server program

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include <time.h>
#include<strings.h>
main()
{
  unsigned long t;
  time_t tick;
  char *st,buffer[512];
  int n,i,l,ml,sockfd,newsockfd;
```

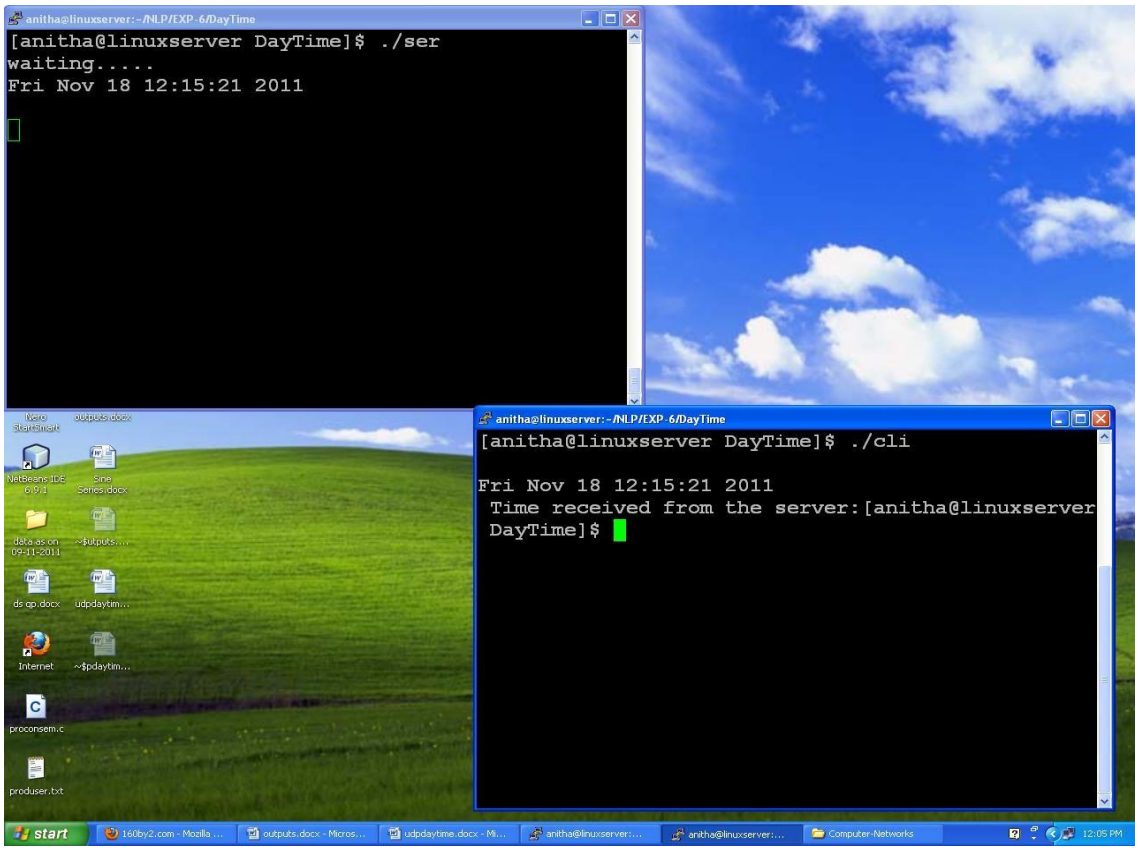
```
  struct sockaddr_in serv_addr,cli_addr;
  sockfd=socket(AF_INET,SOCK_STREAM,0);
  serv_addr.sin_family=AF_INET;
  serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
  serv_addr.sin_port=htons(5555);
  bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
  listen(sockfd,5);
  puts("waiting.... ");
  l=sizeof(cli_addr);
  for(i=0;i<5;i++)
  {
    newsockfd=accept(sockfd,(struct sockaddr *)&cli_addr,&l);
    printf("\nConnected to client");     t=time(&t);
   //snprintf(buffer,sizeof(buffer),"%.24s\r\n",ctime(&tick));
    buffer=ctime(&t);
    //strcpy(buffer,st);
puts(buffer);
n=write(newsockfd,buffer,strlen(buffer));
    close(newsockfd);
  }
}
```

client program

```
#include<string.h>
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
main()
{
  int sockfd,n;
```

