

Ex No: 1 BASIC NETWORKING COMMANDS AND CONFIGURATION

Date:

AIM:

To study basic networking commands and perform the configuration of a network interface in Linux Operating System.

IFCONFIG COMMAND:

ifconfig (interface configurator) command is used to,

1. View IP Address and Hardware / MAC address assigned to interface, gateway, DNS Server ,MTU (Maximum transmission unit) size and many other network configuration parameters.

```
[root@localhost ~]# ifconfig
```

```
eth0  Link encap:Ethernet HWaddr 00:0C:29:28:FD:4C
      inet addr:192.168.50.2 Bcast:192.168.50.255 Mask:255.255.255.0
      inet6 addr: fe80::20c:29ff:fe28:fd4c/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:6093 errors:0 dropped:0 overruns:0 frame:0
      TX packets:4824 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:6125302 (5.8 MiB) TX bytes:536966 (524.3 KiB)
      Interrupt:18 Base address:0x2000
```

```
lo    Link encap:Local Loopback
      inet addr:127.0.0.1 Mask:255.0.0.0
      inet6 addr: ::1/128 Scope:Host
      UP LOOPBACK RUNNING MTU:16436 Metric:1
      RX packets:8 errors:0 dropped:0 overruns:0 frame:0
      TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:480 (480.0 b) TX bytes:480 (480.0 b)
```

2. Assign IP Address to interface

```
[root@localhost ~]#ifconfig eth0 192.168.50.5 netmask 255.255.255.0
```

```
[root@localhost ~]#ifconfig eth0
```

```
eth0  Link encap:Ethernet HWaddr 00:0C:29:28:FD:4C
      inet addr:192.168.50.5 Bcast:192.168.50.255 Mask:255.255.255.0
      inet6 addr: fe80::20c:29ff:fe28:fd4c/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:6119 errors:0 dropped:0 overruns:0 frame:0
      TX packets:4841 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:6127464 (5.8 MiB) TX bytes:539648 (527.0 KiB)
      Interrupt:18 Base address:0x2000
```

3. Enable or disable interface on demand.

```
[root@localhost ~]#ifdown eth0 – disables an interface named eth0
```

```
[root@localhost ~]#ifup eth0 – Enables an interface named eth0
```

PING COMMAND:

1. PING (Packet INternet Groper) command is the best way to test connectivity between two nodes.
2. Both in Local Area Network (LAN) or Wide Area Network (WAN).
3. Ping use ICMP (Internet Control Message Protocol) to communicate to other devices.
4. #ping hostname(ping localhost)
5. #ping ip address (ping 4.2.2.2)
6. #ping fully qualified domain name(ping www.facebook.com)

```
[root@localhost ~]#ping 4.2.2.2
```

```
PING 4.2.2.2 (4.2.2.2) 56(84) bytes of data.
```

```
64 bytes from 4.2.2.2: icmp_seq=1 ttl=44 time=203 ms
```

```
64 bytes from 4.2.2.2: icmp_seq=2 ttl=44 time=201 ms
```

```
64 bytes from 4.2.2.2: icmp_seq=3 ttl=44 time=201 ms
```

HOSTNAME COMMAND:

This command displays the network name of the host machine.

```
[root@localhost network-scripts]# hostname
```

```
localhost.localdomain
```

```
[root@localhost network-scripts]#
```

TRACEROUTE COMMAND:

Traceroute is a command which can shows the path a packet takes from the computer to one specified as destination. It will list all the routers it passes through until it reaches its destination, or fails to and is discarded. In addition to this, it will tell how long each 'hop' from router to router takes.

```
[root@localhost network-scripts]# traceroute www.google.com
```

```
traceroute to www.google.com (172.217.163.132), 30 hops max, 60 byte packets
```

```
1 gateway (172.16.8.1) 0.165 ms 0.131 ms 0.119 ms
```

```
2 220.225.219.38 (220.225.219.38) 7.131 ms 7.133 ms 7.472 ms
```

```
3 * * *
```

```
4 220.227.42.241 (220.227.42.241) 6.951 ms 7.100 ms 7.323 ms
```

```
5 80.81.65.93 (80.81.65.93) 6.869 ms 6.826 ms 6.962 ms
```

```
6 ae10.0.cjr01.sin001.flagtel.com (62.216.128.233) 39.441 ms 221.191 ms 40.260 ms
```

```
7 80.77.1.254 (80.77.1.254) 40.248 ms 38.916 ms 38.304 ms
```

```
8 108.170.240.164 (108.170.240.164) 39.610 ms 108.170.240.242
```

```
(108.170.240.242) 38.963 ms 108.170.240.172 (108.170.240.172) 38.860 ms
```

```
9 72.14.235.152 (72.14.235.152) 39.912 ms * 72.14.234.96 (72.14.234.96) 38.783 ms
```

```

10 216.239.48.226 (216.239.48.226) 122.220 ms 72.14.233.122
(72.14.233.122) 121.132 ms 216.239.48.226 (216.239.48.226) 124.187 ms
(216.239.43.77) 125.901 ms *
11 maa05s04-in-f4.1e100.net (172.217.163.132) 27.956 ms * *
[root@localhost network-scripts]#

