RAJALAKSHMI ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

CS23532 - COMPUTER NETWORKS

LAB MANUAL

Name	: **ABINAYA M**
University Roll No.	: **2116231001005**
Year / Branch	: **3rd Year / INFORMATION TECHNOLOGY**
Semester	: **5th Semester**
Academic Year	: **2025 - 2026**

RAJALAKSHMI ENGINEERING COLLEGE

BONAFIDE CERTIFICATE

NAME ABINAYA M

ACADEMIC YEAR 2025 - 2026 SEMESTER V BRANCH IT

UNIVERSITY REGISTER No, 2116231001005

Certified that this is the bonafide record of work done by the above student in the

COMPUTER NETWORKS [CS23532] Laboratory during the year 2025 - 2026

Signature of Faculty - in - Charge

Submitted for the Practical Examination held on.....

Internal Examiner External Examiner

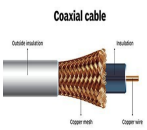
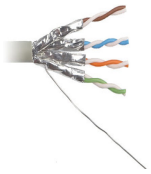
INDEX

Ex. Nos. 1		STUDY OF DIFFERENT TYPES OF NETWORK CABLES
Date :	08-07-25	

AIM: To Perform basic study on various network cables.

STUDY OF CABLES:

Cable Type	Pros	Cons	Usage	Bandwidth	Speed	Image
Cat5 (Category 5)	- Low cost- Easy to install- Widely used	- Lower speeds- Shorter maximum distance (100 meters)	- Home networks- Small office setups	100 Mbps	10/100 Mbps (Fast Ethernet)	 Cat5
Cat5e (Category 5e)	- Enhanced version of Cat5- Reduced crosstalk- More reliable	- Limited to 100 meters- Lower speeds than newer cables	- Ethernet connections- Home/office networks	1 Gbps	1 Gbps (Gigabit Ethernet)	 Cat5e
Cat6 (Category 6)	- Higher bandwidth than Cat5e- Reduced crosstalk- More reliable	- More expensive than Cat5e- Stiff and difficult to work with	- High-speed networking- Large office buildings	10 Gbps (up to 55 meters)	1-10 Gbps (Gigabit Ethernet / 10 Gigabit Ethernet)	 Cat6
Cat6a (Category 6a)	- Supports higher bandwidths- Longer maximum distance	- Expensive- Bulky and hard to install	- Data centers- High-performance networking	10 Gbps (up to 100 meters)	10 Gbps (10 Gigabit Ethernet)	

Cat7 (Category 7)	- Shielded for high noise resistance- High-speed performance	- Expensive- Thick, less flexible cables	- Data centers- High-speed environments	10 Gbps (up to 100 meters)	10 Gbps (10 Gigabit Ethernet)	
Cat8 (Category 8)	- Highest speed & performance- Supports high-frequency signals	- Expensive- Shorter range- Heavy and rigid	- Data centers- Server rooms- High-performance applications	25-40 Gbps (up to 30 meters)	25-40 Gbps (High-Speed Data Centers)	
Coaxial Cable	- Durable- Less susceptible to electromagnetic interference	- Low bandwidth compared to twisted pair cables	- Cable TV- Broadband internet connections	10 Mbps to 10 Gbps	10 Mbps to 10 Gbps (depending on use)	
Fiber Optic	- Extremely high bandwidth- Very long distance- Immune to EMI	- Expensive- Fragile- Requires specialized installation	-Long-distance networking- High-speed data transfer	10 Gbps to 100 Gbps (or higher)	10 Gbps to 100 Gbps (and beyond)	
Twisted Pair (Unshielded)	-Cost-effective- Lightweight- Easy to install	- Prone to interference without shielding	- Telephone lines- Home and office networking	100 Mbps	10/100 Mbps (Fast Ethernet)	
Twisted Pair (Shielded)	- Higher resistance to interference- Better data integrity	- More expensive- Less flexible and harder to install	- Industrial settings- Areas with high interference	100 Mbps	10/100 Mbps (Fast Ethernet)	

RESULT: Thus the study on various network cables was conducted successfully

Ex. Nos. 2		BASIC NETWORKING COMMANDS IN LINUX AND WINDOWS OPERATING SYSTEM
Date :	15-07-25	

AIM: To execute various networking commands in windows and linux

WINDOWS COMMANDS:

1.ipconfig:

The IPCONFIG network command provides a comprehensive view of information regarding the IP address configuration of the device we are currently working on.

OUTPUT:

C:\Users\Lenovo>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet 3:

Media State . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Wireless LAN adapter Local Area Connection* 13:

Media State . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Wireless LAN adapter Local Area Connection* 14:

Media State . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Wireless LAN adapter Wi-Fi 3:

Connection-specific DNS Suffix . :

Link-local IPv6 Address . . . . . : fe80::90d1:aa4b:ced6:d82d%17

IPv4 Address. . . . . : 172.16.76.93

Subnet Mask . . . . . : 255.255.248.0

Default Gateway . . . . . : 172.16.72.1

NOTE:

- IPConfig/all - Provides primary output with additional information about network adapters.
- IPConfig/renew - Used to renew the system's IP address.
- IPConfig/release - Removes the system's current IP address.

2.nslookup

The NSLOOKUP command is used to troubleshoot network connectivity issues in the system. Using the nslookup command, we can access the information related to our system's DNS server, i.e., domain name and IP address.

OUTPUT:

C:\Users\Lenovo>nslookup

Default Server: UnKnown

Address: 172.16.72.1

3.hostname

The HOSTNAME command displays the hostname of the system. The hostname command is much easier to use than going into the system settings to search for it.

OUTPUT:

C:\Users\Lenovo>hostname

DESKTOP-C01BH7D

4.ping

The Ping command is one of the most widely used commands in the prompt tool, as it allows the user to check the connectivity of our system to another host.

OUTPUT:

C:\Users\Lenovo>ping www.google.com

Pinging www.google.com [142.250.195.228] with 32 bytes of data:
Reply from 142.250.195.228: bytes=32 time=8ms TTL=119
Reply from 142.250.195.228: bytes=32 time=8ms TTL=119
Reply from 142.250.195.228: bytes=32 time=7ms TTL=119
Reply from 142.250.195.228: bytes=32 time=13ms TTL=119

Ping statistics for 142.250.195.228:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 7ms, Maximum = 13ms, Average = 9ms

5.Tracert

The TRACERT command is used to trace the route during the transmission of the data packet over to the destination host and also provides us with the “hop” count during transmission.

OUTPUT:

C:\Users\Lenovo>tracert www.google.com

Tracing route to www.google.com [142.250.195.228]
over a maximum of 30 hops:

1	4 ms	1 ms	2 ms	172.16.72.1
2	16 ms	4 ms	5 ms	115.245.95.249
3	*	*	*	Request timed out.
4	10 ms	9 ms	7 ms	172.16.12.64
5	15 ms	5 ms	28 ms	172.16.12.64
6	15 ms	6 ms	6 ms	72.14.217.252
7	9 ms	7 ms	7 ms	216.239.43.131
8	7 ms	5 ms	5 ms	142.250.224.7
9	13 ms	7 ms	9 ms	maa03s43-in-f4.1e100.net [142.250.195.228]

Trace complete.

6.netstat

The Netstat command as the name suggests displays an overview of all the network connections in the device. The table shows detail about the connection protocol, address, and the current state of the network.

OUTPUT:

C:\Users\Lenovo>netstat

Active Connections

Proto	Local Address	Foreign Address	State
TCP	127.0.0.1:49675	DESKTOP-C01BH7D:49676	ESTABLISHED
TCP	127.0.0.1:49676	DESKTOP-C01BH7D:49675	ESTABLISHED
TCP	127.0.0.1:49677	DESKTOP-C01BH7D:49678	ESTABLISHED
TCP	127.0.0.1:49678	DESKTOP-C01BH7D:49677	ESTABLISHED
TCP	127.0.0.1:49681	DESKTOP-C01BH7D:49682	ESTABLISHED
TCP	127.0.0.1:49682	DESKTOP-C01BH7D:49681	ESTABLISHED
TCP	127.0.0.1:49683	DESKTOP-C01BH7D:49684	ESTABLISHED
TCP	127.0.0.1:49684	DESKTOP-C01BH7D:49683	ESTABLISHED
TCP	127.0.0.1:49695	DESKTOP-C01BH7D:49696	ESTABLISHED
TCP	127.0.0.1:49696	DESKTOP-C01BH7D:49695	ESTABLISHED
TCP	172.16.76.93:49408	4.213.25.240:https	ESTABLISHED
TCP	172.16.76.93:50338	pnmaaa-aq-in-f5:https	ESTABLISHED

7.arp

The ARP command is used to access the mapping structure of IP addresses to the MAC address. This provides us with a better understanding of the transmission of packets in the network channel.

OUTPUT:

C:\Users\Lenovo>arp

Displays and modifies the IP-to-Physical address translation tables used by address resolution protocol (ARP).

ARP -s inet_addr eth_addr [if_addr]
ARP -d inet_addr [if_addr]
ARP -a [inet_addr] [-N if_addr] [-v]

-a	Displays current ARP entries by interrogating the current protocol data. If inet_addr is specified, the IP and Physical addresses for only the specified computer are displayed. If more than one network interface uses ARP, entries for each ARP table are displayed.
-g	Same as -a.
-v	Displays current ARP entries in verbose mode. All invalid entries and entries on the loop-back interface will be shown.
inet_addr	Specifies an internet address.
-N if_addr	Displays the ARP entries for the network interface specified by if_addr.
-d	Deletes the host specified by inet_addr. inet_addr may be wildcarded with * to delete all hosts.

-s Adds the host and associates the Internet address inet_addr with the Physical address eth_addr. The Physical address is given as 6 hexadecimal bytes separated by hyphens. The entry is permanent.

eth_addr Specifies a physical address.

if_addr If present, this specifies the Internet address of the interface whose address translation table should be modified. If not present, the first applicable interface will be used.

Example:

> arp -s 157.55.85.212 00-aa-00-62-c6-09 .... Adds a static entry.

> arp -a .... Displays the arp table.

8.systeminfo

Using the SYSTEMINFO command, we can access the system's hardware and software details, such as processor data, booting data, Windows version, etc.

OUTPUT:

C:\Users\Lenovo>systeminfo
Host Name: DESKTOP-C01BH7D
OS Name: Microsoft Windows 11 Pro
OS Version: 10.0.21996 N/A Build 21996
OS Manufacturer: Microsoft Corporation
OS Configuration: Standalone Workstation
OS Build Type: Multiprocessor Free
Registered Owner: Lenovo
Registered Organization:
Product ID: 00331-10000-00001-AA753
Original Install Date: 17-02-2024, 07:14:20 AM
System Boot Time: 15-07-2025, 08:21:33 AM
System Manufacturer: Dell Inc.
System Model: OptiPlex Tower 7010
System Type: x64-based PC
Processor(s): 1 Processor(s) Installed.
[01]: Intel64 Family 6 Model 151 Stepping 5 GenuineIntel ~3000 Mhz
BIOS Version: Dell Inc. 1.9.0, 02-10-2023
Windows Directory: C:\Windows
System Directory: C:\Windows\system32
Boot Device: \Device\HarddiskVolume1
System Locale: en-us;English (United States)
Input Locale: 00004009
Time Zone: (UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi
Total Physical Memory: 16,073 MB
Available Physical Memory: 9,568 MB
Virtual Memory: Max Size: 18,505 MB
Virtual Memory: Available: 11,176 MB
Virtual Memory: In Use: 7,329 MB

Page File Location(s): C:\pagefile.sys
Domain: WORKGROUP
Logon Server: \\DESKTOP-C01BH7D
Hotfix(s): N/A
Network Card(s): 3 NIC(s) Installed.
[01]: Realtek RTL8852BE WiFi 6 802.11ax PCIe Adapter
Connection Name: Wi-Fi 3
DHCP Enabled: No
IP address(es)
[01]: 172.16.76.93
[02]: fe80::90d1:aa4b:ced6:d82d
[02]: Microsoft Wi-Fi Direct Virtual Adapter
Connection Name: Local Area Connection* 14
Status: Media disconnected
[03]: Intel(R) Ethernet Connection (17) I219-LM
Connection Name: Ethernet 3
Status: Media disconnected

Hyper-V Requirements: A hypervisor has been detected. Features required for Hyper-V will not be displayed.

9.getmac

getmac is a command-line utility primarily used on Windows operating systems to display the Media Access Control (MAC) addresses of all network adapters in a computer.

OUTPUT:

C:\Users\Lenovo>getmac

Physical Address Transport Name
==========
==========
4C-82-A9-77-FF-DD \Device\Tcpip_{F3860600-35CB-4270-B177-0395BA041C16}
42-82-A9-77-FF-DD Media disconnected
20-88-10-86-BC-F4 Media disconnected

10.pathping

pathping is an indispensable tool for anyone troubleshooting network performance issues on Windows, as it provides a deeper insight into packet flow and potential bottlenecks than its simpler counterparts.

LINUX COMMANDS

1. ip command

Used to show and manipulate routing, devices, and IP addresses in Linux networking.

OUTPUT:

```
fr03@fedora:~$ ip
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
       ip [ -force ] -batch filename
where  OBJECT := { address | addrlabel | fou | help | ila | ioam | l2tp | link |
                   macsec | maddress | monitor | mptcp | mroute | mrule |
                   neighbor | neighbour | netconf | netns | nexthop | ntable |
                   ntbl | route | rule | sr | stats | tap | tcpmetrics |
                   token | tunnel | tuntap | vrf | xfrm }
       OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |
                   -h[uman-readable] | -iec | -j[son] | -p[retty] |
                   -f[amily] { inet | inet6 | mpls | bridge | link } |
                   -4 | -6 | -M | -B | -0 |
                   -l[oops] { maximum-addr-flush-attempts } | -echo | -br[ief] |
                   -o[neline] | -t[imestamp] | -ts[hort] | -b[atch] [filename] |
                   -rc[vbuf] [size] | -n[etns] name | -N[umeric] | -a[ll] |
                   -c[olor]}
```

2.ip -V command

Displays the version information of the ip command tool with verbose output.

OUTPUT:

```
kfr03@fedora:~$ ip -V
ip utility, iproute2-6.10.0, libbpf 1.4.7
kfr03@fedora:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
1000
       link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
       inet 127.0.0.1/8 scope host lo
```

valid_lft forever preferred_lft forever

inet6 ::1/128 scope host noprefixroute

valid_lft forever preferred_lft forever

2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000

link/ether 20:88:10:86:75:c9 brd ff:ff:ff:ff:ff:ff

3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether e6:82:dd:5f:b2:f1 brd ff:ff:ff:ff:ff:ff permaddr 4c:82:a9:78:01:41

inet 172.16.76.82/21 brd 172.16.79.255 scope global noprefixroute wlp2s0

valid_lft forever preferred_lft forever

inet6 fe80::4f1:c483:3baa:f12f/64 scope link noprefixroute

valid_lft forever preferred_lft forever

3 .ip addr

Displays all IP addresses and network interface details on the system.

OUTPUT:

kfr03@fedora:~$ ip addr

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid_lft forever preferred_lft forever

inet6 ::1/128 scope host noprefixroute

valid_lft forever preferred_lft forever

2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000

link/ether 20:88:10:86:75:c9 brd ff:ff:ff:ff:ff:ff

3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether e6:82:dd:5f:b2:f1 brd ff:ff:ff:ff:ff:ff permaddr 4c:82:a9:78:01:41

inet 172.16.76.82/21 brd 172.16.79.255 scope global noprefixroute wlp2s0

valid_lft forever preferred_lft forever

inet6 fe80::4f1:c483:3baa:f12f/64 scope link noprefixroute

valid_lft forever preferred_lft forever

4.ip addr show

Shows detailed information about all network interfaces and their assigned IP addresses.

OUTPUT:

kfr03@fedora:~$ ip addr show

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid_lft forever preferred_lft forever

inet6 ::1/128 scope host noprefixroute

valid_lft forever preferred_lft forever

2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000

link/ether 20:88:10:86:75:c9 brd ff:ff:ff:ff:ff:ff

3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether e6:82:dd:5f:b2:f1 brd ff:ff:ff:ff:ff:ff permaddr 4c:82:a9:78:01:41

inet 172.16.76.82/21 brd 172.16.79.255 scope global noprefixroute wlp2s0

valid_lft forever preferred_lft forever

inet6 fe80::4f1:c483:3baa:f12f/64 scope link noprefixroute

valid_lft forever preferred_lft forever

5.ip link

Displays and manages network interfaces (links) on the system.

OUTPUT:

kfr03@fedora:~$ ip link

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT group default qlen 1000

link/ether 20:88:10:86:75:c9 brd ff:ff:ff:ff:ff:ff

3: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DORMANT group default qlen 1000

link/ether e6:82:dd:5f:b2:f1 brd ff:ff:ff:ff:ff:ff permaddr 4c:82:a9:78:01:41

6.ip route

Displays the kernel's IP routing table (shows how packets are routed).

OUTPUT:

kfr03@fedora:~$ ip route

default via 172.16.72.1 dev wlp2s0 proto static metric 600

172.16.72.0/21 dev wlp2s0 proto kernel scope link src 172.16.76.82 metric 600

kfr03@fedora:~$ ip route show

default via 172.16.72.1 dev wlp2s0 proto static metric 600

7.ifconfig

Legacy Linux command used to view and configure network interfaces (now replaced by ip command).

OUTPUT:

kfr03@fedora:~$ ifconfig

enp0s31f6: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500

ether 20:88:10:86:75:c9 txqueuelen 1000 (Ethernet)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

device interrupt 19 memory 0x70600000-70620000

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 794 bytes 76920 (75.1 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 794 bytes 76920 (75.1 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 172.16.76.82 netmask 255.255.248.0 broadcast 172.16.79.255
inet6 fe80::4f1:c483:3baa:f12f prefixlen 64 scopeid 0x20<link>
ether e6:82:dd:5f:b2:f1 txqueuelen 1000 (Ethernet)
RX packets 267468 bytes 321463323 (306.5 MiB)
RX errors 0 dropped 1195 overruns 0 frame 0
TX packets 51262 bytes 13161940 (12.5 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

8.dig

Performs DNS lookups and displays detailed DNS query results for a domain.

OUTPUT:

kfr03@fedora:~$ dig

; <<>> DiG 9.18.33 <<>>
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50538
;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;. IN NS

;; ANSWER SECTION:

```
.       87203   IN      NS      g.root-servers.net.
.       87203   IN      NS      m.root-servers.net.
.       87203   IN      NS      i.root-servers.net.
.       87203   IN      NS      h.root-servers.net.
.       87203   IN      NS      b.root-servers.net.
.       87203   IN      NS      f.root-servers.net.
.       87203   IN      NS      l.root-servers.net.
.       87203   IN      NS      c.root-servers.net.
.       87203   IN      NS      e.root-servers.net.
.       87203   IN      NS      a.root-servers.net.
.       87203   IN      NS      d.root-servers.net.
.       87203   IN      NS      k.root-servers.net.
.       87203   IN      NS      j.root-servers.net.

;; Query time: 256 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Mon Jul 21 23:02:18 EDT 2025
;; MSG SIZE  rcvd: 239

kfr03@fedora:~$ dig google.com

; <<>> DiG 9.18.33 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43012
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;google.com.                    IN      A


;; ANSWER SECTION:
```

google.com. 35 IN A 142.250.77.142

;; Query time: 7 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Mon Jul 21 23:02:28 EDT 2025
;; MSG SIZE rcvd: 55

9.nslookup

Queries DNS to obtain domain name or IP address mapping information.

OUTPUT:

kfr03@fedora:~$ nslookup google.com
Server: 127.0.0.53
Address: 127.0.0.53#53

Non-authoritative answer:
Name: google.com
Address: 142.250.77.142
Name: google.com
Address: 2404:6800:4007:80f::200e

10.netstat

Displays network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.