```
    char buffer[512];
    struct sockaddr_in serv_addr;
    bzero((char *)&serv_addr,sizeof(serv_addr));
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
    serv_addr.sin_port=htons(5555);
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
    n=read(sockfd,buffer,512,0);
    printf("\n Time received from the server:%s",buffer);
    close(sockfd);
}
```

OUTPUT:



(c) Implementation of Iterative Daytime server using connectionless service(UDP).

Description: This program implements server as a iterative Daytime server, which sends the day and time to the client . This uses the connection less concept for implementing, which means it doesn't establishes the connection. This uses UDP protocol.

Server side :

i. First creates the socket using socket system call.
ii. Binds the name (i.e. IP address, port number and family) of the server to the socket.
iii. Server blocks for receiving the message from the client by using recv from system call.
iv. Extracts the time from the system and sends to the client.

Client side :

i. First creates the socket.
ii. Initializes the server address to sockaddress_in structure to which connect request has to send.
iii. Sends the message to server using send to system call by specifying server address along with the message.
iv. Then waits for the day and time message from the server with recv from system call.

server program

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<strings.h>
#include<strings.h>
main()
{
  unsigned long t;
  char *st,buffer[512];
```

```
  int n,i,l,ml,sockfd,newsockfd;
  struct sockaddr_in serv_addr,cli_addr;
  sockfd=socket(AF_INET,SOCK_DGRAM,0);
  serv_addr.sin_family=AF_INET;
  serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
  serv_addr.sin_port=htons(5555);
  bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
  l=sizeof(cli_addr);
     cliLen = sizeof(cliAdr);
     n = recvfrom(sockFd,buffer,MAX_MSG,0 ,(struct
sockaddr*)&cliAdr,&cliLen);
     if(n < 0)
     {
        errExit("recvfrom error \n");
     }
   Printf("\nMessage received from client: %s", buffer);
    t=time(&t);
    st=(char *)ctime(&t);
   strcpy(buffer,st);
   n=sizeof(buffer);
   if(sendto(sockFd,buffer, n,0,(struct sockaddr *)&cliAdr,cliLen) !=n)
     {      errExit("sendto error \n");
     }
Printf ("Time sent…");
  }
}
```

Client program

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

```
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#define SRV_IP_ADRS  "127.0.0.1"
#define SRV_UDP_PORT 8000
#define MAX_MSG      100
void errExit(char *str)
{
   puts(str);
   exit(0);
}
int main( )
{
   int sockFd;
   struct sockaddr_in srvAdr;
   char txmsg[MAX_MSG];
   char rxmsg[MAX_MSG];
   int n;
   if((sockFd = socket (AF_INET,SOCK_DGRAM,0))< 0)
   {        errExit("can't open datagram socket \n");
   }
   memset ( &srvAdr,0,sizeof(srvAdr));
   srvAdr.sin_family = AF_INET;
   srvAdr.sin_addr.s_addr = inet_addr(SRV_IP_ADRS);
   srvAdr.sin_port = htons (SRV_UDP_PORT);
   printf("Enter message to send :\n");
   fgets(txmsg,MAX_MSG,stdin);
   n = strlen(txmsg)+1;
   if(sendto(sockFd,txmsg,n,0,(struct sockaddr *)&srvAdr,sizeof(srvAdr ))!=n)
   {        errExit("sendto error \n");
```

```
    }
    n = recv(sockFd,rxmsg,MAX_MSG,0);
    printf("n=%d\n",n);
    if(n < 0)
    {       errExit("recv error \n");
    }
    printf("Time Received from the server :%s\n",rxmsg);
}
```

Output:

Server: Message received from client: Daytime
Time sent

Client:

Enter the message to send: Daytime
Time received from the server: FRI NOV 18 12:15:21 2011

(d) Implementation of iterative echo server using connection oriented socket system calls

Description: This program implements server as a iterative server, which receives the messages from the client and sends the same message as echo to the client. This uses the connection oriented concept for implementing, which means first it establishes the connection, uses the connection for communication and closes it after the communication is ended.

Server side :

i. First creates the socket using socket system call.
ii. Binds the name (i.e. IP address, port number and family) of the server to the socket.
iii. Makes the socket to be ready in passive state by creating queues for storing the client requests using listen system call. In this server is listening for the clients.
iv. Once the connect request comes from the client, it stores in the queue and calls accept system call for establishing the connection. This returnes the new connection id, which is used while sending and receiving the messages.
v. Receives the message from the client and echoes it back to the client.
vi. Close the system call after completing the communication.

Client side :

i. First creates the socket.
ii. Initializes the server address to sock address_in structure to which connect request has to send.

iii. Makes a connection request to the server using connection system call. If the connection is established at the server it gets the connection id, which is used for the communication. Otherwise it returns negative number as an error.

iv. If connection is established, sends and receives the messages using send and recv system calls.

Server Program

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#define    SRV_TCP_PORT  5556
#define    MAX_MSG       100
void errExit( char *str )
{
  puts(str);
  exit(0);
}
int main( )
{
    int  sockFd,newSockFd;
   struct  sockaddr_in  srvAdr,cliAdr;
   int cliLen,n;
   char  mesg[MAX_MSG];
   if ( ( sockFd= socket(AF_INET, SOCK_STREAM ,0) ) < 0 )
   {
      errExit ("can.t open stream socket \n");
   }
   memset(&srvAdr ,0,sizeof(srvAdr));  //initializes the srvAdr structure to zero
```