```

ROUTE COMMAND:

Route command is used to show/manipulate the IP routing table. It is primarily used to setup static routes to specific host or networks via an interface.

```

[root@localhost network-scripts]# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default gateway 0.0.0.0 UG 100 0 0 enp3s0
172.16.8.0 0.0.0.0 255.255.252.0 U 100 0 0 enp3s0
[root@localhost network-scripts]#

```

NETSTAT COMMAND:

Netstat (network statistics) is a command line tool for monitoring network connections both incoming and outgoing as well as viewing routing tables, interface statistics etc.

```

[root@localhost network-scripts]# netstat -r
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
default gateway 0.0.0.0 UG 0 0 0 enp3s0
172.16.8.0 0.0.0.0 255.255.252.0 U 0 0 0 enp3s0
[root@localhost network-scripts]#

```

NSLOOKUP COMMAND:

nslookup command. nslookup (name server lookup) is a tool used to perform DNS lookups inLinux. It is used to display DNS details, such as the IP address of a particular computer, the MX records for a domain or the NS servers of a domain. nslookup can operate in two modes: interactive and non-interactive.

```

[root@localhost network-scripts]# nslookup www.google.com
Server: 172.16.8.1
Address: 172.16.8.1
Non-authoritative answer:
Name: www.google.com
Address: 172.217.163.132
Name: www.google.com
Address: 2404:6800:4007:80e::2004
[root@localhost network-scripts]#

```

ARP COMMAND:

Arp command is used to display the arp cache table

```
[root@localhost network-scripts]# arp
Address          HWtype HWaddress     Flags Mask      Iface
172.16.9.82      ether  ec:a8:6b:69:d3:e1  C          enp3s0
172.16.10.118    ether  00:11:5b:ff:cc:a5  C          enp3s0
gateway          ether  08:35:71:f2:b4:a1  C          enp3s0
172.16.10.91     ether  00:e0:4c:b2:5b:06  C          enp3s0
172.16.10.124    ether  00:0f:ea:93:f4:55  C          enp3s0
172.16.8.51      ether  00:1a:4d:a6:98:30  C          enp3s0
[root@localhost network-scripts]# ^C
```

WHOIS COMMAND:

It's queries an official Internet database to determine the current owner of a network domain name or host name

```
[root@localhost network-scripts]# whois google.com
[Querying whois.verisign-grs.com]
[Redirected to whois.markmonitor.com]
[Querying whois.markmonitor.com]
[whois.markmonitor.com]
Domain Name: google.com
Registry Domain ID: 2138514_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.markmonitor.com
Registrar URL: http://www.markmonitor.com
Updated Date: 2018-02-21T10:45:07-0800
Creation Date: 1997-09-15T00:00:00-0700
```

NETWORK CONFIGURATION IN LINUX

1. **Open the network interface configuration file of your computer and write the values of the following configuration parameters**

Output:

```
a) IPADDR           : 172.16.8.138
b) HWADDR           : 00:27:0e:13:ea:6a
c) PREFIX/SUBNET MASK : 23
d) GATEWAY           : 172.16.8.1
e) DNS               : 172.16.8.1
f) NAME              : New 302-3-ethernet connection
g) TYPE              : Ethernet
```

2. **Display the network configuration details of your network interface.**

Output:

```
TYPE = Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO="static"
IPADDR=172.16.8.13
PREFIX=23
GATEWAY=172.16.8.1
DNS=172.16.8.1
DEFROUTE=yes
IPV4_FAILURE_FATAL=yes
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=Stable_privacy
NAME= New 302-3-ethernet connection
UUID=C7628055-dbb6-470f-a68-c71dafdefa
ONBOOT=yes
```

3. **Assign the following IP address to your device 172.16.8.127 and display the change in IP address.**

Output:

To change the IP address:

```
[root@localhost network-scripts]#ifconfig eth0 172.16.8.127 netmask 255.255.255.0
```

To verify the Change:

```
[root@localhost network-scripts]# ifconfig
eth0  Link encap:Ethernet  HWaddr 00:0C:29:28:FD:4C
      inet addr:192.168.50.5  Bcast:172.16.8.127  Mask:255.255.255.0
      inet6 addr: fe80::20c:29ff:fe28:fd4c/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:6119 errors:0 dropped:0 overruns:0 frame:0
      TX packets:4841 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:6127464 (5.8 MiB)  TX bytes:539648 (527.0 KiB)
      Interrupt:18 Base address:0x2000
```

4. **Disconnect the network interface and check the dis-connectivity through a command and display the results. Finally activate the interface.**

Output:

To Disconnect the Interface:

```
[root@localhost network-scripts]#ifdown enp3so
Device 'enp3so' successfully disconnected.
(Check for a internet connectivity. It will not happen)
```