OUTPUT:

```
kfr03@fedora:~$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address               Foreign Address         State
tcp        0      0 fedora:39720               203.137.36.34.bc.:https ESTABLISHED
tcp        0      0 fedora:43930               proxy14.fedorapro:https TIME_WAIT
tcp        0      0 fedora:43104               bkk03s01-in-f3.1e:https ESTABLISHED
tcp        0      0 fedora:46004               202.152.107.34.bc:https ESTABLISHED
tcp        0      0 fedora:58714               209.100.149.34.bc:https ESTABLISHED
tcp        0      0 fedora:33580               191.144.160.34.bc:https ESTABLISHED
tcp        0      0 fedora:49604               pnmaaa-aq-in-f10.:https ESTABLISHED
tcp        0      0 fedora:57424               172.66.168.9:https      TIME_WAIT
kfr03@fedora:~$ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address               Foreign Address         State
tcp        0      0 0.0.0.0:27500              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:llmnr              0.0.0.0:*               LISTEN
tcp        0      0 _localdnsproxy:domain      0.0.0.0:*               LISTEN
tcp        0      0 localhost:ipp              0.0.0.0:*               LISTEN
tcp        0      0 _localdnsstub:domain       0.0.0.0:*               LISTEN
tcp        0      0 fedora:39720               203.137.36.34.bc.:https ESTABLISHED
tcp        0      0 fedora:43104               lcmaaa-aq-in-f3.1:https ESTABLISHED
tcp        0      0 fedora:46004               202.152.107.34.bc:https ESTABLISHED
tcp        0      0 fedora:58714               209.100.149.34.bc:https ESTABLISHED
tcp        0      0 fedora:33580               191.144.160.34.bc:https ESTABLISHED
tcp        0      0 fedora:49604               pnmaaa-aq-in-f10.:https ESTABLISHED
tcp        0      0 fedora:41356               mirror.twds.com.t:https ESTABLISHED
tcp        0      0 fedora:38644               bkk02s02-in-f14.1:https ESTABLISHED
```

10.traceroute

Shows the path that packets take to reach a network host, revealing each hop along the route.

OUTPUT:

```
kfr03@fedora:~$ traceroute google.com
traceroute to google.com (142.250.77.142), 30 hops max, 60 byte packets
 1  _gateway (172.16.72.1)  1.680 ms  1.632 ms  1.614 ms
 2  static-41.229.249.49-tataidc.co.in (49.249.229.41)  5.058 ms  4.269 ms  5.028 ms
 3  142.250.171.162 (142.250.171.162)  167.350 ms  166.819 ms  169.162 ms
 4  * * *
 5  142.251.55.30 (142.251.55.30)  169.503 ms 216.239.54.196 (216.239.54.196)  170.524 ms
142.251.60.186 (142.251.60.186)  178.759 ms
 6  142.251.55.63 (142.251.55.63)  174.789 ms *^C
kfr03@fedora:~$ tracepath google.com
 1?: [LOCALHOST]                      pmtu 1500
 1:  _gateway                              1.807ms
 1:  _gateway                              1.945ms
 2:  _gateway                              1.883ms pmtu 1460
 2:  static-41.229.249.49-tataidc.co.in          4.591ms
 3:  142.250.171.162                             174.527ms asymm  7


kfr03@fedora:~$ host google.com
google.com has address 142.250.77.142
google.com has IPv6 address 2404:6800:4007:80f::200e
google.com mail is handled by 10 smtp.google.com.
```

11.hostname

Displays or sets the system's hostname (network name of the machine).

OUTPUT:

kfr03@fedora:~$ hostname

fedora

RESULT: Thus,various networking commands on windows and linux were executed successfully.

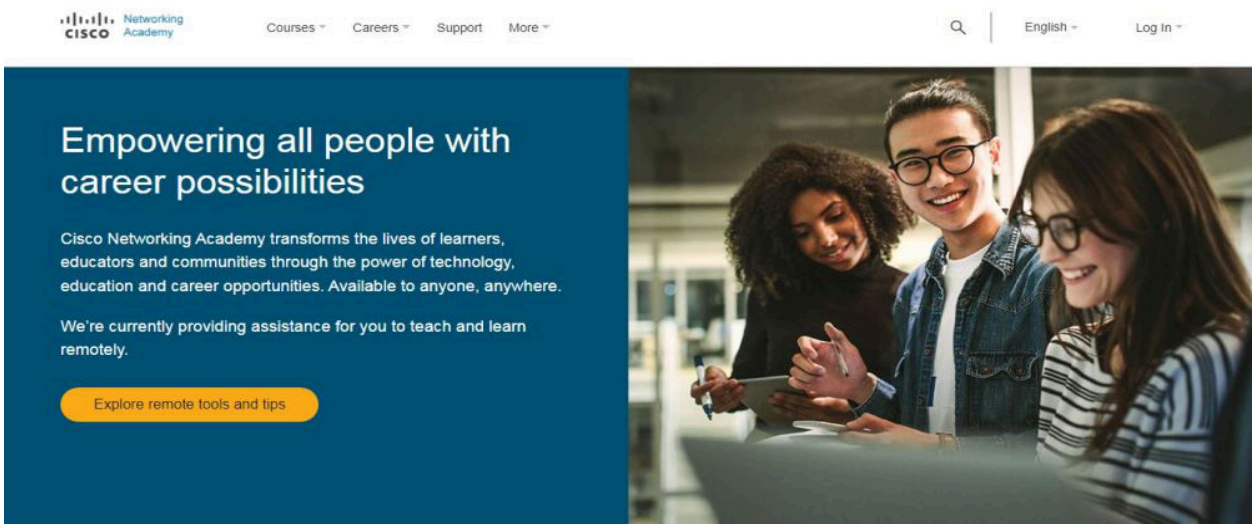
Ex. Nos. 3		CISCO PACKET TRACER INSTALLATION
Date :	22-07-25	

AIM: To install CISCO PACKET TRACER.

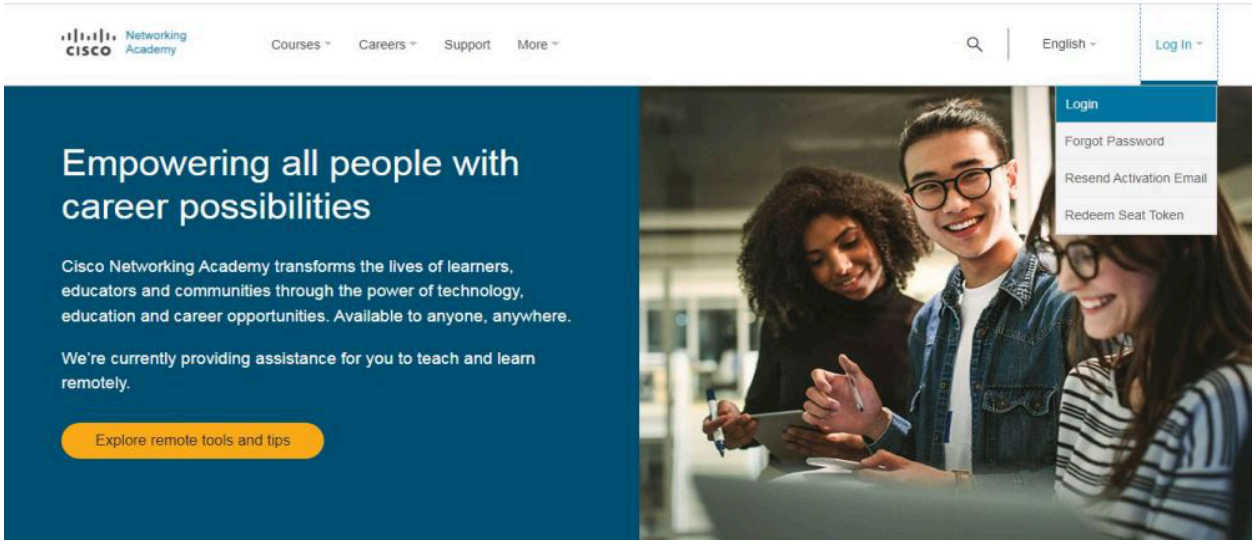
PROCEDURE:

Steps to install Packet Tracer on Windows:

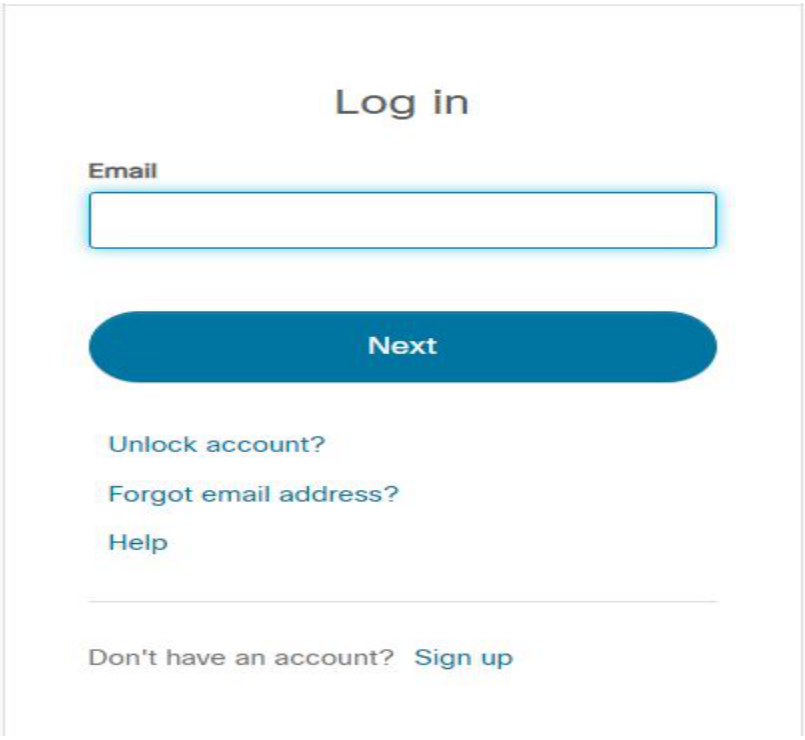
Step 1: Visit the official website of Netacad using any web browser.



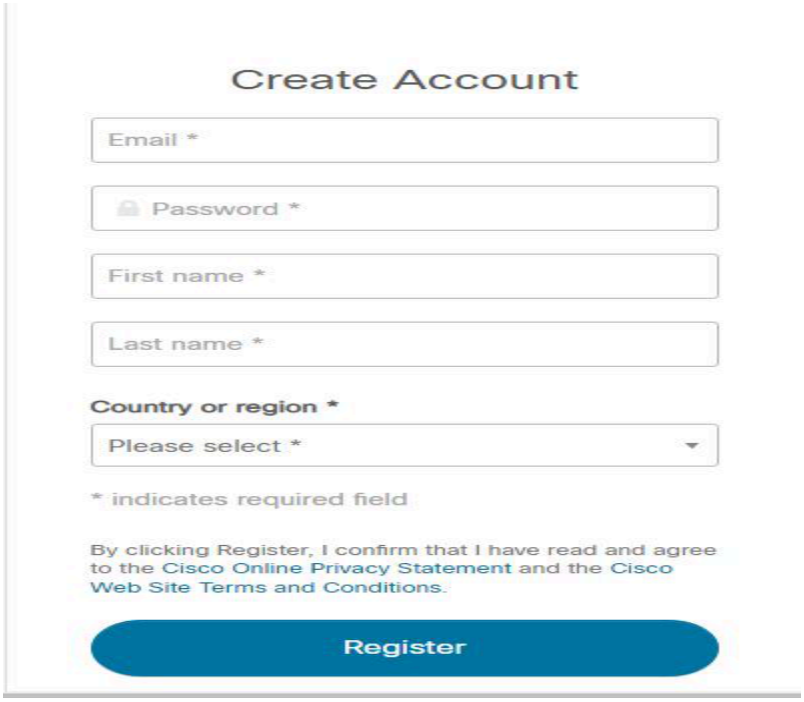
Step 2: Press the login button and select log In option.



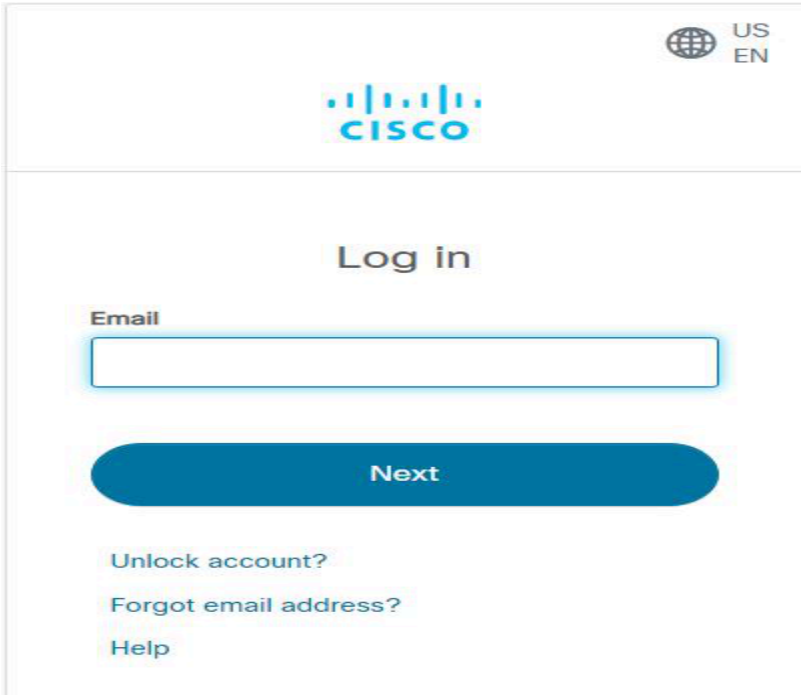
Step 3: Next screen will appear, click on the sign-up option.



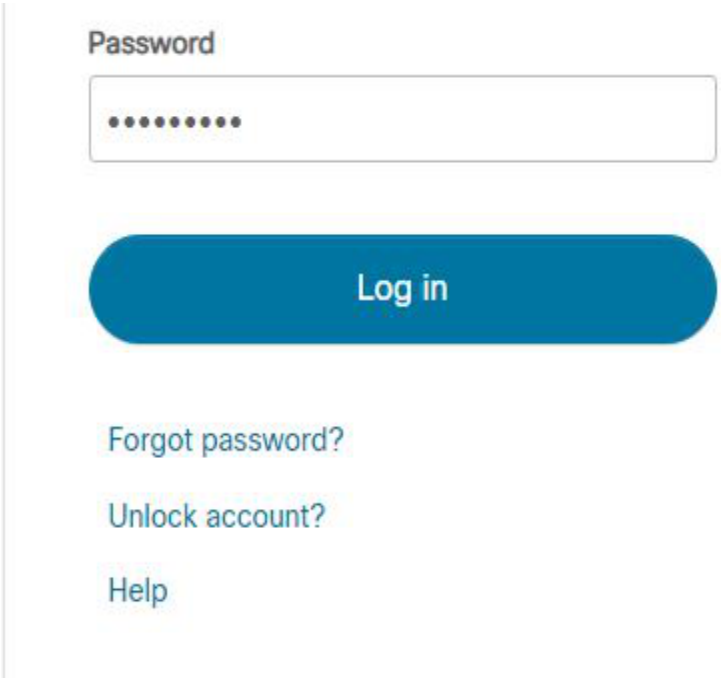
Step 4: Next screen will appear and will ask for email and password and other simple details, fill them and click on Register.



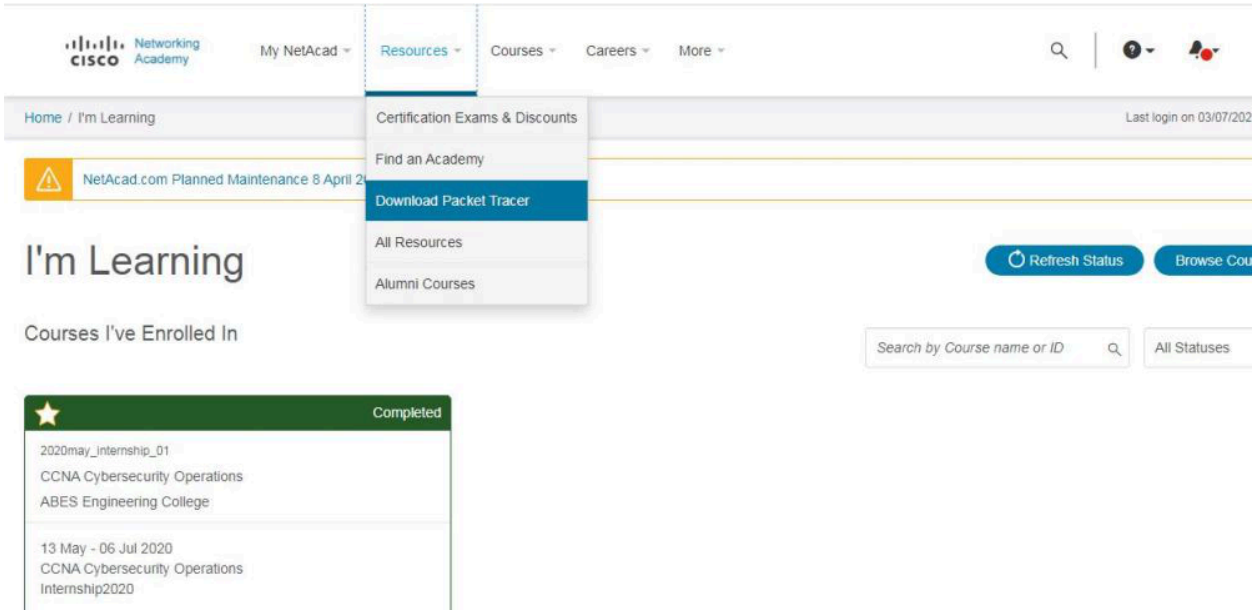
Step 5: Now the login screen appears again so fill in the Email id.



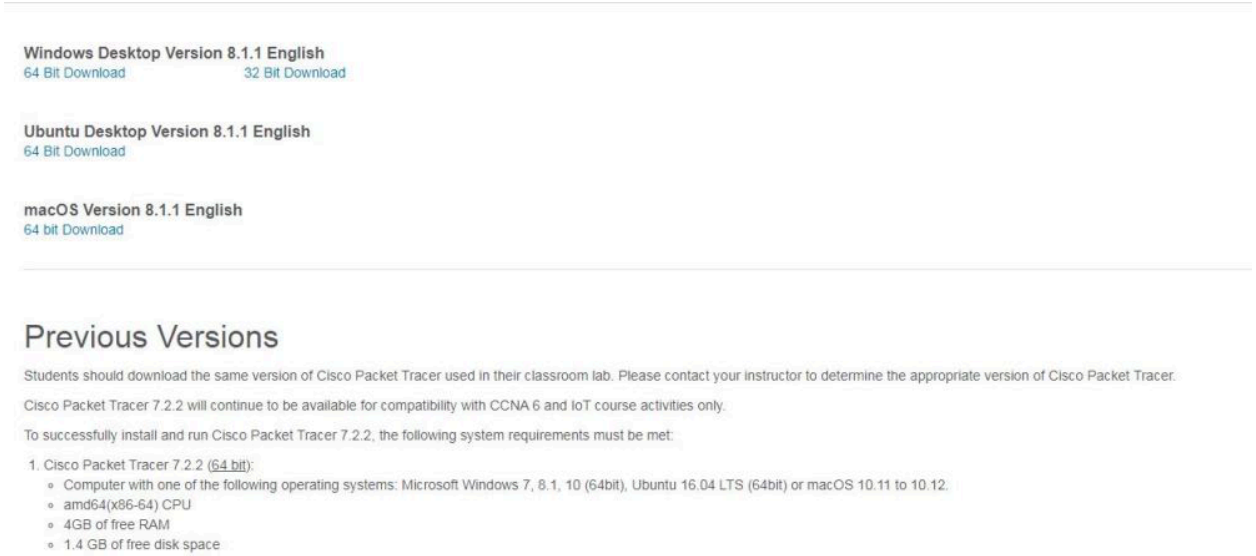
Step 6: On the next screen enter the password and press the Login button.