```
// Initialize the structure members to Family name, port no. and ip address
srvAdr.sin_family = AF_INET;
 srvAdr.sin_addr.s_addr = htonl(INADDR_ANY);
 srvAdr.sin_port = htons(SRV_TCP_PORT);
 if (bind (sockFd,(struct sockaddr *)&srvAdr, sizeof (srvAdr )) < 0)
 {
    errExit ("Can.t bind local address \n");
 }
 listen(sockFd,5);    // Waiting for the clients, max. 5 clients can be in waiting state
 while (1)
 {
    printf("server waiting for new connection :\n");
    cliLen = sizeof(cliAdr);
    newSockFd = accept(sockFd,( struct sockaddr *) &cliAdr, &cliLen );
    if(newSockFd < 0)
    {
       errExit ("accept error \n");
    }
    printf("connected to client :%s \n", inet_ntoa (cliAdr.sin_addr ));
    while(1)
    {
       n =recv(newSockFd, mesg,MAX_MSG,0);
       if ( n < 0)
       {
           errExit ("recv error \n");
       }
       if(n== 0)
       {
          close (newSockFd);
          break;
```

```
            }
            mesg [n] =0;
        if(send (newSockFd,mesg,n,0) !=n)
            {
                errExit("send error \n");
            }
            printf("Received following message :\n %s \n ",mesg);
          }
        }
    }
```

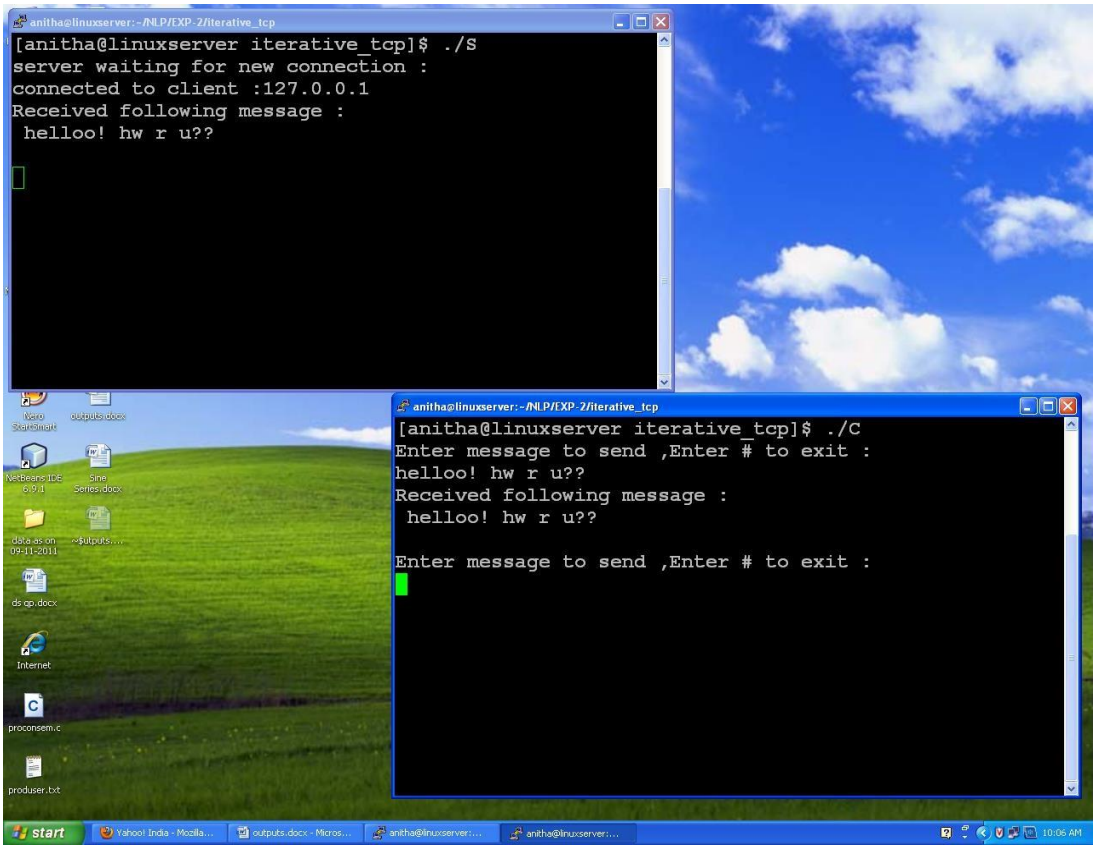
client Program

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#define  SRV_IP_ADRS     "127.0.0.1"
#define  SRV_TCP_PORT  5556
#define  MAX_MSG         100
void errExit(char *str)
{
   puts(str);
   exit(0);
}
int main ( )
{
   int  sockFd;
   struct sockaddr_in  srvAdr;
   char  txmsg[MAX_MSG];
```

```
char  rxmsg[MAX_MSG];
int n;
if( (sockFd =socket(AF_INET,SOCK_STREAM,0))<0)
{
   errExit("can't open stream socket \n");
}
memset(&srvAdr, 0,sizeof(srvAdr));
srvAdr.sin_family = AF_INET;
srvAdr.sin_addr.s_addr = inet_addr(SRV_IP_ADRS);
srvAdr.sin_port = htons(SRV_TCP_PORT);
if (connect(sockFd,(struct sockaddr *)&srvAdr,sizeof(srvAdr))<0)
{
   errExit("can't connect to server \n");
}
while(1)
{
   printf("Enter message to send ,Enter # to exit :\n");
   fgets(txmsg,MAX_MSG,stdin);
   if( txmsg[0] == '#')
   {
      break;
   }
   n = strlen(txmsg)+1;
   if ( send(sockFd,txmsg,n,0 ) != n)
   {
      errExit("send error \n");
   }
   n=recv(sockFd,rxmsg,MAX_MSG,0);
   if ( n < 0)
   {
```

```
            errExit("recv error \n");
        }
        printf("Received following message : \n %s \n",rxmsg );
    }
    close (sockFd);
}
```