To connect to the Interface:

```
[root@localhost network-scripts]#ifup enp3so
Connection successfully activated.
```

5. **Check whether the following hosts are reachable from your device and explain the response.**

Output:

a) 172.16.8.2

```
[root@localhost network-scripts]#ping 172.16.8.2
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=3.14ms
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=2.12ms
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=3.44ms
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=1.14ms
172.16.8.2 is reachable.
```

b)DNS Server

```
[root@localhost network-scripts]#ping 172.16.8.1
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=3.14ms
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=2.12ms
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=3.44ms
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=1.14ms
DNS is reachable.
```

c) GATEWAY

```
[root@localhost network-scripts]#ping 172.16.8.1
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=3.14ms
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=2.12ms
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=3.44ms
64 bytes from 172.16.8.2:icmp_seq=1 ttl=64 time=1.14ms
GATEWAY is reachable.
```

6. **Disconnect the network interface and Check whether the following hosts are reachable from your device and explain the response.**

Output:

```
[root@localhost network-scripts]#ifdown enp3so
Device 'enp3so' successfully disconnected.
```

a)127.0.0.1

```
[root@localhost network-scripts]#ping 127.0.0.1
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.014ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.012ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.044ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.014ms
127.0.0.1 is reachable because it is loopback address.
```

b)GATEWAY

```
[root@localhost network-scripts]#ping 172.16.8.1
GATEWAY is unreachable.
```

7. **Find the IPv4 and IPv6 addresses of the following URLs.**

Output:

a)www.google.com

```
nslookup www.google.com
IPV4 172.217.163.68
IPV6 2404:6800:4007:80c::2004
```

b)www.facebook.com

```
nslookup www.facebook.com
IPV4 157.240.24.35
IPV6 2a03:2880:f139:83:face:booc:0:25de
```

c)www.rajalakshmi.org

```
nslookup www.rajalakshmi.org
IPV4 220.227.30.5
```

8. **Display the ARP table of your machine.**

Output:

```
[root@localhost network-scripts]#arp
```

Address	HWType	HWaddress	Flags	Mark	Iface
172.16.8.1	ether	00:27:0e:13:ea:6a	c		enp3so
172.16.8.2	ether	00:27:0e:13:ea:6a	c		enp3so

9. **Display the kernel routing table of your machine.**

Output:

```
[root@localhost network-scripts]#route
```

Kernel IP routing table

Destination	Gateway	Genmask	Flags	metric	Ref	Iface
Default	gateway	0.0.0.0	ua	100	0	U enp3s0
172.16.0.0	0.0.0.0	255.255.0.0	U	100	0	O enp3s0

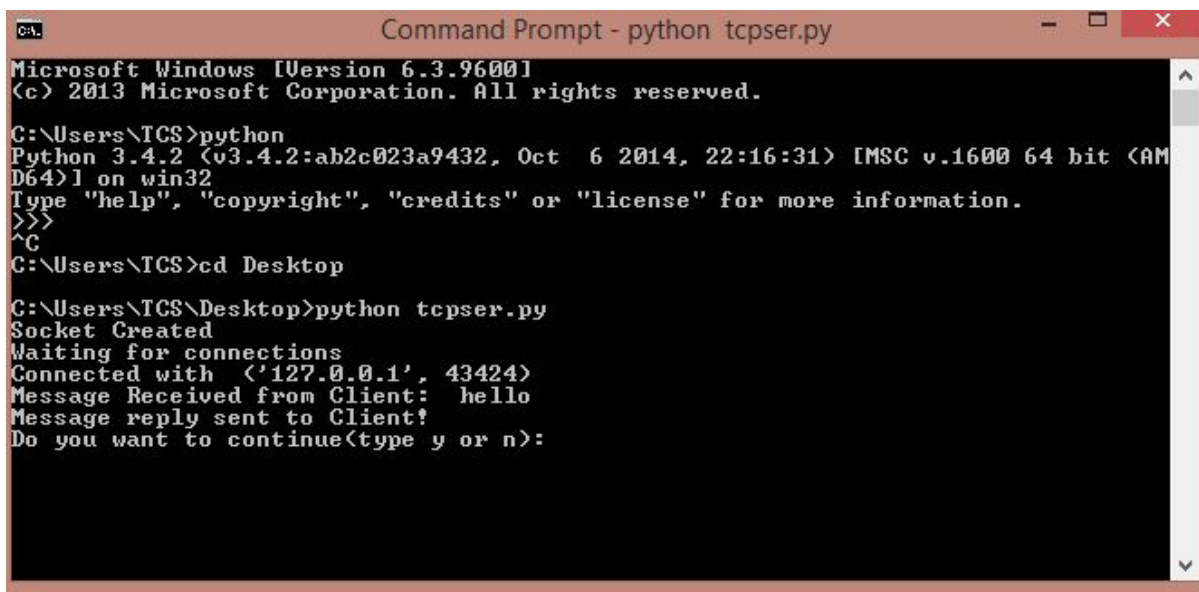
10. List out the hops(IP addresses of the intermediate nodes) taken by a packet to reach its destination.