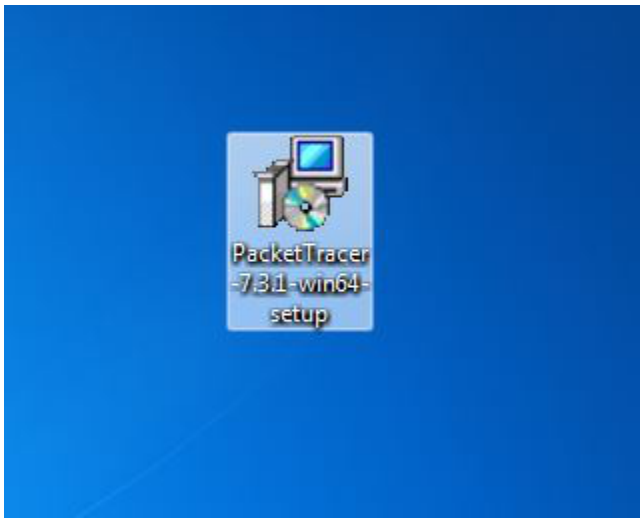
Step 7: Dashboard will initialize, now click on Resources and choose Download Packet Tracer Option.



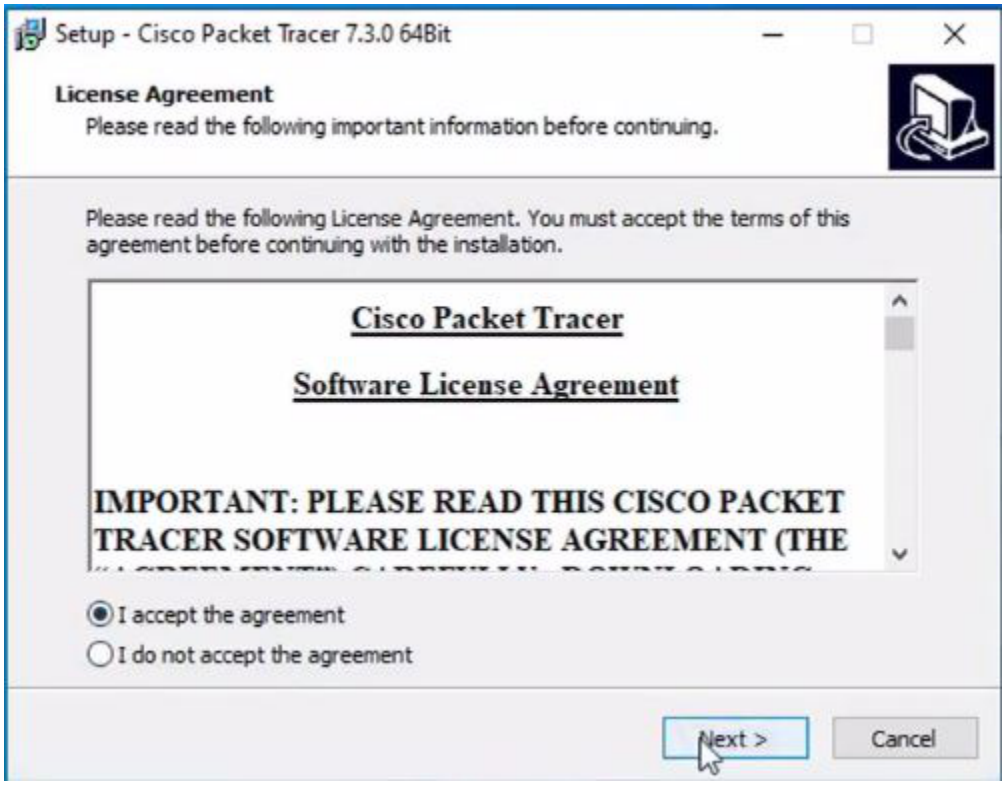
Step 8: On the next web page choose the operating system to download the packet tracer. Downloading will start automatically.



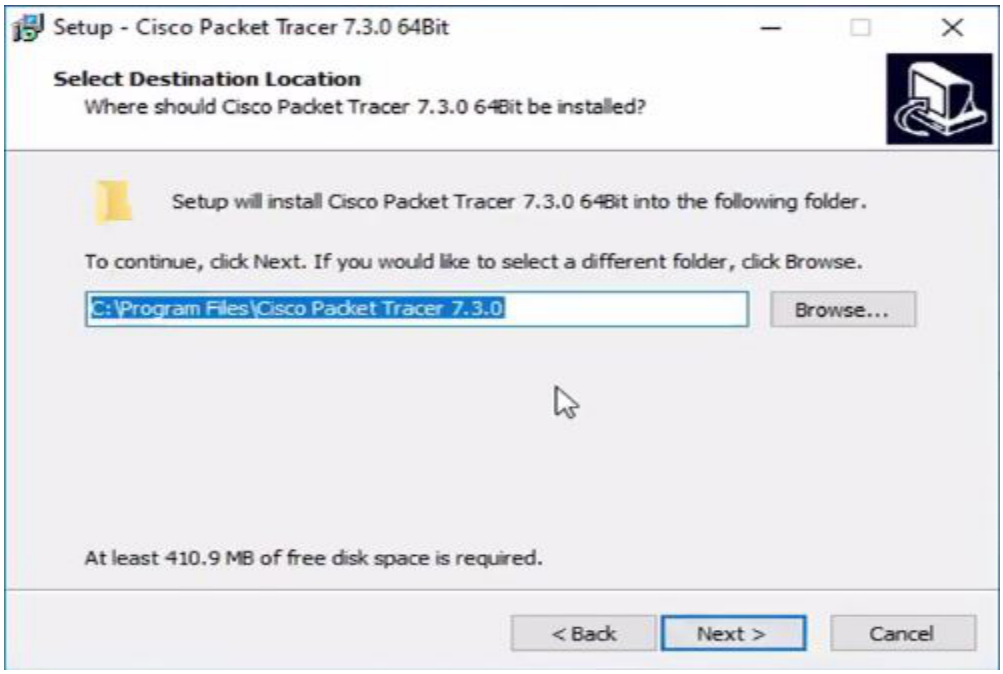
Step 9: Check for the executable file in your system and run it.



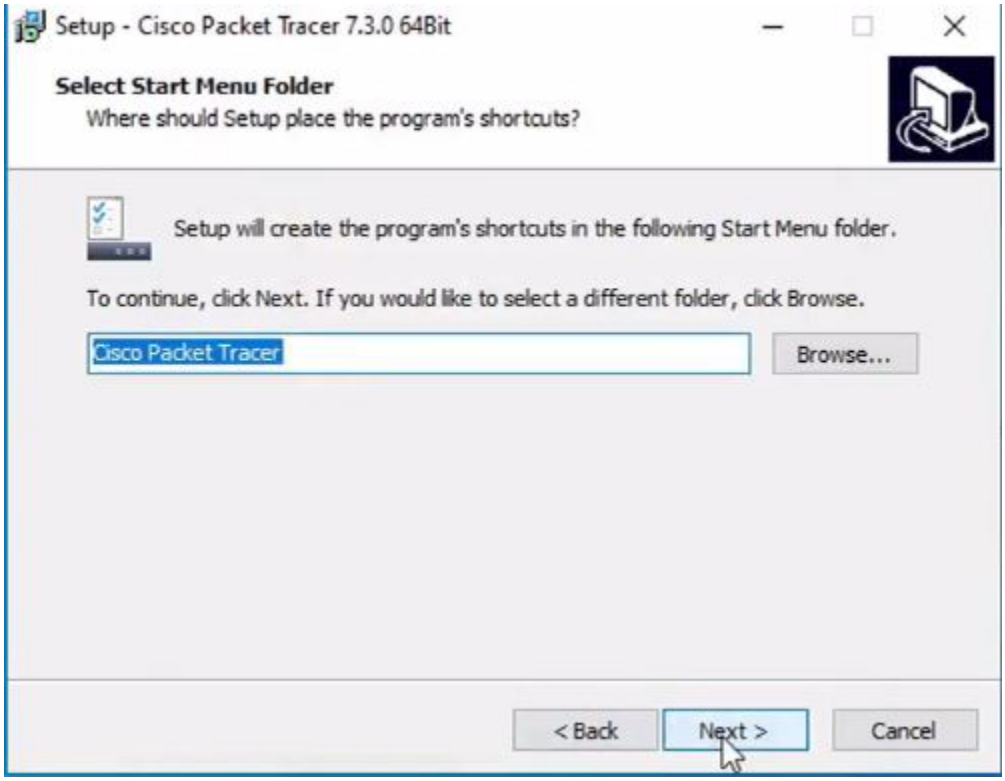
Step 10: Next screen is of License Agreement so Click on **I accept** the license.



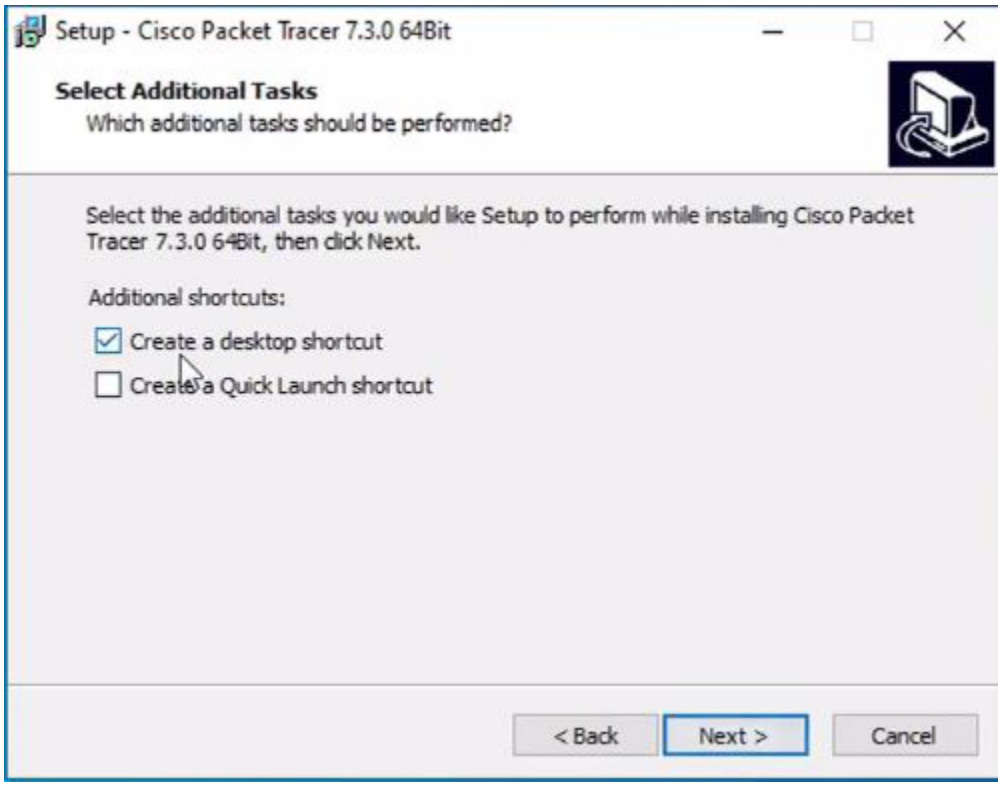
Step 11: Choose the installing location which has sufficient space.



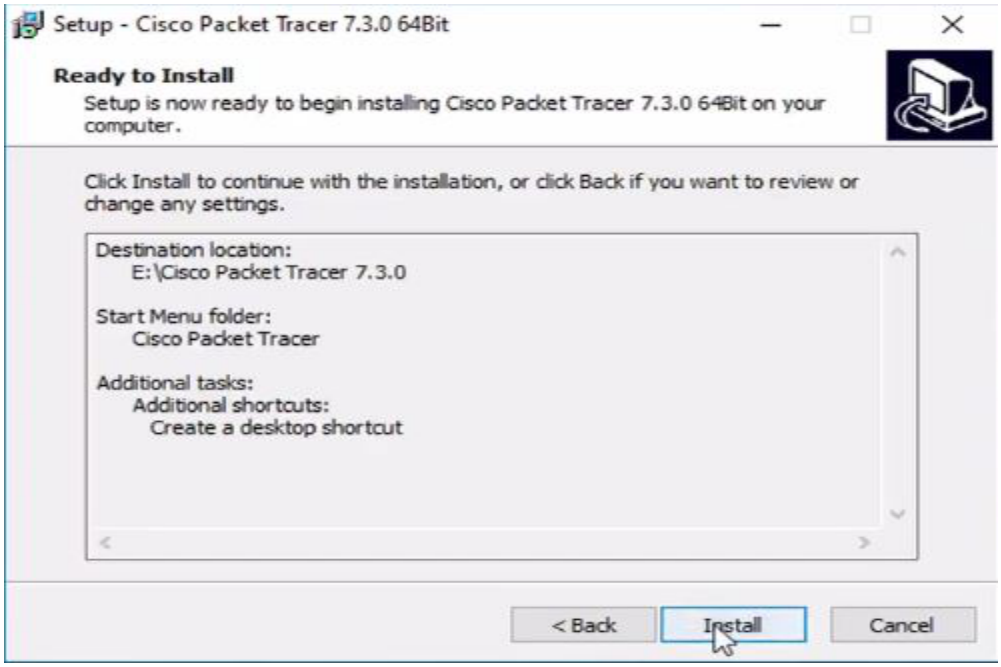
Step 12: Select the start menu folder and click the **Next** button.



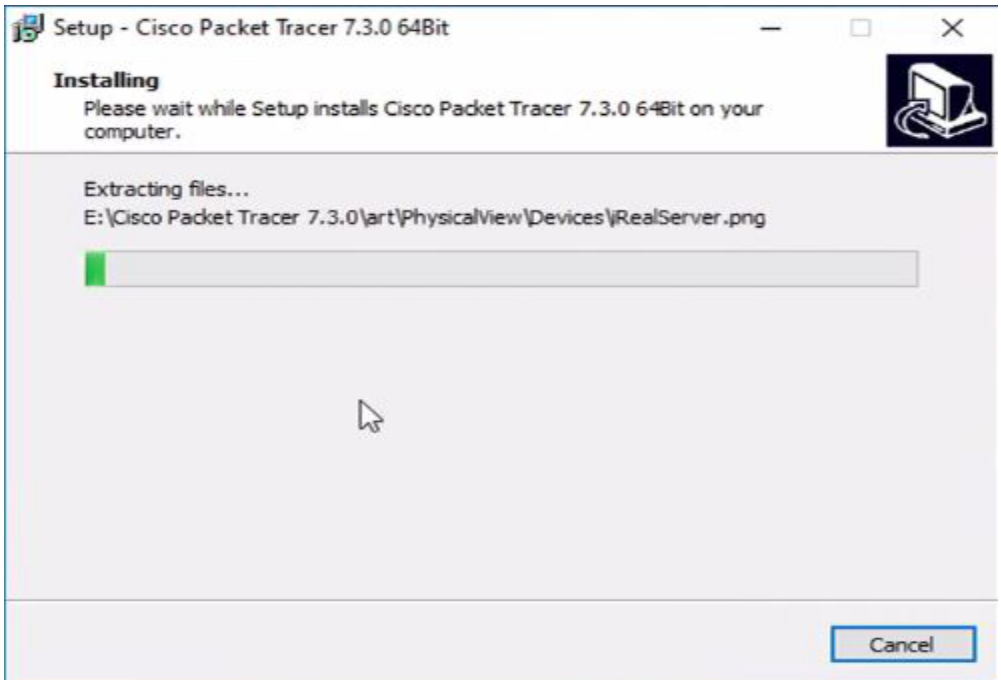
Step 13: Check the box for creating a desktop icon and click on the **Next** button.



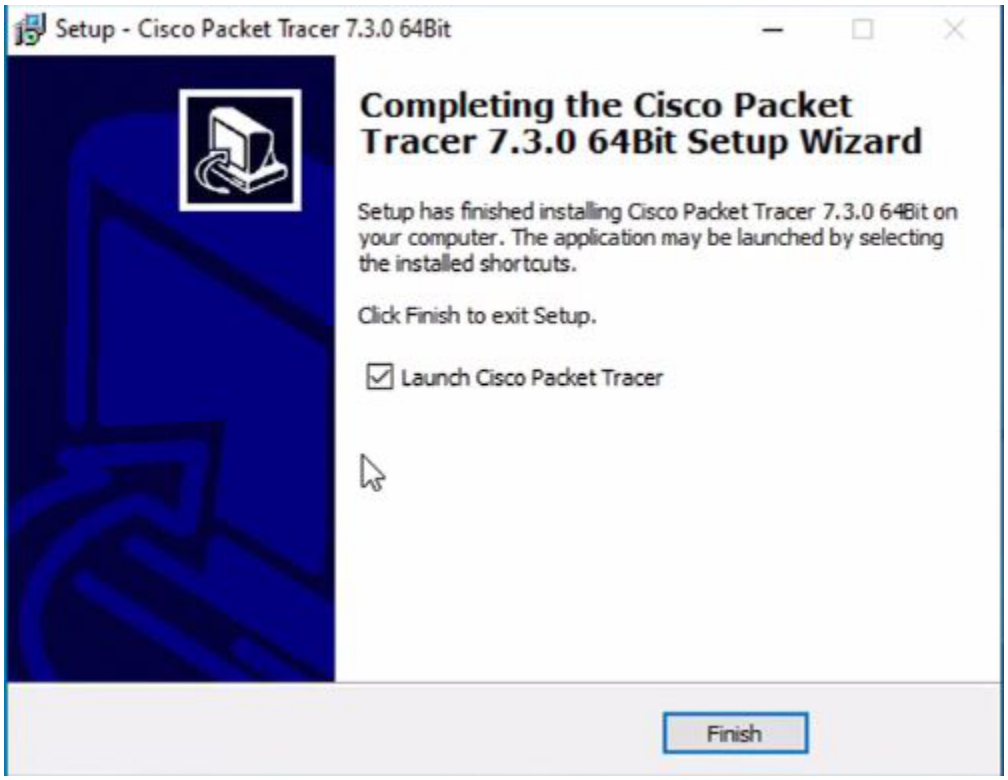
Step 14: Now packet tracer is ready to install so click on the **Install** button.



Step 15: The installation process will start and will hardly take a minute.

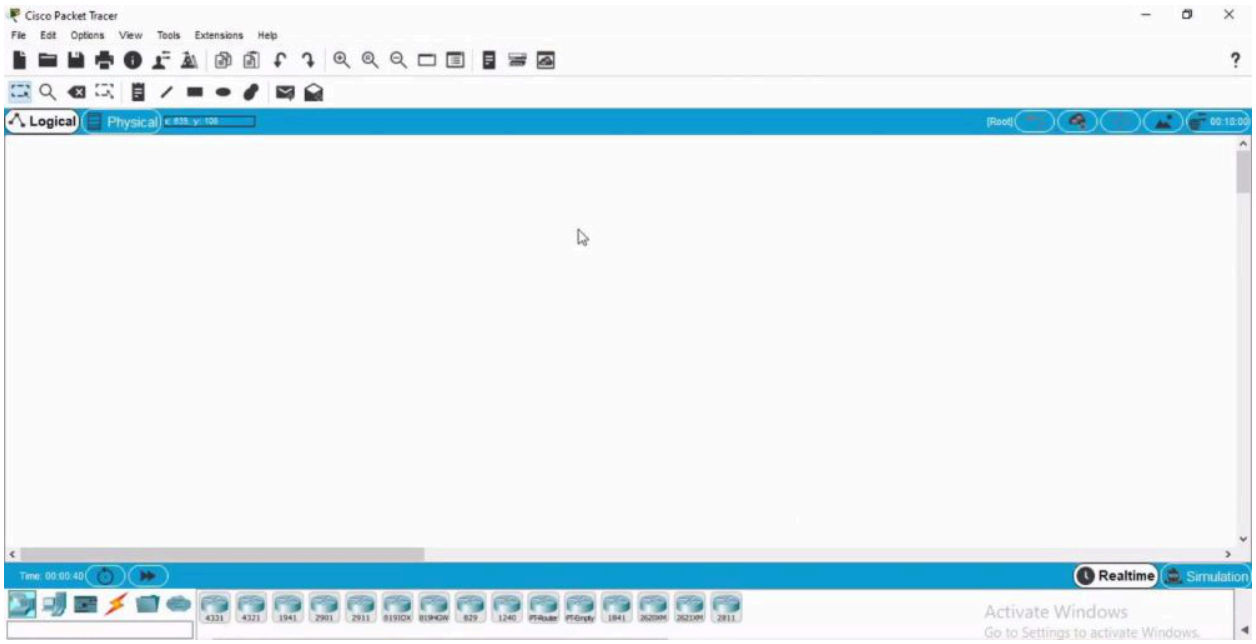


Step 16: Click on the **Finish** button to complete the installation.