OUTPUT:



(e) Implementation of iterative echo server using connectionless socket system calls

Description: This program implements server as a iterative server, which receives the messages from the client and sends the same message as echo to the client. This uses the connection less concept for implementing, which means no connection establishes before the communication, in this each message is attached with the address of the server because there is no fixed path, each message uses its own path to reach to destination. This communication is not reliable and it delivers fast.

Server side :

i. First creats the socket using socket system call.
ii. Binds the name (i.e. IP address, port number and family) of the server to the socket.
iii. Server blocks for receiving the message from the client by using recvfrom system call.
iv. Sends the message by echoing to client using sendto system call.

Client side :

v. First creates the socket.
vi. Initializes the server address to sockaddress_in structure to which connect request has to send.
vii. Sends the message to server using sendto system call byspecifying serveraddress along with the message.
viii. Then waits for the echo message from the server with recvfrom system call.

Server program:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdlib.h>
#define SRV_UDP_PORT 8000
#define MAX_MSG      100
void errExit(char *str)
{
   puts(str);
   exit(0);
}
int main( )
{
   int sockFd;
   struct sockaddr_in srvAdr,cliAdr;
```

```
int cliLen,n;
char mesg[MAX_MSG];
if((sockFd = socket(AF_INET,SOCK_DGRAM,0)) < 0 )
{
   errExit("Can't open datagram socket \n");
}
memset (&srvAdr ,0,sizeof (srvAdr));
srvAdr.sin_family = AF_INET;
srvAdr.sin_addr.s_addr = htonl (INADDR_ANY);
srvAdr.sin_port = htons(SRV_UDP_PORT);
if(bind(sockFd,(struct sockaddr*)&srvAdr,sizeof(srvAdr))<0)
{
   errExit ("can't bind local address \n");
}
printf("server waiting for messages \n");
while(1)
{
   cliLen = sizeof(cliAdr);
   n = recvfrom(sockFd,mesg,MAX_MSG,0 ,(struct sockaddr*)&cliAdr,&cliLen);
   if(n < 0)
   {          errExit("recvfrom error \n");
   }
   if(sendto(sockFd,mesg,n ,0,(struct sockaddr *)&cliAdr,cliLen) !=n)
   {          errExit("sendto error \n");
   }
   printf("Received following message from
clients\n%s\n",inet_ntoa(cliAdr.sin_addr),mesg);
}}
```

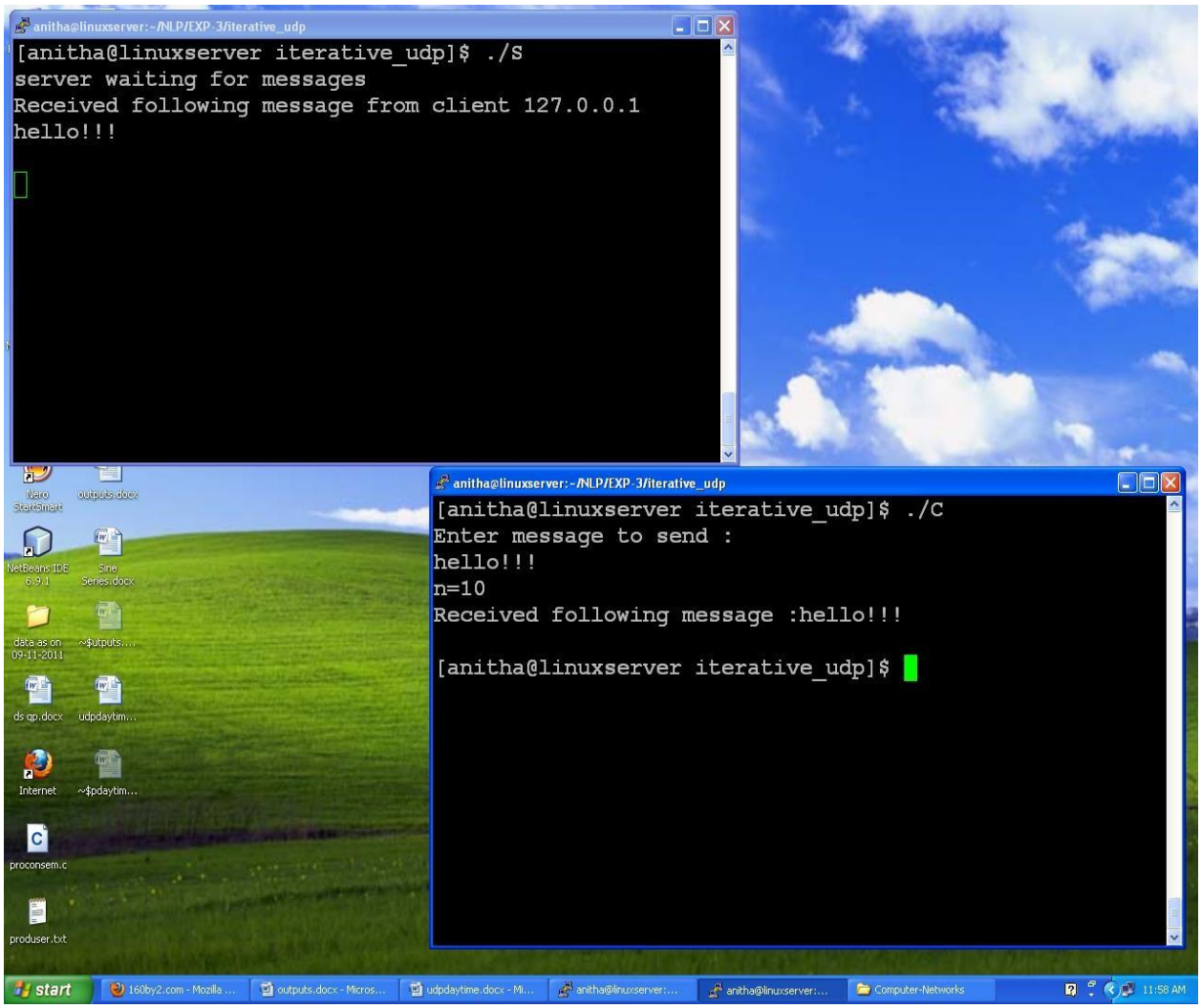
Client program