Output:

[root@localhost network-scripts]#traceroute www.google.com

Traceroute to www.google.com (172.217.163.68) 30 hops.60 byte packets

OUTPUT



```

C:\Users\TCS>python
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
^C
C:\Users\TCS>cd Desktop
C:\Users\TCS\Desktop>python tcpser.py
Socket Created
Waiting for connections
Connected with ('127.0.0.1', 43424)
Message Received from Client:  hello
Message reply sent to Client!
Do you want to continue(type y or n):

```



```

C:\Users\TCS>cd Desktop
C:\Users\TCS\Desktop>python tcpcli.py
Enter your message:hello
Message Received from Server:  hello
C:\Users\TCS\Desktop>

```

RESULT:

Thus the program for network configuration in linux is executed successfully.

Ex. No:2a STUDY OF SOCKET PROGRAMMING IN PYTHON**Date:****AIM:**

To study the concepts of 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:

Instance 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 <i>backlog</i> 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

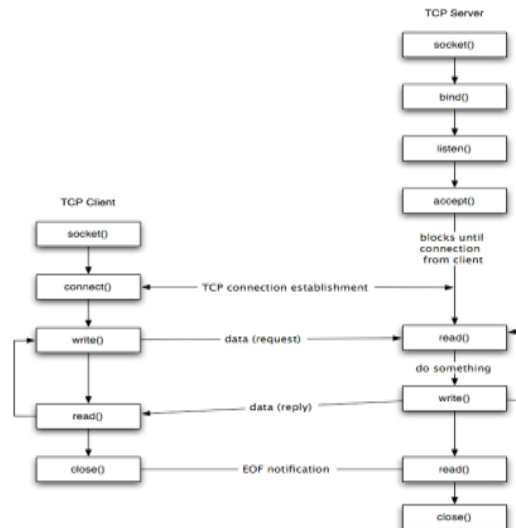


Figure 1. TCP Socket Programming

HOW TO WORK WITH UDP SOCKETS IN PYTHON:

UDP: no “connection” between client & server

- ❖ no handshaking before sending data
- ❖ sender explicitly attaches IP destination address and port # to each packet
- ❖ rcvr extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- ❖ UDP provides *unreliable* transfer of groups of bytes (“datagrams”) between client and server

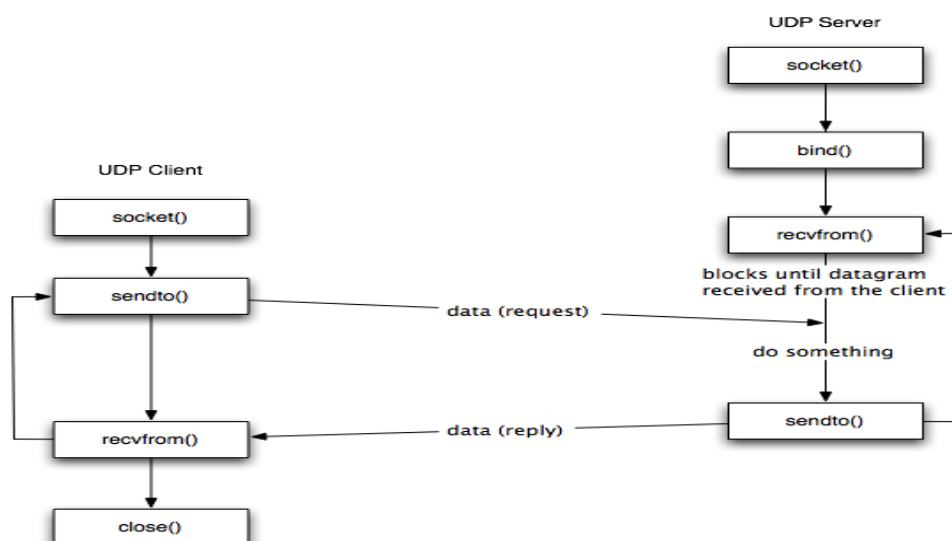


Figure 2. UDP Socket Programming

IMPLEMENTATION OF ECHO CLIENT/SERVER APPLICATION USING TCP:

Server code:

```
from socket import *

server_socket = socket(AF_INET, SOCK_DGRAM)

server_socket.bind(('127.0.0.1', 12000))

print("UDP server is listening")

while True:
    message, address = server_socket.recvfrom(1024)
    print(message)
    print(address)

    server_socket.sendto("This is UDP".encode('utf-8'), address)
```

Client code:

```
from socket import *

client_socket = socket(AF_INET, SOCK_DGRAM)
message = "This is User Datagram Protocol"

client_socket.sendto(message.encode('utf-8'), ("127.0.0.1", 12000))

data, address = client_socket.recvfrom(1024)

print("Message from server : {}".format(data))
```