Step 17: An icon is created on the desktop so run it.

Step 18: Interface is initialized and the software is ready to use.



RESULT:Thus, the CISCO PACKET TRACER was successfully installed on windows.

Ex. Nos. 4		DESIGN A SIMPLE TOPOLOGY USING CISCO PACKET TRACER
Date :	29-07-25	

AIM: To establish simple topology using nodes

PROCEDURE:

Step 1: Open Cisco Packet Tracer

- Launch the Cisco Packet Tracer application on your system.

Step 2: Add Devices to Workspace

- Drag and drop the following devices:
    - 2 PCs (PC0, PC1)
    - 1 Laptop (Laptop0)
    - 1 Printer (Printer0)
    - 1 Switch (2960-24TT)

Step 3: Connect Devices

- Use **Copper Straight-through cables** to connect:
    - Each end device (PCs, Laptop, Printer) to the Switch.
- Verify green link lights appear on both ends.

Step 4: Assign IP Addresses

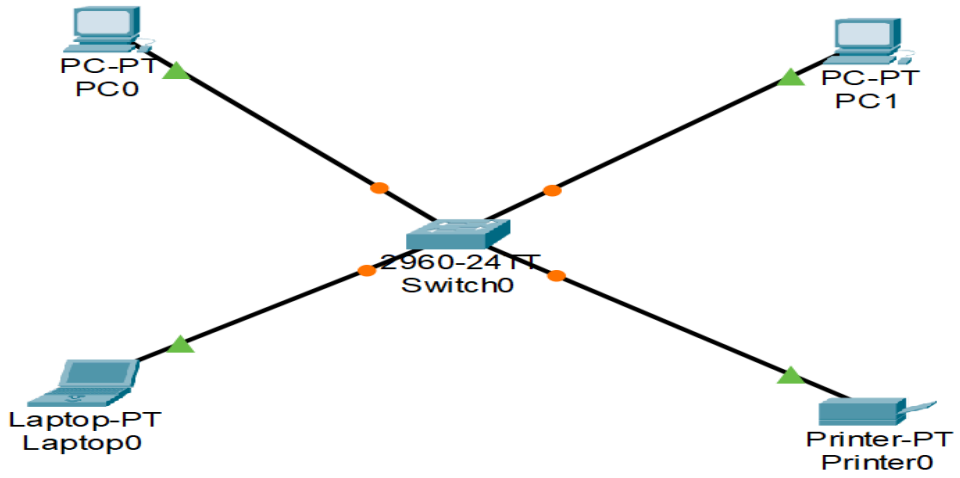
- Click on each device → Desktop tab → IP Configuration.
- Assign static IP addresses as per the IP table defined.

Device Name	IP Address	Subnet Mask	Default Gateway
PC0	192.168.0.2	255.255.255.0	192.168.1.1
PC1	192.168.0.3	255.255.255.0	192.168.1.1
Laptop0	192.168.0.4	255.255.255.0	192.168.1.1
Printer0	192.168.0.5	255.255.255.0	192.168.1.1

Step 5: Verify Connectivity

- Use the ping command from one device to another:
    - Example: ping 192.168.0.3 from PC0 to PC1.
- All pings should be successful.

BASIC SETUP USING CISCO PACKET TRACER:



OUTPUT:

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.0.4

Pinging 192.168.0.4 with 32 bytes of data:

Reply from 192.168.0.4: bytes=32 time<1ms TTL=128
Reply from 192.168.0.4: bytes=32 time<1ms TTL=128
Reply from 192.168.0.4: bytes=32 time<1ms TTL=128
Reply from 192.168.0.4: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.0.4:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

RESULT: The star topology LAN was successfully implemented and verified with proper IP configuration and device connectivity using ping.

Ex. Nos. 5		DESIGN A TOPOLOGY AND CONFIGURE WITH SWITCHES AND PCs USING CISCO PACKET TRACKER
Date :	29-07-25	

AIM: To establish a simple network using switch

PROCEDURE:

Step 1: Open Cisco Packet Tracer

- Launch the Cisco Packet Tracer software.

Step 2: Add Devices

- Add the following to the workspace:
    - 3 Switches (2950-24): Switch0, Switch1, Switch2
    - 3 PCs: PC0, PC2, PC3

Step 3: Connect the Devices

- Use Copper Straight-through cables to connect:
    - PC0 to Switch0
    - PC2 to Switch1
    - PC3 to Switch2
- Use Copper Cross-over cables (or Straight-through in new PT versions) to connect:
    - Switch0 to Switch1
    - Switch1 to Switch2

Step 4: Assign IP Addresses

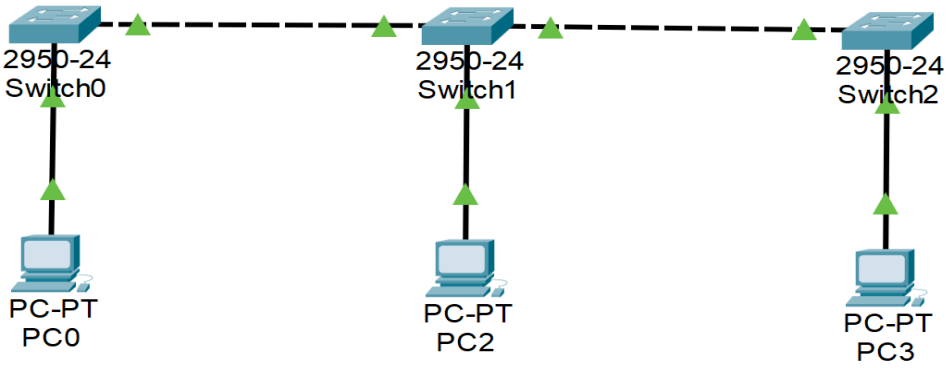
- Click on each PC → Desktop tab → IP Configuration
- Enter the following static IP addresses:

Device	IP Address	Subnet Mask	Default Gateway
PC0	192.168.0.1	255.255.255.0	192.168.1.254
PC2	192.168.0.2	255.255.255.0	192.168.1.254
PC3	192.168.0.4	255.255.255.0	192.168.1.254

Step 5: Verify Connectivity

- Go to PC0 → Open Command Prompt
- Run: ping 192.168.1.2 and ping 192.168.1.3
- Repeat from other PCs to verify successful communication

BASIC SETUP USING CISCO PACKET TRACER:



OUTPUT:

```
C:\>ping 192.168.0.4

Pinging 192.168.0.4 with 32 bytes of data:

Reply from 192.168.0.4: bytes=32 time<1ms TTL=128
Reply from 192.168.0.4: bytes=32 time<1ms TTL=128
Reply from 192.168.0.4: bytes=32 time<1ms TTL=128
Reply from 192.168.0.4: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.0.4:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

RESULT: The bus topology LAN using switches was successfully implemented and verified with proper IP configuration and device connectivity using ping

Ex. Nos. 6		DESIGN A TOPOLOGY AND CONFIGURE WITH ROUTER USING CISCO PACKET TRACKER
Date :	29-07-25	

AIM: To establish networks with switches and routers.

PROCEDURE:

Step 1: Open Cisco Packet Tracer

- Launch the Cisco Packet Tracer application.

Step 2: Add Devices

- Add the following devices to the workspace:
    - 1 Router (1841)
    - 2 Switches (2950-24) labeled Switch1 and Switch2
    - 6 PCs: PC0–PC2 under LAN-1 and PC3–PC5 under LAN-2

Step 3: Connect the Devices

- Connect each PC to its respective switch using Copper Straight-through cables.
- Connect Switch1 to Router's FastEthernet0/0
- Connect Switch2 to Router's FastEthernet0/1

Step 4: Assign IP Addresses

Device	Interface	IP Address	Subnet Mask	Network
Router	FastEthernet0/0	192.168.0.1	255.255.255.0	LAN-1
Router	FastEthernet0/1	10.0.0.1	255.0.0.0	LAN-2
PC0	NIC	192.168.0.2	255.255.255.0	LAN-1
PC1	NIC	192.168.0.3	255.255.255.0	LAN-1
PC2	NIC	192.168.0.4	255.255.255.0	LAN-1
PC3	NIC	10.0.0.2	255.0.0.0	LAN-2
PC4	NIC	10.0.0.3	255.0.0.0	LAN-2
PC5	NIC	10.0.0.4	255.0.0.0	LAN-2

Default Gateway for:

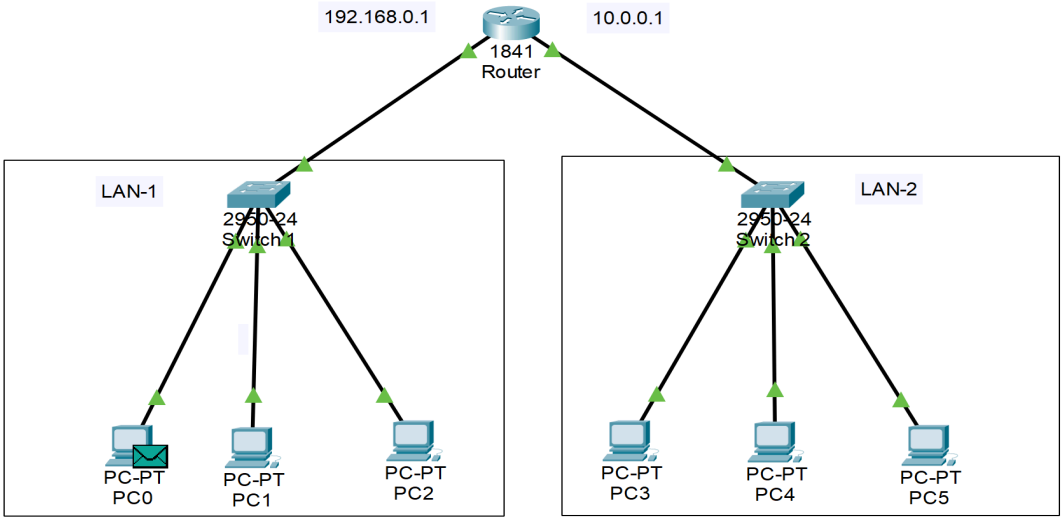
- LAN-1 PCs: 192.168.0.1
- LAN-2 PCs: 10.0.0.1

Step 5: Configure the Router Interfaces

Step 6: Verify Connectivity

- From any PC in LAN-1, ping any PC in LAN-2.
    - Example: From PC0 → ping 10.0.0.2
- All ping requests should be successful.

BASIC SETUP USING CISCO PACKET TRACER:



OUTPUT:

```
C:\>ping 10.0.0.3

Pinging 10.0.0.3 with 32 bytes of data:

Reply from 10.0.0.3: bytes=32 time<1ms TTL=127
Reply from 10.0.0.3: bytes=32 time<1ms TTL=127
Reply from 10.0.0.3: bytes=32 time=1ms TTL=127
Reply from 10.0.0.3: bytes=32 time<1ms TTL=127

Ping statistics for 10.0.0.3:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

RESULT: Two LANs were successfully interconnected using a router, and cross-network communication was verified using ping.

Ex. Nos. 7		DESIGN WIRELESS NETWORK USING CISCO PACKET TRACER
Date :	26-08-25	

AIM: To establish wireless network.

PROCUDURE:

Step 1: Open Cisco Packet Tracer

- Launch the **Cisco Packet Tracer** application.

Step 2: Add Devices

Add the following devices to your workspace:

- **2 Routers** (1941) — Label them Router8 and Router6
- **2 Switches** (2960-24TT) — Label them Switch2 (LAN-1) and Switch1 (LAN-2)
- **2 Access Points** (Access Point-PT)
- **5 End Devices** — Laptops/PCs as shown in the diagram
- **Serial DCE Cable** (to connect routers)
- **Copper Straight-Through Cables** (for PC-to-switch and switch-to-router connections)

Step 3: Connect the Devices

LAN-1 (Left Side)

- Connect **PC0, PC1, Laptop0 → AccessPoint0** (wireless connection in PT)
- Connect **AccessPoint0 → Switch2** (Copper Straight-Through)
- Connect **Switch2** → Router8 **GigabitEthernet0/0**

LAN-2 (Right Side)

- Connect **PC2, PC3 → AccessPoint1** (wireless connection)
- Connect **AccessPoint1 → Switch1** (Copper Straight-Through)
- Connect **Switch1** → Router6 **GigabitEthernet0/0**

Inter-Router Link

- Connect Router8 Serial0/0/0 → Router6 Serial0/0/0 using **Serial DCE Cable**

Step 4: Assign IP Addresses

Device	Interface	IP Address	Subnet Mask	Network
Router8	G0/0	192.168.0.1	255.255.255.0	LAN-1
Router8	S0/0/0 (DCE)	10.0.0.1	255.0.0.0	WAN link
Router6	G0/0	172.16.31.1	255.255.0.0	LAN-2
Router6	S0/0/0	10.0.0.2	255.0.0.0	WAN link
PC0	NIC	192.168.0.2	255.255.255.0	LAN-1
PC1	NIC	192.168.0.3	255.255.255.0	LAN-1
Laptop0	NIC	192.168.0.4	255.255.255.0	LAN-1
PC2	NIC	172.16.31.2	255.255.0.0	LAN-2
PC3	NIC	172.16.31.3	255.255.0.0	LAN-2

Default Gateway:

- **LAN-1 PCs/Laptop** → 192.168.0.1
- **LAN-2 PCs** → 172.16.31.1

Step 5: Configure Router Interfaces

On Router8 (DCE side):

```
enable
configure terminal
interface gigabitEthernet0/0
ip address 192.168.0.1 255.255.255.0
no shutdown
exit
interface serial0/0/0
ip address 10.0.0.1 255.0.0.0
no shutdown
exit
```

On Router6:

```
enable
configure terminal
interface gigabitEthernet0/0
ip address 172.16.31.1 255.255.0.0
no shutdown
exit
interface serial0/0/0
ip address 10.0.0.2 255.0.0.0
no shutdown
exit
```

Step 6: Add Static Routes

On Router8:

```
ip route 172.16.0.0 255.255.0.0 10.0.0.2
```

On Router6:

```
ip route 192.168.0.0 255.255.255.0 10.0.0.1
```

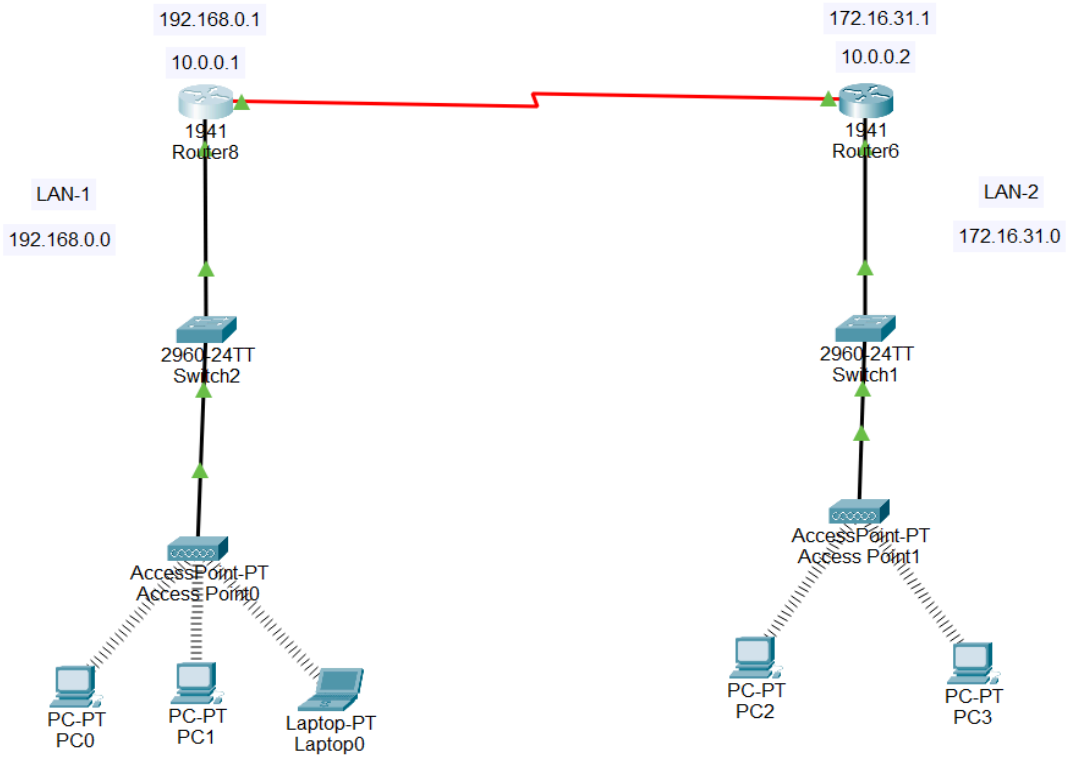
Step 7: Verify Connectivity

- From **PC0**:
  ping 172.16.31.2

- From **PC3**:
  ping 192.168.0.2

All pings should succeed.

BASIC SETUP:



RESULT: Thus,wireless network was established successfully using CISCO PacketTracer.

Ex. Nos. 8		STUDY - SIMPLE AND COMPLEX PROTOCOL DATA UNIT
Date :	26-08-25	

AIM: To perform a study experiment on simple and complex Protocol Data Units.

DESCRIPTION:

In networking, "simple PDU" and "complex PDU" refer to the complexity of the simulation of data packets in tools like Cisco Packet Tracer, rather than inherent characteristics of the Protocol Data Units (PDUs) themselves. A simple PDU simulates a basic packet movement between two directly connected devices, while a complex PDU simulation involves multiple hops through network devices, showcasing the process of de-encapsulation and encapsulation at each layer as data travels through the network.

Simple PDU Simulation

Purpose: To visualize the most basic interaction between two devices.

Process: A packet is sent from a source device to a destination device, typically without intermediate hops or complex routing decisions.

Example: Sending a packet from a PC directly to a router or another PC.

Observation: You can see the packet travel from the source to the destination, and the contents of the PDU at the destination are displayed to see how data is received and de-encapsulated.

Complex PDU Simulation

Purpose:

To demonstrate the detailed steps of data transmission across a more extensive network.

Process:

A packet travels through a series of network devices (switches, routers) and interacts with various protocols and technologies, like Spanning Tree Protocol (STP).

Example:

Sending a packet from a PC, through an access switch, core switch, and finally to a router.

Observation:

The simulation shows how data is framed at the Data Link layer, packetized at the Network layer, and how network devices process these packets, including routing decisions and the encapsulation/de-encapsulation steps.

Key Differences in Simulation

Network Path:

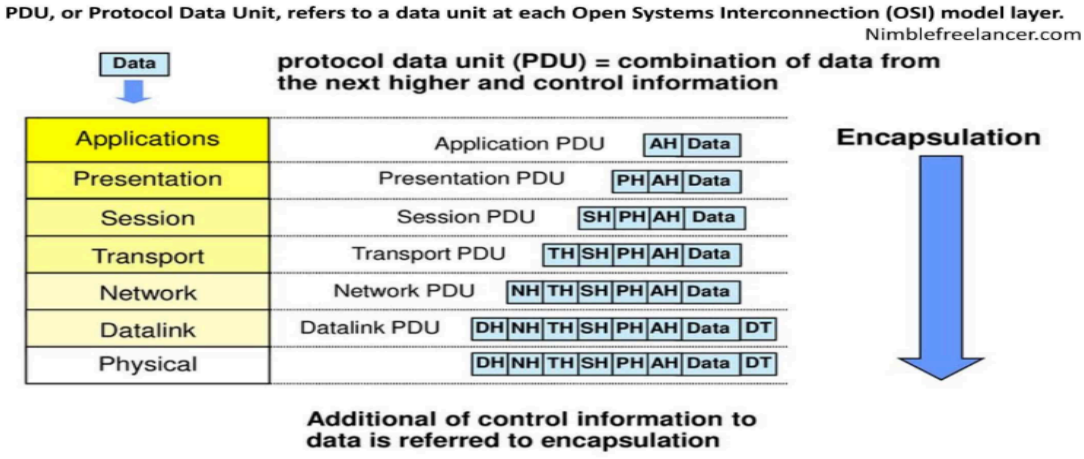
Simple PDUs follow a direct, uncomplicated path, while complex PDUs traverse a more intricate path with multiple devices.