```
#include <stdio.h>
#include <sys/types.h>
```

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#define SRV_IP_ADRS  "127.0.0.1"
#define SRV_UDP_PORT 8000
#define MAX_MSG      100

void errExit(char *str)
{
   puts(str);
   exit(0);
}
int main( )
{
   int sockFd;
   struct sockaddr_in srvAdr;
   char txmsg[MAX_MSG];
   char rxmsg[MAX_MSG];
   int n;
   if((sockFd = socket (AF_INET,SOCK_DGRAM,0))< 0)
   {
      errExit("can't open datagram socket \n");
   }
   memset ( &srvAdr,0,sizeof(srvAdr));
   srvAdr.sin_family = AF_INET;
   srvAdr.sin_addr.s_addr = inet_addr(SRV_IP_ADRS);
   srvAdr.sin_port = htons (SRV_UDP_PORT);
   printf("Enter message to send :\n");
```

```
    fgets(txmsg,MAX_MSG,stdin);
    n = strlen(txmsg)+1;
    if(sendto(sockFd,txmsg,n,0,(struct sockaddr *)&srvAdr,sizeof(srvAdr ))!=n)
    {       errExit("sendto error \n");
    }
    n = recv(sockFd,rxmsg,MAX_MSG,0);
    printf("n=%d\n",n);
    if(n < 0)
    {
       errExit("recv error \n");
    }
    printf("Received following message :%s\n",rxmsg);
}
```

OUTPUT:



(f) Implementation of concurrent echo server using connection oriented socket system calls

Description: This program implements server as a concurrent server, which receives the messages from the client and sends the same message as echo to the client. In this it creates a child process using fork system call for each client request. This child process service the client request. In this way it services multiple clients at the same time. This uses the connection oriented concept for implementing, which means first it establishes the connection, uses the connection for communication and closes it after the communication is ended.

Server side :

i. First creates the socket using socket system call.

ii. Binds the name (i.e. IP address, port number and family) of the server to the socket.

iii. Makes the socket to be ready in passive state by creating queues for storing the client requests using listen system call. In this server is listening for the clients.
iv. Once the connect request comes from the client, it stores in the queue and calls accept system call for establishing the connection. This returns the new connection id, which is used while sending and receiving the messages.
v. Creates the child process using fork system call.
vi. Receives and sends the messages from/to the client in the child process.
vii. Parent process waits for the next client.
viii. Close the system call after completing the communication.

Client side :

i. First creates the socket.
ii. Initializes the server address to sockaddress_in structure to which connect request has to send.
iii. Makes a connection request to the server using connection system call. If the connection is established at the server it gets the connection id, which is used for the communication. Otherwise it returns negative number as an error.
iv. If connection is established, sends and receives the messages using send and recv system calls.