OUTPUT

Server.py

```
>>>
===== RESTART: D:/Study/SEM 5 Notes/CN/LAB/LAB EXP PGM/Server.py =====
UDP server is listening
b'This is User Datagram Protocol'
('192.168.43.111', 54759)

===== RESTART: D:/Study/SEM 5 Notes/CN/LAB/LAB EXP PGM/Server.py =====
UDP server is listening
b'This is User Datagram Protocol'
('192.168.43.111', 50564)
Dinesh S 200701502
```

Client.py

```
>>>
===== RESTART: D:/Study/SEM 5 Notes/CN/LAB/LAB EXP PGM/Exp no 2/Client.py =====
Message from server : b'This is UDP'
>>> DINESH S 200701502
```

RESULT:

Thus, the implementation of an echo using sockets in TCP and UDP has been written, executed and the output was verified successfully.

Ex No:2b**CHAT PROGRAM USING SOCKETS****Date:****Aim :**

To implement chat program using sockets

Algorithm :**a) Server Side :**

- i. Import Socket and Time module in Python.
- ii. Use gethostname() and gethostbyname() to retrieve the hostname of the machine and translate host name to IPV4 format address.
- iii. Use bind() function to bind the socket to a given address.
- iv. Use listen() function to accept connection requests from clients.
- v. Use accept() function to wait and accept connection from clients.
- vi. Use the recv() function to retrieve messages through TCP.
- vii. Then decoding is done by decode() function.
- viii. Open a while loop. Get the message as input.
- ix. If input is '[e]' then exit the chat room.
- x. Else, send the message to the client and display it.
- xi. Display the received and sent messages.

b) Client Side :

- i. Import Socket and Time module in Python.
- ii. Get the hostname and IP address of the host.
- iii. Enter the address of the server to chat with.
- iv. Connect to that server socket using socket() function.
- v. Send and receive messages using send() and recv() functions.
- vi. Open a while loop. Print the messages received from the server.
- vii. If the message is '[e]', then leave the server and exit the connection.
- viii. Else, display the received and sent messages.

PROGRAM**Client.py**

```
import time, socket, sys
```

```
print("\nWelcome to Chat Room\n")
```

```
print("Initialising....\n")
```

```
time.sleep(1)
```

```
s = socket.socket()
```

```
shost = socket.gethostname()
```

```
ip = socket.gethostbyname(shost)
```

```
print(shost, "(", ip, ")\n")
host = input(str("Enter server address: "))
name = input(str("\nEnter your name: "))
port = 1234
print("\nTrying to connect to ", host, "(", port, ")\n")
time.sleep(1)
s.connect((host, port))
print("Connected...\n")

s.send(name.encode())
s_name = s.recv(1024)
s_name = s_name.decode()
print(s_name, "has joined the chat room\nEnter [e] to exit chat room\n")

while True:
    message = s.recv(1024)
    message = message.decode()
    print(s_name, ":", message)
    message = input(str("Me : "))
    if message == "[e]":
        message = "Left chat room!"
        s.send(message.encode())
        print("\n")
        break
    s.send(message.encode())
```

Server.py

```
import time, socket, sys

print("\nWelcome to Chat Room\n")
print("Initialising...\n")
time.sleep(1)

s = socket.socket()
host = socket.gethostname()
ip = socket.gethostbyname(host)
port = 1234
s.bind((host, port))
print(host, "(", ip, ")\n")
name = input(str("Enter your name: "))

s.listen(1)
print("\nWaiting for incoming connections...\n")
conn, addr = s.accept()
print("Received connection from ", addr[0], "(", addr[1], ")\n")

s_name = conn.recv(1024)
s_name = s_name.decode()
print(s_name, "has connected to the chat room\nEnter [e] to exit chat room\n")
conn.send(name.encode())

while True:
    message = input(str("Me : "))
    if message == "[e]":
        message = "Left chat room!"
        conn.send(message.encode())
```



```

        print("\n")
        break
    conn.send(message.encode())
    message = conn.recv(1024)
    message = message.decode()
    print(s_name, ":", message)

```

OUTPUT

Server.py

```

Welcome to Chat Room

Initialising....

Dinesh ( 192.168.43.111 )

Enter your name: Dinesh

Waiting for incoming connections...

Received connection from 192.168.43.111 ( 52252 )

Dinesh has connected to the chat room
Enter [e] to exit chat room

Me : hi
Dinesh : How are you da
Me : Fine da What about you
Dinesh : Fine da. did you watch PS 1 movie da
Me : no da no tickets available
Dinesh : Haha i watched da super movie da
Me : ok shut and leave da bodysoda
Dinesh : HAHA!!! ,Bye
Me : Bye
Dinesh : _J

```

Client.py

```
Welcome to Chat Room

Initialising....

Dinesh ( 192.168.43.111 )

Enter server address: 192.168.43.111

Enter your name: Dinesh

Trying to connect to 192.168.43.111 ( 1234 )

Connected...

Dinesh has joined the chat room
Enter [e] to exit chat room

Dinesh : hi
Me : How are you da
Dinesh : Fine da What about you
Me : Fine da. did you watch PS 1 movie da
Dinesh : no da no tickets available
Me : Haha i watched da super movie da
Dinesh : ok shut and leave da bodysoda
Me : HAHA!!! ,Bye
Dinesh : Bye
```

RESULT

Thus the chat program using sockets is successfully executed

Ex No: 2c UPPER CASE CONVERTOR USING UDP SOCKET**Date:****AIM:**

To convert uppercase to lowercase using UDP socket.