Device Interaction:

Complex PDU simulations involve more interactions, such as frame flooding by switches or routing by routers.

Information Displayed:

Complex simulations provide more detail about how the PDU is modified at each layer and by each device.



RESULT: Thus,the study experiment on simple and complex pdu was done successfully.

Ex. Nos. 9		STUDY - SOCKET PROGRAMMING USING PYTHON
Date :	02-09-25	

AIM: To study and understand basic socket programming in Python.

DESCRIPTION:

INTRODUCTION:

In networks, the services provided to the user follow the traditional client/server model. One computer acts as a server to provide a certain service and another computer represents the client side which makes use of this service. In order to communicate over the network a network socket comes into play, mostly only referred to as a socket.

SOCKET DEFINITION:

A network socket is an endpoint of a two-way communication link between two programs or processes - client and server in our case - which are running on the network. This can be on the same computer as well as on different systems which are connected via the network.

Both client/server communicate with each other by writing to or reading from the network socket. The technical equivalent in reality is a telephone communication between two participants. The network socket represents the corresponding number of the telephone line, or a contract in case of cell phones.

SOCKET PROGRAMMING IN PYTHON

Python's core networking library is Socket Module.
Python's socket module has both class-based and instances-based methods.
Class-based method is an intuitive approach which doesn't need an instance of a socket object.
For example, in order to print a machine's IP address, you don't need a socket object. Instead, just call the socket's class-based methods.
In instance-based method, if some data needed to be sent to a server application, it is more intuitive to create a socket object to perform that explicit operation.
This module has everything you need to build socket servers and clients.

SAMPLE CLASS METHODS:

1. gethostname method: Used to get the name of the machine

```
>>> import socket
>>> socket.gethostname()
'DESKTOP-K146K1I'
```

2. gethostbyname method: used to get the IP address of the machine

```
>>> host=socket.gethostname()
>>> socket.gethostbyname(host)
'172.16.11.202'
```

remote=’www.gmail.com’ (remote host IP address can also be got)

```
>>> socket.gethostbyname(remote)
'172.217.163.37'
```

3. inet_aton() and inet_ntoa() methods: Converting IP address in decimal notation to binary format

```
import socket
for ip_addr in ['127.0.0.1', '192.168.0.1']:
packed_ip_addr = socket.inet_aton(ip_addr) [decimal notation to
binary format]
unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr) [binary format
to decimal notation]
print(packed_ip_addr)
print(unpacked_ip_addr)
```

OUTPUT:

```
b'\x7f\x00\x00\x01'
127.0.0.1
b'\xc0\xa8\x00\x01'
192.168.0.1
```

4. getserverport() method: used to get the service(protocol) name by giving its port number and transport layer protocol name.

```
import socket
protocolname = 'tcp'
for port in [80, 25]:
serp=socket.getservbyport(port,protocolname)
print(serp)
```

OUTPUT:

```
http
smtp
```

5. htonl() and ntohl() methods: used to convert data from host to network byte order and vice versa

```
import socket
data=1234
data1=socket.htonl(data)
print(data1)
print(socket.ntohl(data1))
```

OUTPUT:

```
3523477504
1234
```

SAMPLE INSTANCE METHODS:

Istance method	Description
sock.bind((adrs, port))	Bind the socket to the address and port
sock.accept()	Return a client socket (with peer address information)
sock.listen(backlog)	Place the socket into the listening state, able to pend *backlog* outstanding connection requests
sock.connect((adrs, port))	Connect the socket to the defined host and port
sock.recv(buflen[, flags])	Receive data from the socket, up to buflen bytes
sock.recvfrom(buflen[, flags])	Receive data from the socket, up to buflen bytes, returning also the remote host and port from which the data came
sock.send(data[, flags])	Send the data through the socket
sock.sendto(data[, flags], addr)	Send the data through the socket
sock.close()	Close the socket
sock.getsockopt(lvl, optname)	Get the value for the specified socket option
sock.setsockopt(lvl, optname, val)	Set the value for the specified socket option

HOW TO WORK WITH TCP SOCKETS IN PYTHON:

Socket programming with TCP

Client must contact server:

- Server must be first running
- Server must have created socket that welcomes client's contact

Client connects to server by:

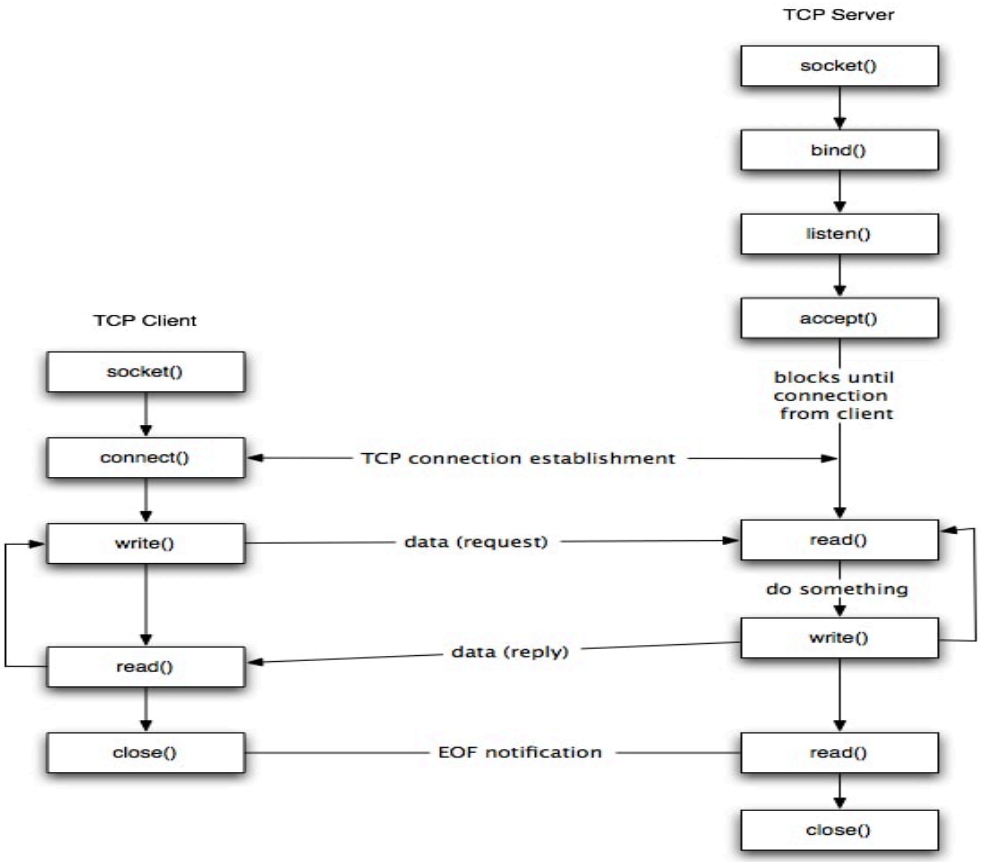
- creating TCP socket, specifying IP address, port number of server process
- client socket is now bound to that specific server

Server accepts connect by:

- creating new connection-specific socket
- allows server to talk with multiple clients

Application viewpoint:

- TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server



IMPLEMENTATION:

CLIENT:

```
import socket
try:
        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        print("socket object created successfully from client end")

except socket.error as err:
        print("socket cant be connected %s" %(err))

port=12345
s.connect(("172.16.76.82",port))
name=input("Enter your name ")
s.send(name.encode());
print(s.recv(1024).decode())

s.close()
```

SERVER:

```
import socket
try:
        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        print("socket created successfully")

except socket.error as err:
        print("Error occurred in socket creation %s" %(err))

port=12345
s.bind(('',port)) #bind an ampty string on to this port ,so it listen to any ip
print("socket binded to port ",port)
s.listen(5)
while True:
        c,addr=s.accept()
        print("Got connection from ",addr)
        name = c.recv(1024).decode()
        print("Connector name  ",name)
        c.send(("Thank you for the connection "+name).encode())
        c.close()
        break
```

OUTPUT:

```
D:\03>python socclient.py
socket object created successfully from client end
Enter your name Aryna Sabalenka
Thank you for the connection Aryna Sabalenka

D:\03>python socserver.py
socket object created successfully from server end
Got Connection from Aryna Sabalenka
Thank you for the connection Aryna Sabalenka
```

RESULT: Thus, the study exercise was performed successfully.

Ex. Nos. 10		CHAT APPLICATION USING SOCKETS IN PYTHON
Date :	02-09-25	

AIM: To build chat application using socket programming in python.

SOURCE CODE:

CLIENT SIDE:

```
import socket

try:
        s=socket.socket()
        print("socket object created successfully from client end")

except socket.err as err:
        print("Error occurred in socket creation %s %(err)")

port=12345
s.connect(("172.16.76.82",port))
while True:
        data=input("C: ")
        s.send(data.encode())
        if data=="bye":
                break


        print("S: ",s.recv(1024).decode())


s.close()
```

SERVER SIDE:

```
import socket

try:
        s=socket.socket()
        print("socket created successfully")

except socket.error as err:
        print("Error occurred in socket creation %s %(err)")

port=12345
s.bind(('',port))

print("socket binded to port ",port)

s.listen()
c,addr=s.accept()

while True:

        data=c.recv(1024).decode()
        print("C: ",data)
        if(data=="bye"):
                break

        serverdata=input("S: ")
        c.send(serverdata.encode())


s.close()
```

OUTPUT:

```
D:\03>python chatserver.py
socket created successfully
socket binded to port  12345
C:  Hi
S: Hello
C:  How are you
S: Iam fine
C:  Nice
S: okay
C:  bye

D:\03>python chatclient.py
socket object created successfully from client end
C: Hi
S:  Hello
C: How are you
S:  Iam fine
C: Nice
S:  okay
C: bye
```

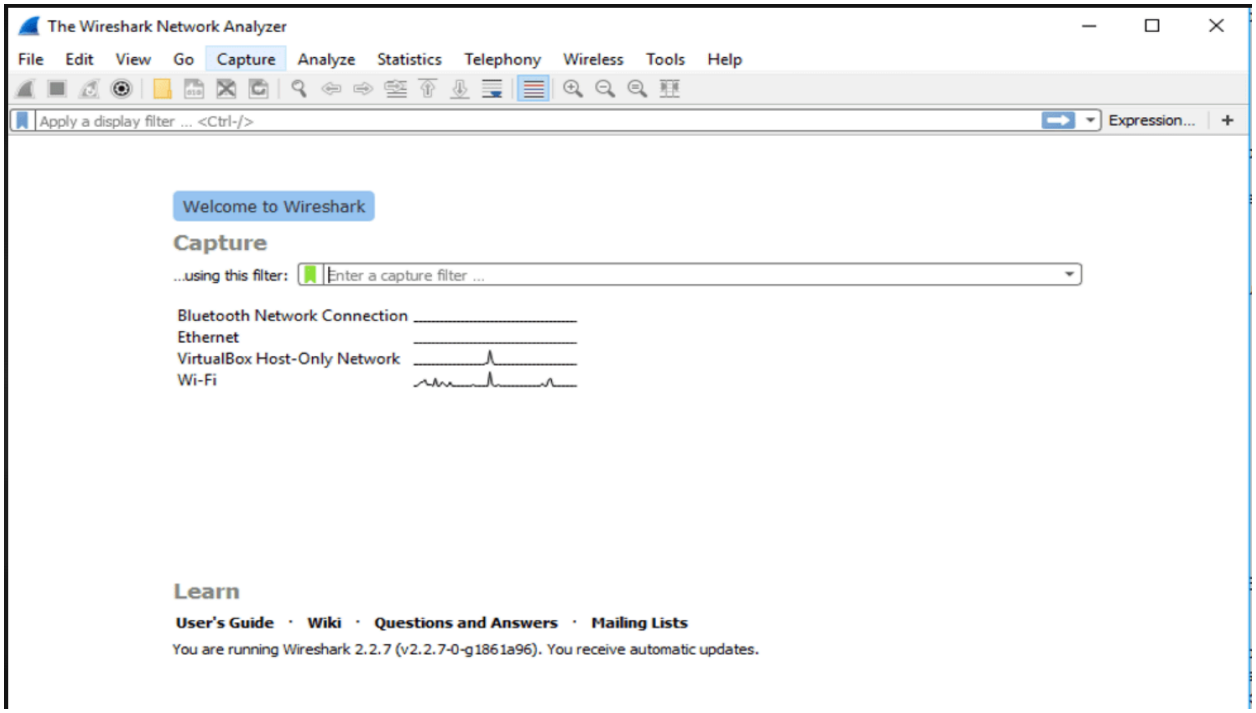
RESULT: Thus, the Chat Application using python was built successfully.

Ex. Nos. 11		STUDY - WIRESHARK TOOL
Date :	09-09-25	

AIM : To study about wireshark tool.

DESCRIPTION:

Wireshark – Network Packet Analyzer



Overview:

Wireshark is a powerful, open-source network analysis tool (formerly known as *Ethereal*) that captures and displays network packets in real time. It presents data in a human-readable format, allowing users to analyze and understand network behavior in depth. With its advanced features like filters, color-coding, and protocol decoding, Wireshark is widely used for diagnosing network issues, studying protocols, and monitoring traffic patterns.

Key Capabilities

- **Capture network traffic** in real time across different interfaces.
- **Decode protocols** using built-in dissectors.
- **Filter packets** using capture and display filters for focused analysis.
- **View smart statistics** such as packet counts and protocol distribution.
- **Analyze and troubleshoot** performance or connectivity issues.
- **Browse traffic interactively** to inspect individual packets in detail.

Common Use Cases

Wireshark is used by a wide range of professionals, including:

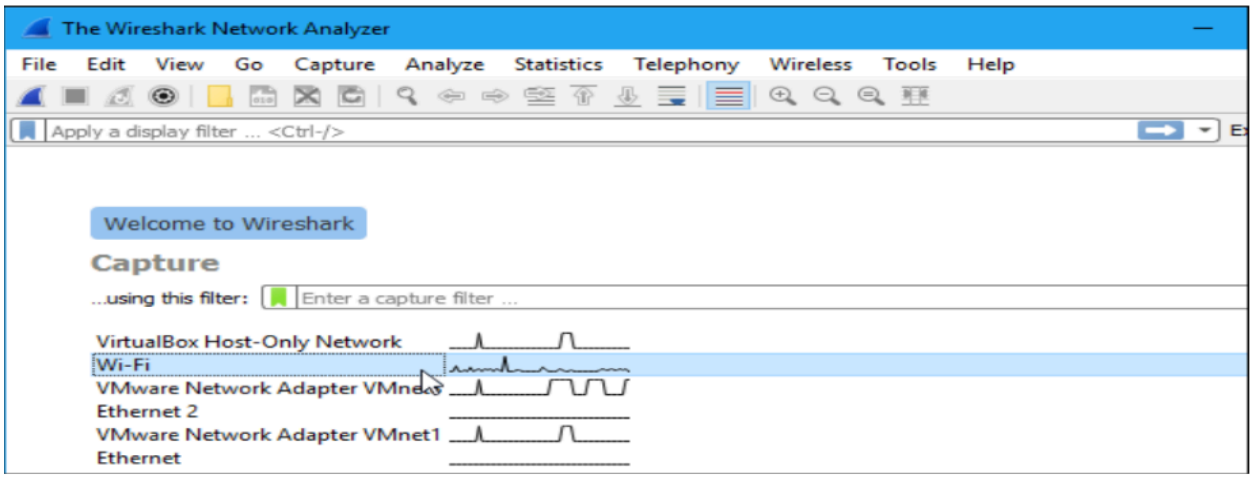
- **Network Administrators:** To identify and fix network problems.
- **Network Security Engineers:** To investigate security incidents and suspicious traffic.
- **Developers:** To debug protocol implementations and ensure compliance.
- **Learners and Researchers:** To understand how network protocols function.

Getting Wireshark

Wireshark can be downloaded for Windows and macOS from its official website.
For Linux or other UNIX-like systems, it's available through standard package repositories — for instance, Ubuntu users can find it in the *Ubuntu Software Center*.

Capturing Packets

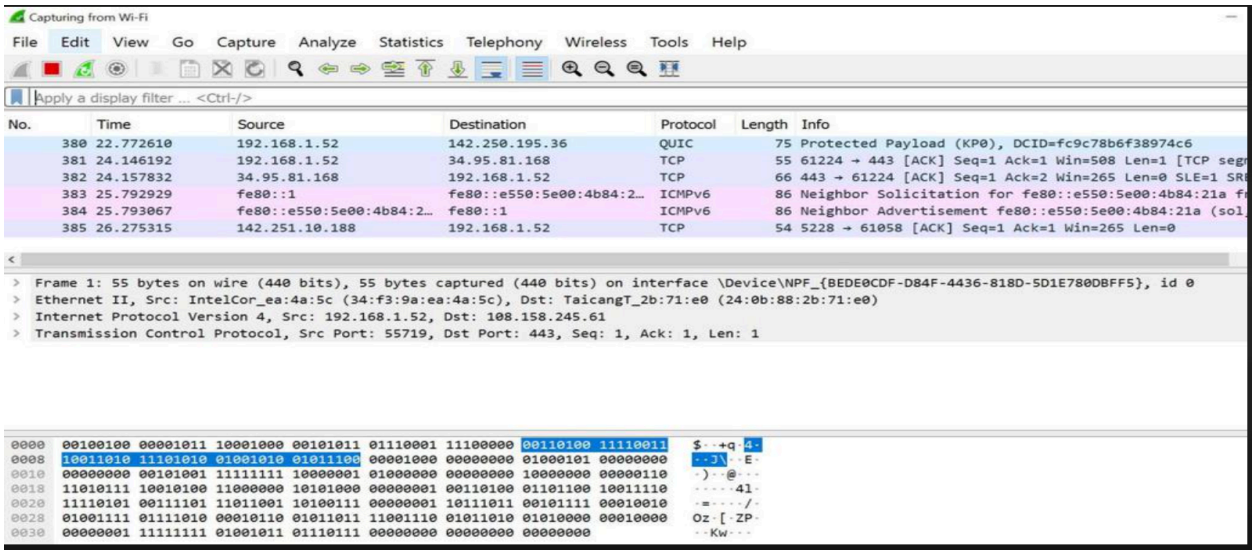
After downloading and installing **Wireshark**, launch the application.
You will see a list of available **network interfaces** (such as Wi-Fi, Ethernet, or Bluetooth).
Double-click on the name of the desired interface under **Capture** to begin capturing packets on that interface.



As soon as you start capturing, packets will begin appearing in real time.
Each line represents one packet sent or received by your system.
If **Promiscuous Mode** is enabled (enabled by default), Wireshark will capture all packets on the network, not just those addressed to your device.

You can verify this by navigating to **Capture → Options** and ensuring that the **"Enable promiscuous mode on all interfaces"** checkbox is checked.

To stop capturing packets, click the **red square "Stop" button** on the top-left toolbar.