Server program

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#define SRV_TCP_PORT 8888
#define MAX_MSG      100
void errExit( char *str )
{
   puts(str);
   exit(0);
```

```
}
int main( )
{
   int sockFd,newSockFd;
   struct sockaddr_in srvAdr,cliAdr;
   int cliLen,n;
   char mesg[MAX_MSG];
   int pid;
   if( (sockFd = socket(AF_INET, SOCK_STREAM ,0) ) < 0 )
   {
      errExit("can't open stream socket \n");
   }
   memset(&srvAdr ,0 , sizeof(srvAdr));
   srvAdr.sin_family = AF_INET;
   srvAdr.sin_addr.s_addr = htonl(INADDR_ANY);
   srvAdr.sin_port = htons(SRV_TCP_PORT);
   if(bind(sockFd,(struct sockaddr*)&srvAdr,sizeof (srvAdr )) < 0)
   {
      errExit("Can't bind local address \n");
   }
   listen(sockFd,5);
   while(1)
   {
      printf("server waiting for new connection :\n");
      cliLen = sizeof(cliAdr);
      newSockFd = accept(sockFd,(struct sockaddr*) &cliAdr, &cliLen );
      if (newSockFd < 0)
      {
         errExit("accept error \n");
      }
```

```
printf("connected to client :%s \n",inet_ntoa (cliAdr.sin_addr));
pid=fork( );
printf("parent\n");
if(pid == 0)  /*** child process ***/
{
   printf("child");
    while(1)
      {
       n = recv(newSockFd, mesg,MAX_MSG,0);
       if(n < 0)
       {
          errExit("recv error \n");
       }
       if(n == 0)
       {
          break;
       }
       mesg [n] =0;
       if(write(newSockFd,mesg,n,0) != n)
       {
          errExit ("send error \n");
       }
       printf("Received and sent following message :\n%s\n",mesg);
   }
   exit(0);
}
else  /*** Parent process ***/
{
   close(newSockFd);
}
```

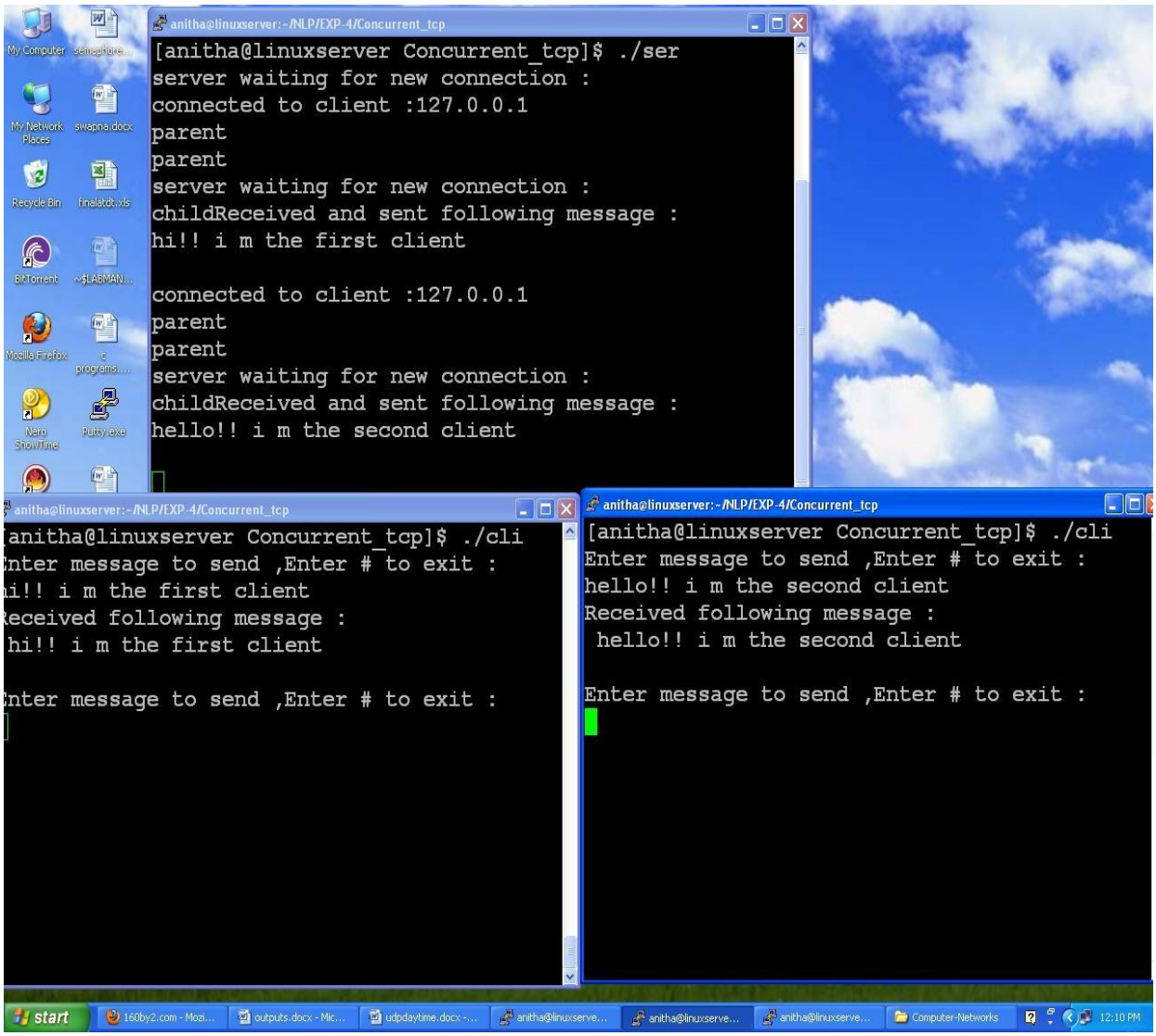
```
    }
printf("parent");
}
```

Client Program

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#define  SRV_IP_ADRS     "127.0.0.1"
#define  SRV_TCP_PORT    8888
#define  MAX_MSG         100
void errExit(char *str)
{
   puts(str);
   exit(0);
}
int main ( )
{
   int  sockFd;
   struct sockaddr_in  srvAdr;
   char txmsg[MAX_MSG];
   char rxmsg[MAX_MSG];
   int n;
   if( (sockFd =socket(AF_INET,SOCK_STREAM,0))<0)
   {
      errExit("can't open stream socket \n");
   }
```

```
memset(&srvAdr, 0,sizeof(srvAdr));
srvAdr.sin_family = AF_INET;
srvAdr.sin_addr.s_addr = inet_addr(SRV_IP_ADRS);
srvAdr.sin_port = htons(SRV_TCP_PORT);
if (connect(sockFd,(struct sockaddr *)&srvAdr,sizeof(srvAdr))<0)
{
   errExit("can't connect to server \n");
}
while(1)
{
   printf("Enter message to send ,Enter # to exit :\n");
   fgets(txmsg,MAX_MSG,stdin);
   if( txmsg[0] == '#')
   {
      break;
   }
   n = strlen(txmsg)+1;
   if ( send(sockFd,txmsg,n,0 ) != n)
   {
      errExit("send error \n");
   }
   n=recv(sockFd,rxmsg,MAX_MSG,0);
   if ( n < 0)
   {
      errExit("recv error \n");
   }
   printf("Received following message : \n %s \n",rxmsg );
}
close (sockFd);  }
```