ALGORITHM:**UDP Server :**

1. Create a UDP socket.
2. Bind the socket to the server address.
3. Wait until the datagram packet arrives from the client.
4. Process the datagram packet and send a reply to the client.
5. Go back to Step 3.

UDP Client :

1. Create a UDP socket.
2. Send a message to the server.
3. Wait until response from the server is received.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

After Establishing a Socket on both client and server side

1. The client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts the characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen

PROGRAM:

```
import socket
```

```
localIP = "192.168.43.111"
```

```
localPort = 8080
```

```
bufferSize = 1024
```

```
msgFromServer = "Hello UDP Client"
```

```
#bytesToSend = str.encode(msgFromServer)
```

```
# Create a datagram socket
```

```
UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
```

```
# Bind to address and ip
```

```
UDPServerSocket.bind((localIP, localPort))
```

```
print("UDP server up and listening")
```

```
# Listen for incoming datagrams
```

```
while(1):
```

```
    #message,bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
    message,address = UDPServerSocket.recvfrom(bufferSize)
```

```
    #message = bytesAddressPair[0]
```

```
    #address = bytesAddressPair[1]
```

```
    clientMsg = f'Message from Client:{message}'
    clientIP = f'Client IP Address:{address}'
```

```
    print(clientMsg)
    print(clientIP)
    bytesToSend = str.encode(clientMsg.upper())
```

```
# Sending a reply to client
```

```
UDPServerSocket.sendto(bytesToSend, address)
```

Client Side

```
import socket

msgFromClient    = input('Enter a word/sentence in lower case: ')

bytesToSend      = str.encode(msgFromClient)

serverAddressPort = ("192.168.43.111", 8080)

bufferSize       = 1024

# Create a UDP socket at client side

UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Send to server using created UDP socket

UDPClientSocket.sendto(bytesToSend, serverAddressPort)

msgFromServer = UDPClientSocket.recvfrom(bufferSize)

msg = "Message from Server {}".format(msgFromServer[0])

print(msg)
```

OUTPUT:

Server.py

```
===== RESTART: D:/Study/SEM 5 Notes/CN/LAB/LAB EXP PGM/Ex np 2 c/Server.py =====
=
UDP server up and listening
|
```

Client.py

```
Enter a word/sentence in lower case: dinesh
DINESH
```

RESULT :

Thus uppercase alphabets were converted to lowercase using socket programming in python and the output has been verified.

Ex No:2d CALCULATOR APPLICATION USING TCP SOCKET**Date:****AIM**

To implement a calculator application using TCP socket.

ALGORITHM**Server Side:**

- Create a socket using socket() system call.
- Bind server's address and port using bind() system call.
- Convert the socket into a listening socket using listen() system call.
- Wait for client connection to complete using accept() system call.
- Receive the Client request using recv() system call which consists of the input and operation to be performed.
- The calculation is performed and the result is passed back to the client by the server.

Client Side:

- Create a socket.
- Fill in the internet socket address structure (with server information).
- Connect to the server using the connect system call.
- The client passes the operator and input numbers to the server.
- Read the result sent by the server, write it to standard output.
- Close the socket connection.

PROGRAM**Server side**

```
# Import socket module
import socket

# Here we use localhost ip address
# and port number
LOCALHOST = "127.0.0.1"
PORT = 8080
# calling server socket method
server = socket.socket(socket.AF_INET,
                        socket.SOCK_STREAM)

server.bind((LOCALHOST, PORT))
server.listen(1)
print("Server started")
print("Waiting for client request..")
# Here server socket is ready for
# get input from the user
clientConnection, clientAddress = server.accept()
print("Connected client :", clientAddress)
msg = ""
# Running infinite loop
while True:
    data = clientConnection.recv(1024)
```

```

msg = data.decode()
if msg == 'Over':
    print("Connection is Over")
    break

print("Equation is recieved")
result = 0
operation_list = msg.split()
oprnd1 = operation_list[0]
operation = operation_list[1]
oprnd2 = operation_list[2]

# here we change str to int converstion
num1 = int(oprnd1)
num2 = int(oprnd2)
# Here we are perform basic arithmetic operation
if operation == "+":
    result = num1 + num2
elif operation == "-":
    result = num1 - num2
elif operation == "/":
    result = num1 / num2
elif operation == "*":
    result = num1 * num2

print("Send the result to client")
# Here we change int to string and
# after encode send the output to client
output = str(result)
clientConnection.send(output.encode())
clientConnection.close()