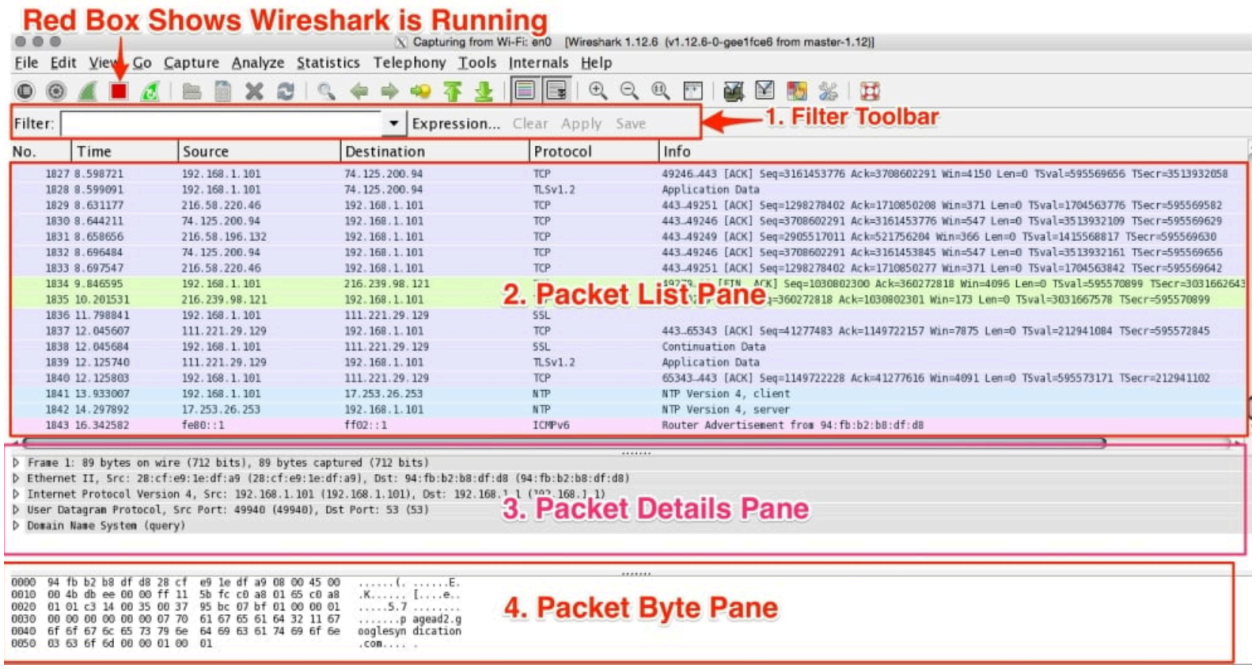
Wireshark Interface Overview

Wireshark's interface is divided into **three main panes**:

1. **Packet List Pane** – Displays all captured packets.
   Each row corresponds to a single packet. Selecting a packet here shows its details below.

2. **Packet Details Pane** – Displays the protocols and fields of the selected packet.
   The data is shown in a hierarchical tree structure that can be expanded or collapsed.

3. **Packet Bytes Pane** – Displays the raw data (in hexadecimal and ASCII) of the selected packet.

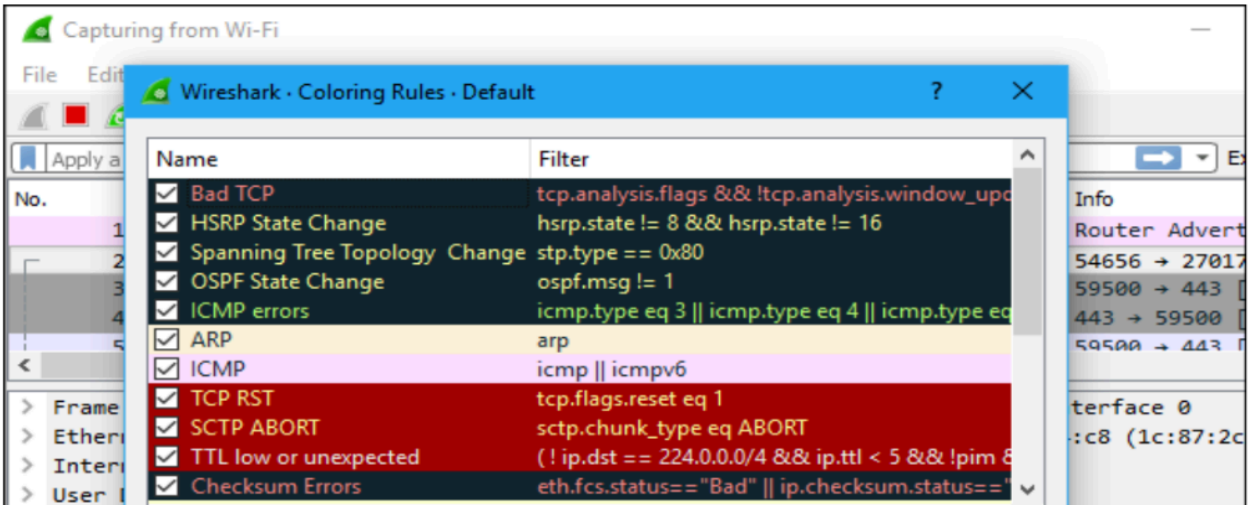


Color Coding in Wireshark

Wireshark highlights packets in different colors for easy identification:

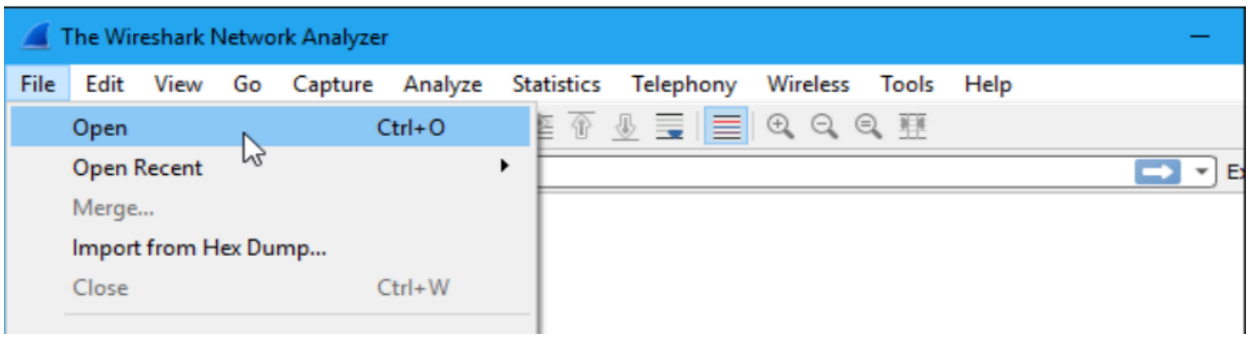
- **Light Purple:** TCP traffic
- **Light Blue:** UDP traffic
- **Black:** Packets with errors (e.g., out-of-order delivery)

To view or modify color rules, go to **View → Coloring Rules**.



Sample Captures

If there is no live network traffic to analyze, you can use sample capture files available from the Wireshark Wiki.
Download a file and open it by selecting File → Open, then browse for your .pcap file.
You can also save your own captures for later analysis using File → Save.

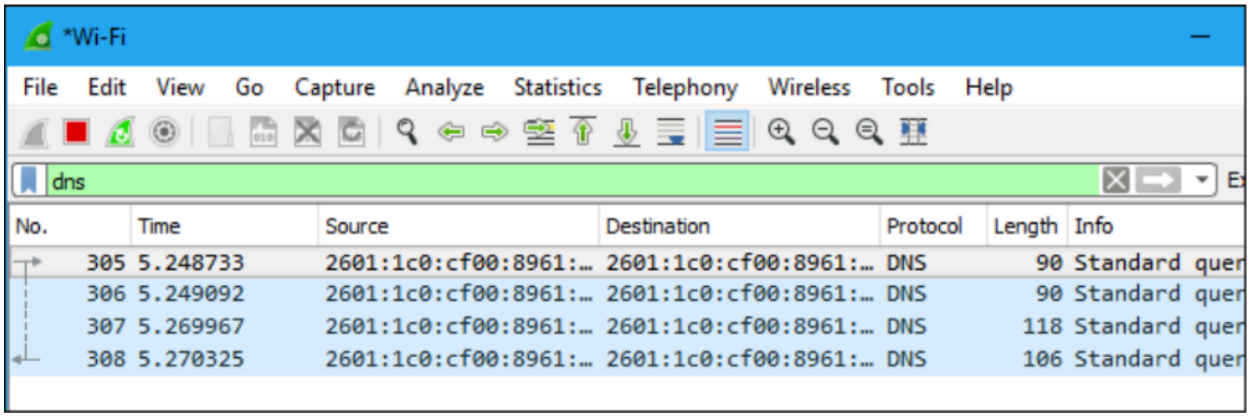


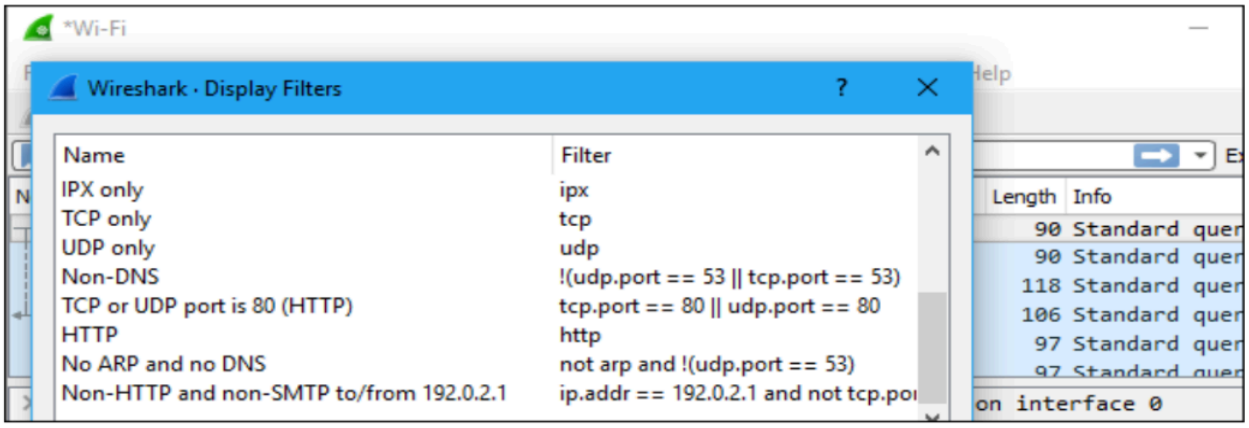
Filtering Packets

Wireshark provides filters to narrow down packet data for specific analysis.

- To apply a filter, type it in the Display Filter Bar and press Enter.
  For example, typing dns shows only DNS packets.

- You can also create or manage filters from Analyze → Display Filters.



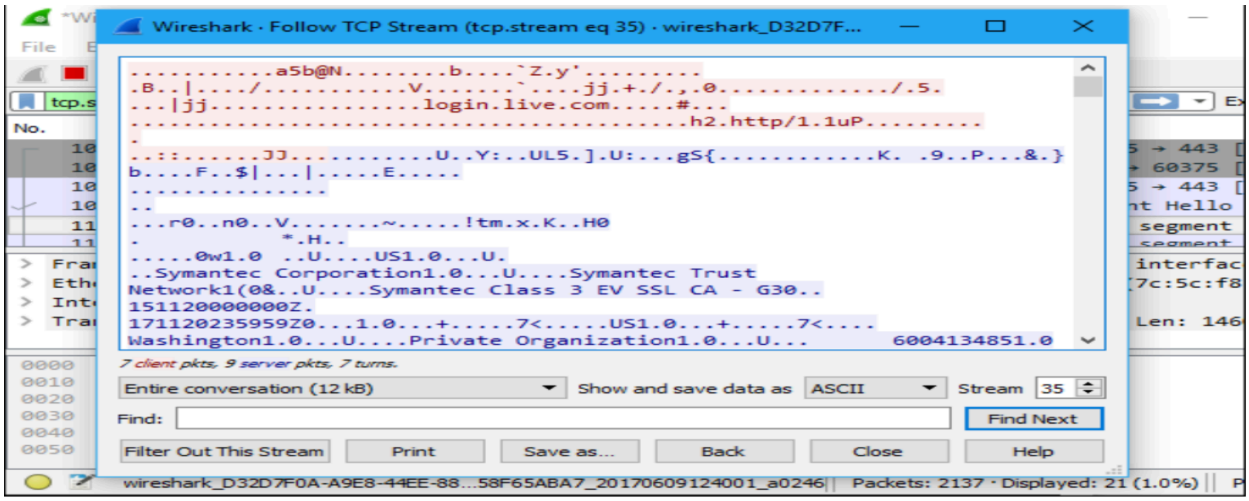


For more details, refer to *"Building display filter expressions"* in the official Wireshark documentation.

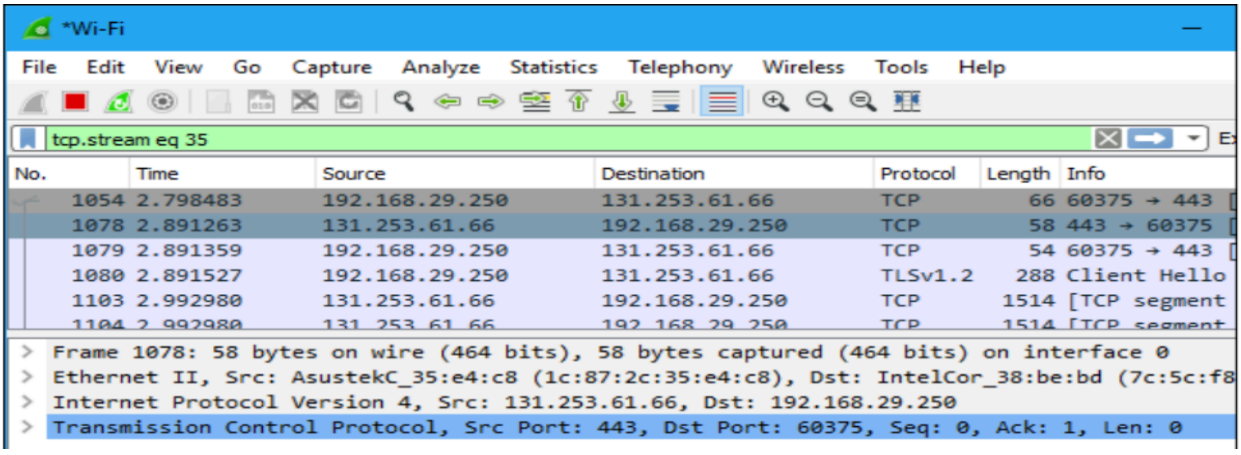
Following TCP Streams

To analyze complete conversations:

1. Right-click a packet in the list.
2. Select **Follow → TCP Stream** (or another protocol stream).
3. Wireshark will display the full communication between the client and server.

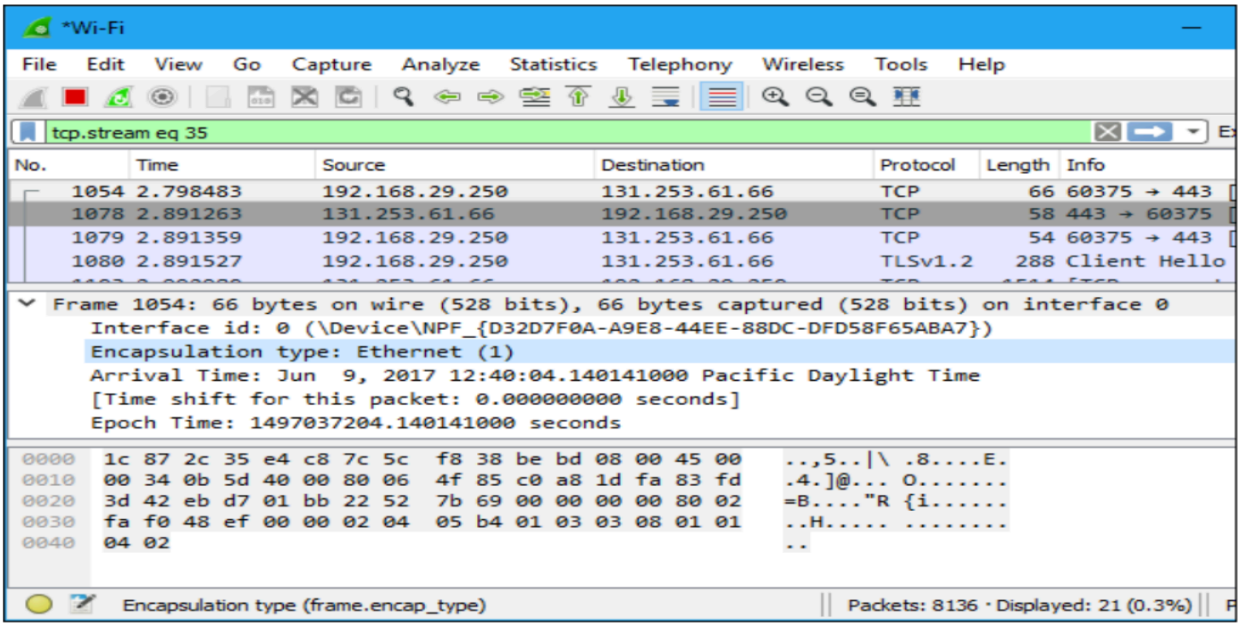


Close the window and you'll find a filter has been applied automatically. Wireshark is showing you the packets that make up the conversation.

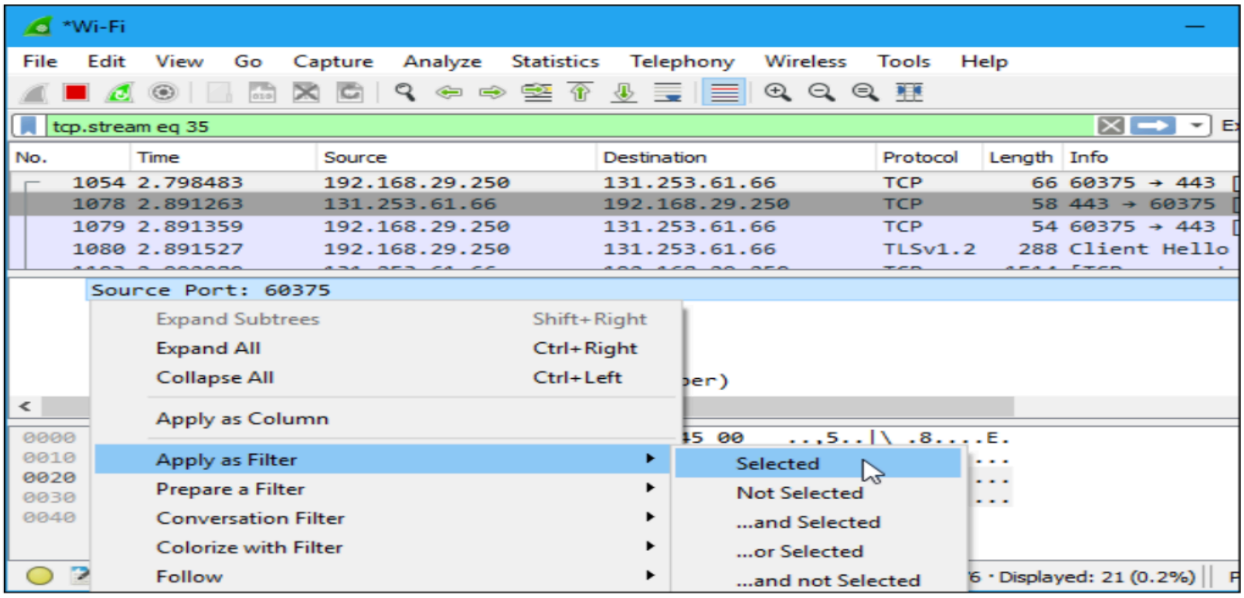


Inspecting Packets

Select any packet to view its detailed breakdown by protocol layer (Ethernet, IP, TCP, etc.).