OUTPUT:



(g) Implementation of concurrent echo server using connectionless socket system calls

Server:-

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
main(int argc,char *argv[])
{
int sockfd,rval,pid;
char buff1[20],buff2[20];
```

```
struct sockaddr_in server,client;
int len;
sockfd=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
if(sockfd==-1)
{
perror("\n SOCK_ERR\n");
exit(1);
}
server.sin_family=AF_INET;
server.sin_addr.s_addr=inet_addr("192.168.0.5");
server.sin_port=htons(3221);
rval=bind(sockfd,(struct sockaddr *)&server,sizeof(server));
if(rval!=-1)
{
pid=fork();
if(pid==0)
{
printf("\n child process executing\n");
printf("\n child process ID is:%d\n",getpid());
len=sizeof(client);
rval=recvfrom(sockfd,buff1,20,0,(struct sockaddr *)&client,&len);
if(rval==-1)
{
perror("\n RECV_ERR\n");
exit(1);
}
else
{
printf("\n received message is:%s\n",buff1);
}
rval=sendto(sockfd,buff1,sizeof(buff1),0,(struct sockaddr *)&client,sizeof(client));
if(rval!=-1)
{
printf("\n message sent successfully\n");
}
else
{ perror("\n SEND_ERR\n");
  exit(1);
}
}
else
printf("\n parent process \n");
printf("parent process ID is %d\n",getppid());
}
else
```

```
{ perror("\n BIND_ERR\n");
  exit(1);
}
}
```

Client:-

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
main(int argc,char * argv[])
{
int sockfd,rval;
char buff1[20],buff2[20];
struct sockaddr_in server,client;
int len;
sockfd=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
if(sockfd==-1)
{
perror("\n SOCK_ERR\n");
exit(1);
}
server.sin_family=AF_INET;
server.sin_addr.s_addr=inet_addr("192.168.0.5");
server.sin_port=htons(3221);
rval=bind(sockfd,(struct sockaddr *)&client,sizeof(client));
if(rval==-1)
{
perror("\n BIND_ERR\n");
exit(1);
}
printf("\n enter a message\n");
scanf("%s",buff1);
rval=sendto(sockfd,buff1,sizeof(buff1),0,(struct sockaddr *)&server,sizeof(server));
if(rval!=-1)
{
printf("\n message sent succesfully\n");
}
else
{
perror("\n SEND_ERR\n");
exit(1);
}
```

```
len=sizeof(server);
rval=recvfrom(sockfd,buff1,20,0,(struct sockaddr *)&server,&len);
if(rval==-1)

perror("\n RECV_ERR\n");
exit(1);
}
else
{
printf("\n received message is %s\n",buff1);
}
}
```

Output:-

Step 1:- cc echoservercl.c
./a.out

```
                child process executing
                child process s ID is:3012
 parent process
parent process ID is 922
received message is:hi

                message sent successfully
parent process ID is 1
```

step 2:- cc echoclientcl.c
./a.out

```
                enter a message
hi
                message sent succesfully
                received message is hi
```

8. Programming using RPC

Remote Procedure Call is a software communication protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details. RPC is used to call other processes on the remote systems like a local system. A procedure call is also sometimes known as a function call or a subroutine call.