```

Client side

```

# Import socket module
import socket

# In this Line we define our local host
# address with port number
SERVER = "127.0.0.1"
PORT = 8080
# Making a socket instance
client = (socket.socket(socket.AF_INET,socket.SOCK_STREAM))
# connect to the server
client.connect((SERVER, PORT))
# Running a infinite loop
while True:
    print("Example : 4 + 5")
    # here we get the input from the user

```

```

inp = input("Enter the operation in \
the form opreand operator oprenad: ")
# If user wants to terminate
# the server connection he can type Over
if inp == "Over":
    break
# Here we send the user input
# to server socket by send Method
client.send(inp.encode())

# Here we received output from the server socket
answer = client.recv(1024)
print("Answer is "+answer.decode())
print("Type 'Over' to terminate")

client.close()

```

OUTPUT

Server.py

```

===== RESTART: D:/Study/SEM 5 Notes/CN/LAB/LAB EXP PGM/Ex no 2d/Server.py =====
Server started
Waiting for client request..
|

```

Client.py

```

>> |
>> |
>> | 4+5
>> | 9
>> |

```

RESULT

Thus a calculator application between client and server using socket programming in python is executed and the output has been verified

Ex No:2e

FILE TRANSFER USING SOCKET PROGRAMMING

Date:

AIM:

To create file transfer applications using socket programming in python.

PROCEDURE:

Server Side:

- Import socket package
- Set the port number
- Set host variable to get hostname by gethostname() method.
- s.bind method binds host and port address.
- Set listen value to 5
- accept method is used to accept connection from server.
- Get the data using the recv method.
- Send the connection using the send method.
- Close the connection.

Client Side:

- Import socket package
- Call the socket function and store it in a variable.
- Get host variable to get hostname by gethostname() method.
- Set the port address.
- Connect to the server and send the message.
- Receive the data using the recv method.
- Close the connection.

Program:

Client

```
import socket          # Import socket module
s = socket.socket()     # Create a socket object
host = socket.gethostname() # Get local machine name
port = 60000           # Reserve a port for your service.
```

```
s.connect((host, port))
s.send("Hello server!".encode('utf-8'))
with open('received_file.txt', 'wb') as f:
    print('file opened')
    while True:
        print('receiving data...')
        data = s.recv(1024)
        print('data :', (data))
        if not data:
            break
        # write data to a file
        f.write(data)
f.close()
```

```
print('Successfully get the file')
s.close()
print('connection closed')
```

Server

```
import socket          # Import socket module
port = 60000           # Reserve a port for your service.
s = socket.socket()     # Create a socket object
host = socket.gethostname() # Get local machine name
s.bind((host, port))    # Bind to the port
s.listen(5)            # Now wait for client connection.
print('Server listening....')
while True:
    conn, addr = s.accept() # Establish connection with client.
    print('Got connection from', addr)
    data = conn.recv(1024)
    print('Server received', repr(data))
    filename='file.txt'
    f = open(filename,'rb')
    l = f.read(1024)
    while (l):
        conn.send(l)
        print('Sent ',repr(l))
        l = f.read(1024)
    f.close()
    print('Done sending')
    # conn.send('Thank you for connecting'.encode('utf-8'))
    conn.close()
```

OUTPUT:

Server.py

```
>>>
===== RESTART: D:\Study\SEM 5 Notes\CN\LAB\LAB EXP PGM\Ex no 2 e\Server.py =====
Server listening....
Got connection from ('192.168.43.111', 60164)
Server received b'Hello server!'
```

Client.py

```
>
===== RESTART: D:\Study\SEM 5 Notes\CN\LAB\LAB EXP PGM\Ex no 2 e\Client.py =====
file opened
receiving data...
```

RESULT:

Thus transfer of files between client and server using socket programming in python is executed and the output has been verified.

Ex No:2f CONNECTING AND PING A SERVER USING SOCKETS**Date:****AIM :**

To connect and ping a Server using Sockets

ALGORITHM :

- Import Socket module in Python.
- Create a socket object.
- Set the default port number of the socket to 80.
- Use gethostname() to retrieve the hostname of the server and translate host name to IPV4 format address.
- Use connect() function to connect to the server by specifying host IP and port number.
- Print the host Ip and Port number and also print "Connection is Successful".