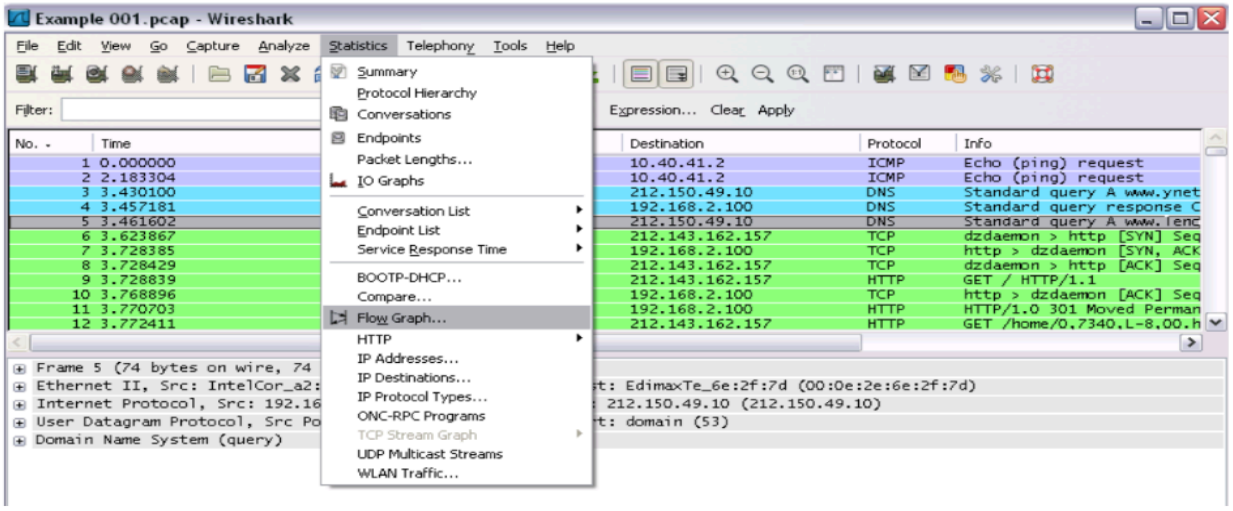
You can also right-click a field and use **Apply as Filter → Selected → A → B** to filter similar packets.

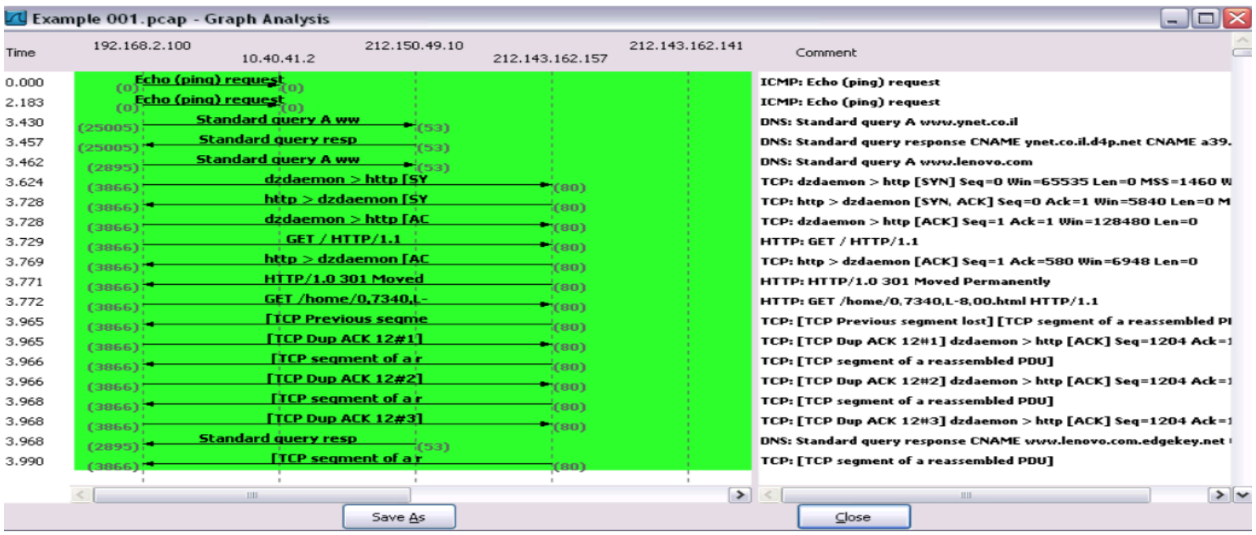


Flow Graph

Wireshark provides a **Flow Graph** feature for a visual representation of the communication between network nodes.
It helps understand how packets travel between source and destination over time.

To access it:
Go to **Statistics → Flow Graph** and select your desired parameters.





RESULT : Thus, the wireshark tool was studied successfully.

Ex. Nos. 12		PERFORM ERROR DETECTION AND ERROR CORRECTION USING HAMMING CODE [C /PYTHON]
Date :	16-09-25	

AIM: To implement Hamming Code for error detection and correction in digital data transmission using Python.

DESCRIPTION:

Hamming Code is an error-detecting and error-correcting code developed by Richard W. Hamming.

- It adds parity bits at specific positions (powers of 2) in a data word to detect and correct single-bit errors during transmission.
- In this experiment, message bits are encoded with parity bits (using even parity).
- At the receiver side, parity checks are recalculated; any mismatch produces a **syndrome** that identifies the exact bit in error.
- The erroneous bit is then flipped automatically to recover the correct codeword.

KEY STEPS:

1. Calculate number of parity bits p required for message bits k using $2^p \geq k + p + 1$.
2. Insert parity bits at positions that are powers of 2 (from the right).
3. Compute parity bits using XOR for even parity.
4. Introduce a single-bit error to test correction.
5. Detect and correct the error using the syndrome.

CODE:

```
#Hamming Code

def calculate_parity_bits(k):
    p=0
    while (1 << p) < (p+k+1):
        p+=1
    return p


def encode(message,k):
    data=list(map(int,message))
    p=calculate_parity_bits(k)
    n=k+p
    code=[0]*n
    j=0
    parity_posns=[n-2**i for i in range(0,p)]

    #fill msg bits from left to right
    for i in range(n):
        if i in parity_posns:
            continue
        code[i]=data[j]
        j+=1
```

```
  #fill parity bits from right to left
    for i_left in range(p):
        posn_right=1 << i_left
        posn_left=n-posn_right
        x_or=0
        for j_left in range(n):
            p_right=n-j_left
            if posn_right & p_right:
                x_or^=code[j_left]
        code[posn_left]=x_or


    return code


def correction(code):
    n=len(code)
    p=0
    while((1 << p) < (n+1)):
        p+=1
    syndrome=0
    for i_left in range(p):
        posn_right=1 << i_left
        x_or=0
        for j_left in range(n):
```

```
        p_right=n-j_left
        if posn_right & p_right:
                x_or^=code[j_left]

    if(x_or):
        print("Error at bit ",posn_right," from right")
        syndrome+=posn_right

  if syndrome:
          err_left=n-syndrome
           if 0<=err_left<n:
                   code[err_left]^=1

return code,syndrome
```

```
if __name__=="__main__":
    message=input("Enter msg bits")
    encoded_word=encode(message,len(message))
    print("Encoded codeword is ","".join(map(str,encoded_word)))
    encoded_word[-4]^=1
    print("Received codeword is ","".join(map(str,encoded_word)))
    corrected_word,syndrome=correction(encoded_word)

    if(not(syndrome)):
        print("No error")
     else:
        print("Corrected Codeword is ","".join(map(str,corrected_word)))
```

OUTPUT:

```
Enter msg bits1001101
Encoded codeword is  10011100101
Received codeword is  10011101101
Error at bit  4  from right
Corrected Codeword is  10011100101
```

RESULT : The Python program successfully encodes the data, detects the position of any single-bit error, and corrects it to obtain the original codeword.

Ex. Nos. 13		DEVELOP A CUSTOMIZED PING COMMAND TO TEST THE SERVER CONNECTIVITY
Date :	23-09-25	

AIM: To write a Python program that executes the system ping command to test the connectivity of a host or IP address and capture the response.

DESCRIPTION:

The **ping** command is a basic network utility used to check whether a host (IP address or domain name) is reachable on a network. It sends **ICMP (Internet Control Message Protocol) Echo Requests** and waits for Echo Replies from the target.

In this program:

1. **subprocess module** is used to run the ping command directly from Python and capture its output.
    - subprocess.run() executes system-level commands.
    - The parameters capture_output=True and text=True ensure that the output is captured as readable text.

2. **platform module** detects the operating system.
    - On **Windows**, the ping command uses -n.
    - On **Linux/macOS**, it uses -c.

3. The program handles errors using **exception handling**:
    - CalledProcessError → ping command failed (host unreachable).
    - FileNotFoundError → ping command not found on the system.
    - General Exception → any other unexpected errors.

4. The user inputs the **hostname or IP address**, and the program displays whether the ping was successful along with the full command output.

CODE:

```
import subprocess
import platform

def simple_ping(host):
        if platform.system().lower()=="windows":
                command=["ping","-n","4",host]
        else:
                command=["ping","-c","4",host]


        try:
                result=subprocess.run(command,capture_output=True,check=True,text=True)
                print("Capture  successful for the host ",host,"\n",result.stdout)

        except subprocess.CalledProcessError as e:
                print("Ping failed for the host : ",host,"\n",e.stderr)

        except FileNotFoundError:
                print("Error: Ping Command not found : ")

        except Exception as e:
                print("An unexpected Error encountered ",e)


if __name__=="__main__":
        host=input("Enter host name or ip address to ping")
        simple_ping(host)
```

OUTPUT:

```
D:\03>python simpleping.py
Enter host name or ip address to pinggoogle.com
Capture  successful for the host  google.com

Pinging google.com [142.251.43.238] with 32 bytes of data:
Reply from 142.251.43.238: bytes=32 time=8ms TTL=117
Reply from 142.251.43.238: bytes=32 time=39ms TTL=117
Reply from 142.251.43.238: bytes=32 time=14ms TTL=117
Reply from 142.251.43.238: bytes=32 time=19ms TTL=117

Ping statistics for 142.251.43.238:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 8ms, Maximum = 39ms, Average = 20ms
```

RESULT: Thus, the Python program to perform a ping test using the system command and to display the network connectivity result was executed successfully.

Ex. Nos. 14		SLIDING WINDOW
Date :	30-09-25	

AIM: To simulate and compare the working of Sliding Window Protocols — namely Go-Back-N (GBN) and Selective Repeat (SR) — using Python, to understand how reliable data transmission is achieved in computer networks.

DESCRIPTION:

The **Sliding Window Protocol** is a key data link layer technique that ensures reliable and efficient transmission of frames between sender and receiver.It allows multiple frames to be sent before waiting for acknowledgment (ACK), improving channel utilization and throughput.

Two main types of sliding window protocols are demonstrated:

1. Go-Back-N (GBN):

- The sender can send several frames up to the *window size* without waiting for ACKs.
- The receiver acknowledges only the **last correctly received frame**.
- If a single frame is lost or damaged, the sender **retransmits the entire window** starting from that frame.
- It is simpler but may cause redundant retransmissions.

2. Selective Repeat (SR):

- Each frame is acknowledged **individually** by the receiver.
- If a frame is lost, the sender **resends only that specific frame**, not the entire window.
- This approach improves efficiency and reduces bandwidth waste, especially over long-distance or noisy links.

Both techniques are implemented in Python using random loss simulation and timed transmissions (time.sleep()), illustrating how frames are sent, acknowledged, and retransmitted when errors occur.

IMPLEMENTATION:

APPROACH-1 : GBN

```
import random
import time

def sender(total_frames, window_size):
    sent = 0
    while sent < total_frames:
        end = min(sent + window_size, total_frames)
        print(f"\nSender: Sending frames {sent} to {end - 1}")

        ack = receiver(sent, end)
        if ack == -1:
            print("Receiver: Frame lost! Resending window...")
        else:
            print(f"Receiver: ACK received for frame {ack}")
            sent = ack + 1

        time.sleep(1)

def receiver(start, end):
    # Random loss rate
    if random.random() < 0.5:
        return -1
    return end - 1

# ---- Get input from user ----
total_frames = int(input("Enter total number of frames: "))
window_size = int(input("Enter window size: "))

sender(total_frames, window_size)
```

OUTPUT:

```
Enter total number of frames: 10
Enter total number of frames: 10
Enter window size: 4

Sender: Sending frames 0 to 3
Receiver: ACK received for frame 3

Enter window size: 4

Sender: Sending frames 0 to 3
Receiver: ACK received for frame 3


Sender: Sending frames 0 to 3
Receiver: ACK received for frame 3

Receiver: ACK received for frame 3

Sender: Sending frames 4 to 7
Receiver: ACK received for frame 7

Sender: Sending frames 4 to 7
Receiver: ACK received for frame 7


Sender: Sending frames 8 to 9
Sender: Sending frames 8 to 9
Receiver: Frame lost! Resending window...
Receiver: Frame lost! Resending window...

Sender: Sending frames 8 to 9
Receiver: ACK received for frame 9
```

APPROACH-2 SR

```
import random
import time

def sender(total_frames,window_size):
    sent = 0
    acked = [False] * total_frames

    while sent < total_frames:
        window_start = sent
        window_end = min(sent + window_size, total_frames)

        print(f"\nSender: Sending frames {window_start} to {window_end - 1}")

        for i in range(window_start, window_end):
            if not acked[i]:
                print(f"  -> Sending frame {i}")
                if receiver(i):
                    print(f"  <- ACK received for frame {i}")
                    acked[i] = True
                else:
                    print(f"  <- Frame {i} lost! Will resend later")

        while sent < total_frames and acked[sent]:
            sent += 1

        time.sleep(1)

    print("\n All frames successfully sent and acknowledged!")

def receiver(frame):
    # Simulate random frame loss (15% chance)
    if random.random() < 0.3:
        return False
    return True

# ---- Get input from user ----
total_frames = int(input("Enter total number of frames: "))
window_size = int(input("Enter window size: "))

sender(total_frames, window_size)
```

OUTPUT:

```
Enter total number of frames: 10
Enter window size: 4

Sender: Sending frames 0 to 3
  -> Sending frame 0
  <- ACK received for frame 0
  -> Sending frame 1
  <- Frame 1 lost! Will resend later
  -> Sending frame 2
  <- Frame 2 lost! Will resend later
  -> Sending frame 3
  <- ACK received for frame 3

Sender: Sending frames 1 to 4
  -> Sending frame 1
  <- ACK received for frame 1
  -> Sending frame 2
  <- ACK received for frame 2
  -> Sending frame 4
  <- ACK received for frame 4

Sender: Sending frames 5 to 8

Sender: Sending frames 5 to 8
Sender: Sending frames 5 to 8
  -> Sending frame 5
  -> Sending frame 5
  <- ACK received for frame 5
  -> Sending frame 6
  <- ACK received for frame 6
  -> Sending frame 7
  <- ACK received for frame 7
  -> Sending frame 8
  <- ACK received for frame 6
  -> Sending frame 7
  <- ACK received for frame 7
  -> Sending frame 8
  -> Sending frame 7
  <- ACK received for frame 7
  -> Sending frame 8
  <- ACK received for frame 8
  <- ACK received for frame 8
```

Sender: Sending frames 9 to 9
-> Sending frame 9
<- Frame 9 lost! Will resend later

Sender: Sending frames 9 to 9
-> Sending frame 9
<- ACK received for frame 9

All frames successfully sent and acknowledged!

RESULT: The simulation was successfully executed for both Go-Back-N and Selective Repeat Sliding Window Protocols.

- In Go-Back-N, when a frame was lost, the sender retransmitted the entire window of frames again until acknowledgment was received.

- In Selective Repeat, only the specific lost frame was retransmitted, demonstrating better efficiency and reduced retransmission overhead.

Thus, the experiment clearly shows how both protocols ensure reliable data transfer, with Selective Repeat being more efficient in error-prone environments.

Ex. Nos. 15		ANALYZE THE DIFFERENT TYPES OF SERVERS USING WEBALIZER TOOL
Date :	07-10-25	

AIM: To analyze log files using webalizer.

DESCRIPTION:

1. Introduction

- **Webalizer** is an open-source web log analysis tool used to generate statistical reports on website usage.
- It converts raw web server log files into **graphical and tabular summaries**, helping users understand visitor activity and traffic patterns.
- The reports are displayed in **HTML format**, which can be easily viewed in any web browser.

2. Objective

- To analyze and visualize web server logs for better understanding of website performance and user access behavior.
- To present statistical data such as **hits, visits, pages viewed, referrers, and user agents** in an easily interpretable form.

3. Working Principle

- Webalizer reads **web server log files** (e.g., Apache access logs).
- It extracts information like:
    - IP address of the visitor
    - Date and time of access
    - Requested URL or page
    - HTTP response code
    - Bytes transferred

    - Referrer and browser details
- The tool then **analyzes and summarizes** this information into monthly, daily, and hourly reports.

4. Input and Output

- **Input:** Log files in **Common Log Format (CLF)** or **Combined Log Format**.
- **Output:**
    - HTML files containing visual graphs and data tables.
    - Metrics include **Hits, Files, Pages, Visits, Sites, and Data Transferred**.
    - Additional sections: Top URLs, Entry/Exit pages, Browsers, Referrers, and Countries.

5. Features

- Generates **graphical HTML reports** with color-coded charts.
- Supports **incremental updates** without reprocessing old logs (-p option).
- Fast, efficient, and capable of handling large log files.
- Cross-platform support (Linux, Unix, Windows via WSL).
- No database required; runs directly from the command line.
- Open source under the **GNU General Public License (GPL)**.

6. Procedure

1.Install using the command
>> sudo apt-get install webalizer

2.Obtain or generate a sample web server log file (e.g., access.log).

3.Run Webalizer to generate the report:
>>webalizer access.log -o ./webalizer_report -t "Website Analysis Report"

4.Open the index.html file inside the output folder to view the statistics.