RPC uses the client-server model. The requesting program is a client, and the service-providing

program is the server. Like a local procedure call, an RPC is
a synchronous operation requiring the requesting program to be suspended until the results of the remote procedure are returned. However, the use of lightweight processes or threads that share the same address space enables multiple RPCs to be performed concurrently.
The interface definition language (IDL) -- the specification language used to describe a software component's application programming interface (API) -- is commonly used in Remote Procedure Call software. In this case, IDL provides a bridge between the machines at either end of the link that may be using different operating systems (OSes) and computer languages.

How does RPC work?
When a remote procedure call is invoked, the calling environment is suspended, the procedure parameters are transferred across the network to the environment where the procedure is to execute, and the procedure is then executed in that environment.
When the procedure finishes, the results are transferred back to the calling environment, where execution resumes as if returning from a regular procedure call.
During an RPC, the following steps take place:
1. The client calls the client stub. The call is a local procedure call with parameters pushed onto the stack in the normal way.
2. The client stub packs the procedure parameters into a message and makes a system call to send the message. The packing of the procedure parameters is called marshalling.
3. The client's local OS sends the message from the client machine to the remote server machine.
4. The server OS passes the incoming packets to the server stub.
5. The server stub unpacks the parameters -- called unmarshalling -- from the message.
6. When the server procedure is finished, it returns to the server stub, which marshals the return values into a message. The server stub then hands the message to the transport layer.
7. The transport layer sends the resulting message back to the client transport layer, which hands the message back to the client stub.
8. The client stub unmarshalls the return parameters, and execution returns to the caller.

ALGORITHM:
Server:
Step 1: Start the program
Step 2: Create a socket with address family AF_INET type SOCK_STREAM and default protocol.
Step 3: Initialize a socket and set its attributes.
Step 4: Bind the server to the socket using bind function.
Step 5: wait for the client request, on request establish a connection using accept function.
Step 6: Read the number from the client by using read method Step 7: Display the no.
Step 8: add the digits of a given number.
Step 9: send the result to the client by using write method Step 10: stop the program.
Client:
Step 1: start.
Step 2: create a socket with address family.
Step 3: Initialize the socket and set its attributes set the required port no. Step 4: Type AF_INET

socket with default protocol.
Step 5: Connect to server using connect function to initiate the request. Step 6: send a no to the server using write method.
Step 7: receive the result using read method.
Step 8: stop

Algorithm for installing RPC extension

Step 1: sudo apt-get install rpcbind
step 2: rpc/info
(This confirms rpc bind is installed in your system)
Step 3: Create a new folder'RPC'.
Step 4: Creating a new file "ADD.X".
add.x
struct numbers{ // defining structure to identify which arguements are passing int a;
int b;
}; //they will be combined into one structure called "numbers"
program ADD_PROG{ //this is written as per the syntax of the x file.
version ADD_VERS{//THE ADD_PROG IS A PROGRAM THAT TAKES NUMBERS AND GIVES AN INTEGER RESULT
int add(numbers)=1;
}=1
}=0x23451111;
step 5: save file as add.x Vim code:
cd RPC/
rpcgen -a -C add.x //a indicates creation of file and c indicates creation of c structure file
make -f Makefile.add sudo ./add_server
(open a new putty system) sudo ./add_client localhost

server.c
#include "add.h"
int* add_1_svc(numbers *argp, struct svc/_req *rqstp)
{
struct int result;
printf("add(%d, %d) is called\n",argp->a,argp->b); result = argp->a + argp->b
return &result;
}
client.c
void add_prog_1(char *host, int x, int y)
{
CLIENT *clint; int *result;
numbers add_1_arg;
#ifndef DEBUG
clnt= clnt_create (host,ADD_PROG, ADD_VERS,"udp"); if(clnt==NULL)
{

```
clnt_pcreateerror (host); exit(1);
}
#endif //debug add_1_arg.a=x; add_1_arg.b=y;
result_1 = add_1(&add_1_arg, clnt); if(result_1== (int *) NULL){ clnt_perror (clnt, "call
failed");
}
else{
printf("result:%d\n",*result_1);
}
#ifndef DEBUG clnt_destroy (clnt); #endifn//debug
}
int main (int argc, char *argv[])
{
char *host; if(argc < 4)
{
printf("usage : %s server_host \n",argv[0]); exit(1);
}
host = argv[1];
add_prog_1 (host, atoi(argv[2]), atoi (argv[3])); exit(0);
}
```

step 6: re running the same program after entering the code vim
make -f Makefile.add sudo ./add_server
sudo ./add_client localhost
sudo ./add_client localhost 10 20
Output:
Localhost@localhost-virtualbox:-/RPC$ sudo ./add_server Add(10,20) is called
Add(30,50) is called
Localhost@localhost-virtualbox:-/RPC$ sudo ./add_client localhost Usage: ./add_client server_host NUMBER NUMBER Localhost@localhost-virtualbox:-/RPC$ sudo ./add_client localhost 10 20 Result:30
Localhost@localhost-virtualbox:-/RPC$ sudo ./add_client localhost 30 50 Result:80
Localhost@localhost-virtualbox:-/RPC$