IMPLEMENTATION:

@user03 ➜ /workspaces/Linux_os_cmds (main) $ sudo apt-get install webalizer -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
webalizer is already the newest version (2.23.08-3.3build2).
0 upgraded, 0 newly installed, 0 to remove and 76 not upgraded

@user03 ➜ /workspaces/Linux_os_cmds (main) $ wget -O apache_sample.log "https://raw.githubusercontent.com/elastic/examples/master/Common%20Data%20Formats/apache_logs/apache_logs"
--2025-10-16 06:10:18--
https://raw.githubusercontent.com/elastic/examples/master/Common%20Data%20Formats/apache_logs/apache_logs
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.108.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2370789 (2.3M) [text/plain]
Saving to: ‘apache_sample.log’

apache_sample.log
100%[===>] 2.26M 6.60MB/s in 0.3s

2025-10-16 06:10:19 (6.60 MB/s) - ‘apache_sample.log’ saved [2370789/2370789]

@user03 ➜ /workspaces/Linux_os_cmds (main) $ mkdir -p webalizer_report

@user03 ➜ /workspaces/Linux_os_cmds (main) $ webalizer apache_sample.log -o ./webalizer_report

Webalizer V2.23-08 (Linux 6.8.0-1030-azure x86_64) locale: /var/www/webalizer
Using logfile apache_sample.log (clf)
Creating output in ./webalizer_report
Hostname for reports is 'codespaces-14a7ff'
History file not found...
Generating report for May 2015
Saving history information...
Generating summary report

10000 records in 1 seconds, 10000/sec

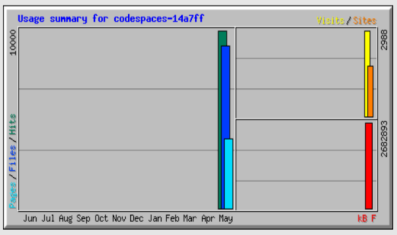
@user03 ➜ /workspaces/Linux_os_cmds (main) $ python3 -m http.server 8080 -d webalizer_report

Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
127.0.0.1 - - [16/Oct/2025 06:11:36] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [16/Oct/2025 06:11:36] "GET /usage.png HTTP/1.1" 200 -
127.0.0.1 - - [16/Oct/2025 06:11:37] code 404, message File not found
127.0.0.1 - - [16/Oct/2025 06:11:37] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [16/Oct/2025 06:13:24] "GET /usage_201505.html HTTP/1.1" 200 -
127.0.0.1 - - [16/Oct/2025 06:13:24] "GET /daily_usage_201505.png HTTP/1.1" 200 -
127.0.0.1 - - [16/Oct/2025 06:13:24] "GET /hourly_usage_201505.png HTTP/1.1" 200 -
127.0.0.1 - - [16/Oct/2025 06:13:24] "GET /ctry_usage_201505.png HTTP/1.1" 200 -

OUTPUT:

Usage statistics for codespaces-14a7ff

Summary by Month
Generated 16-Oct-2025 06:11 UTC



Summary by Month										
Month	Daily Avg				Monthly Totals					
	Hits	Files	Pages	Visits	Sites	kB F	Visits	Pages	Files	Hits
May 2015	2500	2281	974	747	1753	2682893	2988	3898	9126	10000
Totals						**2682893**	**2988**	**3898**	**9126**	**10000**

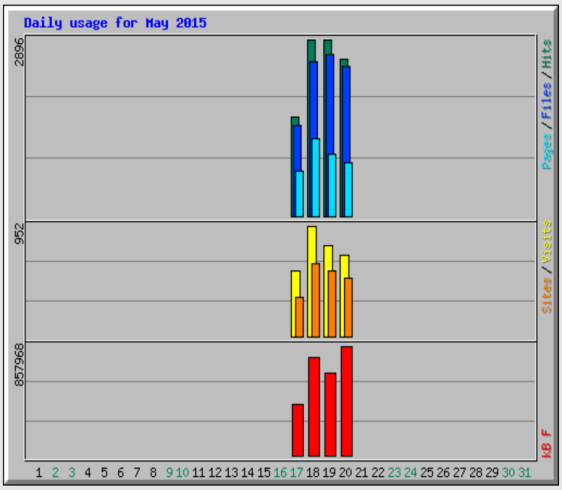
Generated by Webalizer Version 2.23

1. Monthly Summary:

[Daily Statistics] [Hourly Statistics] [URLs] [Entry] [Exit] [Sites] [Referrers] [Search] [Agents] [Countries]

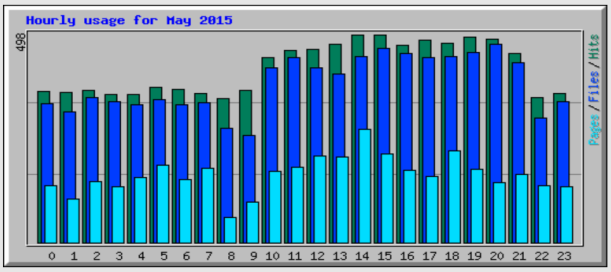
Monthly Statistics for May 2015		
Total Hits		10000
Total Files		9126
Total Pages		3898
Total Visits		2988
Total kB Files		2682893
Total Unique Sites		1753
Total Unique URLs		1259
Total Unique Referrers		404
Total Unique User Agents		551
	Avg	Max
Hits per Hour	104	136
Hits per Day	2500	2896
Files per Day	2281	2645
Pages per Day	974	1277
Sites per Day	438	627
Visits per Day	747	952
kB Files per Day	670723	857968
Hits by Response Code		
Code 200 - OK	91.26%	9126
Code 206 - Partial Content	0.45%	45
Code 301 - Moved Permanently	1.64%	164
Code 304 - Not Modified	4.45%	445
Code 403 - Forbidden	0.02%	2
Code 404 - Not Found	2.13%	213
Code 416 - Requested Range Not Satisfiable	0.02%	2
Code 500 - Internal Server Error	0.03%	3

2. Daily and Hourly Graphs:



Daily Statistics for May 2015

Day	Hits		Files		Pages		Visits		Sites		kB F	
17	1632	16.32%	1496	16.39%	731	18.75%	559	18.71%	341	19.45%	404551	15.08%
18	2893	28.93%	2534	27.77%	1277	32.76%	952	31.86%	627	35.77%	770152	28.71%
19	2896	28.96%	2645	28.98%	1009	25.89%	778	26.04%	561	32.00%	650222	24.24%
20	2579	25.79%	2451	26.86%	881	22.60%	699	23.39%	505	28.81%	857968	31.98%



Hourly Statistics for May 2015

Hour	Hits			Files			Pages			kB F		
	Avg	Total		Avg	Total		Avg	Total		Avg	Total	
0	90	361	3.61%	83	333	3.65%	33	135	3.46%	7426	29704	1.11%
1	90	360	3.60%	78	313	3.43%	26	105	2.69%	22302	89209	3.33%
2	91	365	3.65%	86	347	3.80%	36	146	3.75%	38604	154415	5.76%
3	88	354	3.54%	84	338	3.70%	33	133	3.41%	26405	105620	3.94%
4	88	355	3.55%	82	331	3.63%	38	155	3.98%	55633	222533	8.29%
5	92	371	3.71%	85	342	3.75%	46	185	4.75%	36442	145769	5.43%
6	91	366	3.66%	82	329	3.61%	37	150	3.85%	19074	76298	2.84%
7	89	357	3.57%	84	336	3.68%	44	179	4.59%	17859	71434	2.66%
8	86	345	3.45%	68	273	2.99%	15	61	1.56%	6271	25083	0.93%
9	91	364	3.64%	64	257	2.82%	24	97	2.49%	18396	73585	2.74%
10	110	443	4.43%	104	418	4.58%	42	171	4.39%	6512	26048	0.97%
11	114	459	4.59%	110	442	4.84%	45	180	4.62%	41459	165835	6.18%
12	115	462	4.62%	104	418	4.58%	51	207	5.31%	16410	65639	2.45%
13	118	475	4.75%	100	403	4.42%	51	205	5.26%	49636	198543	7.40%
14	124	498	4.98%	111	445	4.88%	68	272	6.98%	19868	79471	2.96%
15	124	496	4.96%	116	466	5.11%	53	212	5.44%	16245	64980	2.42%
16	118	473	4.73%	113	453	4.96%	43	172	4.41%	22342	89368	3.33%
17	121	484	4.84%	110	442	4.84%	39	159	4.08%	34938	139752	5.21%
18	119	478	4.78%	111	445	4.88%	55	220	5.64%	42202	168807	6.29%
19	123	493	4.93%	113	454	4.97%	44	176	4.52%	17814	71255	2.66%
20	121	486	4.86%	118	475	5.20%	35	143	3.67%	27721	110884	4.13%
21	113	453	4.53%	107	431	4.72%	41	164	4.21%	67899	271598	10.12%
22	86	346	3.46%	74	298	3.27%	34	137	3.51%	54246	216984	8.09%
23	89	356	3.56%	84	337	3.69%	33	134	3.44%	5021	20082	0.75%

3. Top URLs:

Top 30 of 1259 Total URLs					
#	Hits		kB F		URL
1	808	8.08%	2803	0.10%	/favicon.ico
2	575	5.75%	18729	0.70%	/
3	546	5.46%	2534	0.09%	/style2.css
4	538	5.38%	523	0.02%	/reset.css
5	489	4.89%	7109	0.26%	/blog/tags/puppet
6	223	2.23%	2641	0.10%	/projects/xdotool/
7	180	1.80%	0	0.00%	/robots.txt
8	154	1.54%	7194	0.27%	/projects/xdotool/xdotool.xhtml
9	135	1.35%	2482	0.09%	/articles/dynamic-dns-with-dhcp/
10	77	0.77%	1289	0.05%	/blog/geekery/ssl-latency.html
11	60	0.60%	546	0.02%	/blog/geekery/disabling-battery-in-ubuntu-vms.html
12	60	0.60%	946	0.04%	/blog/tags/firefox
13	55	0.55%	889	0.03%	/articles/ssh-security/
14	51	0.51%	536	0.02%	/blog/geekery/solving-good-or-bad-problems.html
15	51	0.51%	1783	0.07%	/presentations/logstash-puppetconf-2012/
16	47	0.47%	23156	0.86%	/images/logstash_OSCON.pdf
17	39	0.39%	341	0.01%	/blog/geekery/installing-windows-8-consumer-preview.html
18	37	0.37%	397	0.01%	/blog/geekery/xvfb-firefox.html
19	37	0.37%	870	0.03%	/presentations/puppet-at-loggly/puppet-at-loggly.pdf.html
20	30	0.30%	545	0.02%	/articles/ppp-over-ssh/
21	30	0.30%	479	0.02%	/projects/keynav/
22	28	0.28%	1218	0.05%	/presentations/logstash-scale11x/
23	26	0.26%	1249	0.05%	/presentations/logstash-1/
24	26	0.26%	457	0.02%	/presentations/logstash-metrics-sf-2012.10/
25	25	0.25%	879	0.03%	/blog
26	25	0.25%	32	0.00%	/presentations/logstash-puppetconf-2012/css/reset.css
27	24	0.24%	1272815	47.44%	/misc/sample.log
28	23	0.23%	286	0.01%	/files/logstash/
29	22	0.22%	339	0.01%	/blog/geekery/debugging-java-performance.html
30	22	0.22%	569	0.02%	/presentations/logstash-puppetconf-2012/css/main.css

4.Entry & Exit Pages

Top 10 of 585 Total Entry Pages					
#	Hits		Visits		URL
1	575	5.75%	509	18.12%	/
2	489	4.89%	323	11.50%	/blog/tags/puppet
3	223	2.23%	206	7.33%	/projects/xdotool/
4	135	1.35%	123	4.38%	/articles/dynamic-dns-with-dhcp/
5	77	0.77%	67	2.39%	/blog/geekery/ssl-latency.html
6	60	0.60%	59	2.10%	/blog/geekery/disabling-battery-in-ubuntu-vms.html
7	51	0.51%	50	1.78%	/presentations/logstash-puppetconf-2012/
8	60	0.60%	43	1.53%	/blog/tags/firefox
9	55	0.55%	39	1.39%	/articles/ssh-security/
10	37	0.37%	35	1.25%	/blog/geekery/xvfb-firefox.html

Top 10 of 587 Total Exit Pages					
#	Hits		Visits		URL
1	575	5.75%	488	17.27%	/
2	489	4.89%	323	11.43%	/blog/tags/puppet
3	223	2.23%	203	7.19%	/projects/xdotool/
4	135	1.35%	125	4.42%	/articles/dynamic-dns-with-dhcp/
5	77	0.77%	66	2.34%	/blog/geekery/ssl-latency.html
6	51	0.51%	50	1.77%	/blog/geekery/solving-good-or-bad-problems.html
7	55	0.55%	49	1.73%	/articles/ssh-security/
8	60	0.60%	48	1.70%	/blog/tags/firefox
9	51	0.51%	39	1.38%	/presentations/logstash-puppetconf-2012/
10	37	0.37%	36	1.27%	/presentations/puppet-at-loggly/puppet-at-loggly.pdf.html

5.Top Sites

Top 30 of 1753 Total Sites									
#	Hits		Files		kB F		Visits		Hostname
1	482	4.82%	420	4.60%	73731	2.75%	250	8.37%	66.249.73.135
2	364	3.64%	364	3.99%	5287	0.20%	220	7.36%	46.105.14.53
3	357	3.57%	288	3.16%	42891	1.60%	10	0.33%	130.237.218.86
4	273	2.73%	93	1.02%	16739	0.62%	10	0.33%	75.97.9.59
5	113	1.13%	113	1.24%	1641	0.06%	94	3.15%	50.16.19.13
6	102	1.02%	102	1.12%	2506	0.09%	81	2.71%	209.85.238.199
7	99	0.99%	95	1.04%	164192	6.12%	63	2.11%	68.180.224.225
8	84	0.84%	56	0.61%	1235	0.05%	36	1.20%	100.43.83.137
9	83	0.83%	82	0.90%	855	0.03%	35	1.17%	208.115.111.72
10	82	0.82%	82	0.90%	804	0.03%	59	1.97%	198.46.149.143
11	74	0.74%	70	0.77%	539	0.02%	36	1.20%	208.115.113.88
12	65	0.65%	65	0.71%	669	0.02%	37	1.24%	108.171.116.194
13	60	0.60%	0	0.00%	19	0.00%	0	0.00%	208.91.156.11
14	60	0.60%	46	0.50%	840	0.03%	34	1.14%	65.55.213.73
15	56	0.56%	33	0.36%	22791	0.85%	30	1.00%	66.249.73.185
16	52	0.52%	52	0.57%	13557	0.51%	1	0.03%	50.139.66.106
17	50	0.50%	48	0.53%	2518	0.09%	5	0.17%	14.160.65.22
18	50	0.50%	50	0.55%	13488	0.50%	1	0.03%	86.76.247.183
19	43	0.43%	43	0.47%	4627	0.17%	2	0.07%	93.17.51.134
20	42	0.42%	41	0.45%	562	0.02%	22	0.74%	208.43.252.200
21	41	0.41%	14	0.15%	256	0.01%	20	0.67%	144.76.194.187
22	41	0.41%	39	0.43%	461	0.02%	10	0.33%	183.179.22.186
23	41	0.41%	14	0.15%	256	0.01%	20	0.67%	199.168.96.66
24	40	0.40%	40	0.44%	3002	0.11%	0	0.00%	209.17.114.78
25	40	0.40%	39	0.43%	2403	0.09%	2	0.07%	210.13.83.18
26	39	0.39%	38	0.42%	2381	0.09%	2	0.07%	115.112.233.75
27	39	0.39%	38	0.42%	2384	0.09%	2	0.07%	59.163.27.11
28	38	0.38%	37	0.41%	423	0.02%	10	0.33%	24.11.96.184
29	38	0.38%	38	0.42%	4266	0.16%	1	0.03%	67.61.65.249
30	37	0.37%	35	0.38%	4683	0.17%	2	0.07%	111.199.235.239

Top 10 of 1753 Total Sites By kB F									
#	Hits		Files		kB F		Visits		Hostname
1	99	0.99%	95	1.04%	164192	6.12%	63	2.11%	68.180.224.225
2	6	0.06%	6	0.07%	159130	5.93%	3	0.10%	94.23.164.135
3	8	0.08%	8	0.09%	107553	4.01%	1	0.03%	190.153.25.242
4	6	0.06%	6	0.07%	106123	3.96%	3	0.10%	100.2.4.116
5	4	0.04%	4	0.04%	106087	3.95%	2	0.07%	88.198.255.242
6	2	0.02%	2	0.02%	106068	3.95%	0	0.00%	184.154.149.126
7	482	4.82%	420	4.60%	73731	2.75%	250	8.37%	66.249.73.135
8	7	0.07%	7	0.08%	67588	2.52%	1	0.03%	117.28.234.67
9	1	0.01%	1	0.01%	63730	2.38%	0	0.00%	82.200.166.110
10	4	0.04%	4	0.04%	53103	1.98%	2	0.07%	192.95.12.193

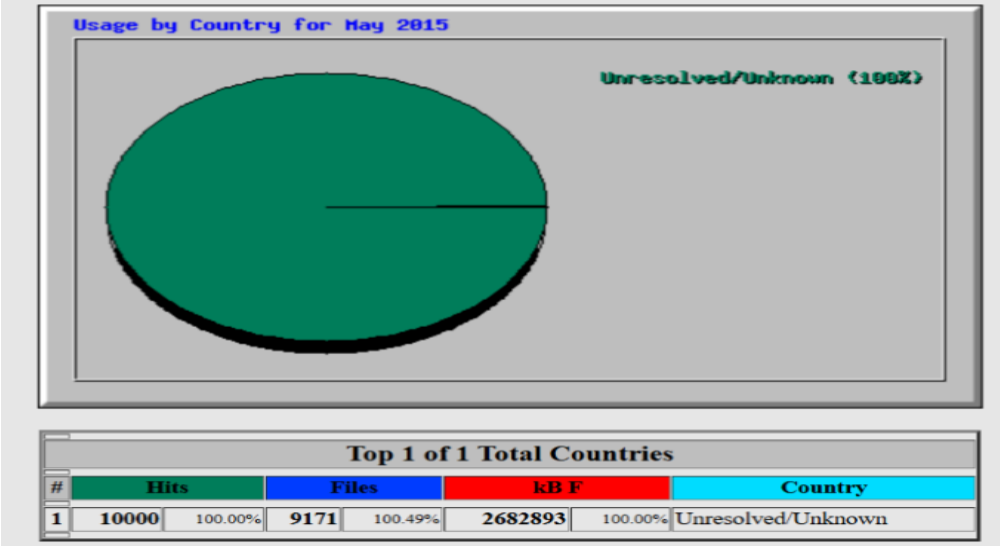
6.Top Referrers

Top 30 of 404 Total Referrers			
#	Hits		Referrer
1	4073	40.73%	- (Direct Request)
2	689	6.89%	http://semicomplete.com/presentations/logstash-puppetconf-2012/
3	658	6.58%	http://www.semicomplete.com/projects/xdotool/
4	406	4.06%	http://semicomplete.com/presentations/logstash-scale11x/
5	335	3.35%	http://www.semicomplete.com/articles/dynamic-dns-with-dhcp/
6	245	2.45%	http://www.semicomplete.com/
7	200	2.00%	http://www.semicomplete.com/style2.css
8	166	1.66%	http://semicomplete.com/
9	148	1.48%	http://semicomplete.com/presentations/logstash-monitorama-2013/
10	144	1.44%	http://www.semicomplete.com/blog/geekery/ssl-latency.html
11	123	1.23%	http://semicomplete.com/presentations/logstash-1/
12	115	1.15%	http://www.semicomplete.com/blog/tags/puppet
13	104	1.04%	https://www.google.com/
14	99	0.99%	http://semicomplete.com/presentations/logstash-metrics-sf-2012.10/
15	78	0.78%	http://www.google.com/url
16	77	0.77%	http://www.semicomplete.com/blog/tags/year review
17	74	0.74%	http://www.semicomplete.com/files/xdotool/docs/html/
18	65	0.65%	http://www.semicomplete.com/blog/geekery/debugging-java-performance.html
19	65	0.65%	http://www.semicomplete.com/projects/keynav/
20	58	0.58%	http://semicomplete.com/presentations/logstash-puppetconf-2013/
21	56	0.56%	http://www.semicomplete.com/articles/ssh-security/
22	53	0.53%	http://www.semicomplete.com/blog/tags/jquery mobile
23	49	0.49%	http://www.semicomplete.com/blog/geekery/xvfb-firefox.html
24	47	0.47%	http://www.semicomplete.com/presentations/logstash-preso-1.0/
25	46	0.46%	http://semicomplete.com/blog/geekery/xvfb-firefox.html
26	45	0.45%	http://www.semicomplete.com/articles/ppp-over-ssh/
27	41	0.41%	http://semicomplete.com/files/logstash/
28	40	0.40%	http://semicomplete.com/presentations/logstash-intro/
29	36	0.36%	http://www.semicomplete.com/files/xdotool/docs/
30	31	0.31%	http://www.google.com/search

7. Agents

Top 15 of 551 Total User Agents			
#	Hits		User Agent
1	1044	10.44%	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36
2	369	3.69%	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.91 Safari/537.36
3	364	3.64%	UniversalFeedParser/4.2-pre-314-svn +http://feedparser.org/
4	296	2.96%	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:27.0) Gecko/20100101 Firefox/27.0
5	271	2.71%	Mozilla/5.0 (iPhone; CPU iPhone OS 6_0 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A5376e Safa
6	268	2.68%	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36
7	237	2.37%	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
8	236	2.36%	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:27.0) Gecko/20100101 Firefox/27.0
9	229	2.29%	Mozilla/5.0 (X11; Linux x86_64; rv:27.0) Gecko/20100101 Firefox/27.0
10	198	1.98%	Tiny Tiny RSS/1.11 (http://tt-rss.org/)
11	175	1.75%	Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36
12	166	1.66%	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:22.0) Gecko/20100101 Firefox/22.0
13	166	1.66%	Mozilla/5.0 (compatible; archive.org_bot +http://www.archive.org/details/archive.org_bot)
14	157	1.57%	Mozilla/5.0 (compatible; Ezooms/1.0; help@moz.com)
15	152	1.52%	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36

8. Countries and Domains



Top 1 of 1 Total Countries							
#	Hits		Files		kB F		Country
1	10000	100.00%	9171	100.49%	2682893	100.00%	Unresolved/Unknown

RESULT: Thus, The Webalizer tool successfully generated detailed web usage statistics, including monthly, daily, and hourly summaries. It displayed comprehensive reports such as top URLs, sites, referrers, user agents, entry and exit pages, countries, and domains.