PROGRAM:**Client code:**

```
from socket import *
from os import system
s = socket(AF_INET, SOCK_STREAM)
s.connect(("127.0.0.1",8000)) # Connect
op='connect'
s.send(op.encode('utf-8')) # Send request
data = s.recv(100).decode()# Get response
print(data)
system("ping "+ gethostname())
s.close()
```

#Server Code:

```

from socket import *
from os import system
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",8000))
s.listen(5)
while True:
    c,a = s.accept()
    print("Received connection from", a)
    data=c.recv(100).decode()
    print(data)
    c.send(data.encode('utf-8'))
result = "ping" + str(a)
system(result)
c.close()

```

OUTPUT:**Server.py**

```

===== RESTART: D:\Study\SEM 5 Notes\CN\LAB\LAI
Socket successfully created
socket binded to 12345
socket is listening
Got connection from ('192.168.43.111', 52617)

```

Client.py

```

>>> Thank you for connecting

```

Result :

Thus, a program to connect and ping a Server using Sockets has been written, executed and the output was verified successfully.

Ex No: 3 IMPLEMENTATION OF PACKET SNIFFING USING RAW SOCKET**Date****AIM:**

To study packet sniffing concept and implement it using raw sockets.

PROGRAM

```
import socket
import struct
import binascii

s= socket.socket(socket.AF_INET,socket.SOCK_RAW,socket.IPPROTO_IP)

s.bind(("127.0.0.1",0))

packet=s.recvfrom(65565)

print(packet)

ethernet_header = packet[0][0:14]

eth_header = struct.unpack("!6s6s2s", ethernet_header)

print("ETHERNET HEADER")

print("*****")

print("Destination Address")

print( binascii.hexlify(eth_header[0]))

print("Source Address")

print( binascii.hexlify(eth_header[1]))

print("Type")

print( binascii.hexlify(eth_header[2]))

print("IP HEADER")

print("*****")

ipheader = packet[0][14:34]

ip_header = struct.unpack("!12s4s4s", ipheader)

print("Destination Address")

print (socket.inet_ntoa(ip_header[1]))
```

```
print("Source Address")
print(socket.inet_ntoa(ip_header[2]))
```

OUTPUT:

```
>>> (b'\x00\x04\x00\x87\x00\x80\x06\x00\x00\x7f\x00\x01\x7f\x00\x00\x0d_\x17a\x95\x08f\x00\x00\x00\x02\xff\x8b\xad\x00\x02\x04\xff\xd7\x01\x03\x08\x01\x04\x02', {'127.0.0.1', 0})
... ETHERNET HEADER
... *****
... Destination Address
... b'45000340087'
... Source Address
... b'400080060000'
... Type
... b'7f00'
... IP HEADER
... *****
... Destination Address
... 102.115.0.0
... Source Address
... 0.0.128.2
... 
```

RESULT:

Thus a study on packet sniffing concept and implement it using raw sockets was done

Ex No : 4 a**STUDY OF REMOTE PROCEDURE CALL- XMLRPC****Date****AIM:**

To study the concepts of Remote Procedure Call-XML RPC.

ALGORITHM

1. The client calls the client stub. The call is a local procedure call, with parameters pushed on to the stack in the normal way.
2. The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called marshaling.
3. The client's local operating system sends the message from the client machine to the server machine.
4. The local operating system on the server machine passes the incoming packets to the server stub.
5. The server stub unpacks the parameters from the message. Unpacking the parameters is called unmarshalling.
6. Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

PROGRAM**XML RPC PROGRAM- SERVER SIDE:**

```
from xmlrpc.server import SimpleXMLRPCServer
```

```
def is_even(n):
```

```
    return n % 2 == 0
```

```
def add(a,b):
```

```
    return a+b
```

```
def sub(a,b):
```

```
    return a-b
```

```
def factorial(n):
```

```
    factorial=1
```

```
    for i in range(1,n+1):
```

```
        factorial = factorial*i
```

```
    return factorial
```

```
def multiply(x, y):
```

```
    return x * y
```

```
def divide(x, y):
```

```
    return x // y
```

```
server = SimpleXMLRPCServer(("localhost", 8000))
```

```
print("Listening on port 8000...")
```

```
server.register_function(is_even, "is_even")
```

```
server.register_function(add, "add")
```

```
server.register_function(sub, "sub")
```

```
server.register_function(factorial, "factorial")
```

```
#server.register_function(factorial,"factorial")
server.register_function(multiply, 'multiply')
server.register_function(divide, 'divide')
```

```
server.serve_forever()
```

XML RPC PROGRAM- CLIENT SIDE:

```
import xmlrpc.client
proxy= xmlrpc.client.ServerProxy('http://localhost:8000/')
for i in range(5):
    a=int(input("Enter a number:"))
    b=int(input("Enter b number:"))
    print("%d is even?: %d" % (a, (proxy.is_even(a)))) #access XML-RPC server through proxy
    print("addition of given number is %d "%((proxy.add(a,b))))
    print("sub of given number is %d "%((proxy.sub(a,b))))
    print("factorial: %d" %((proxy.factorial(a))))
    print("factorial: %d" %((proxy.factorial(b))))
    print("Multiplication of 2 numbers is %d "%(proxy.multiply(a,b)))
    print("Division of 2 numbers is %d "%(proxy.divide(a,b)))
```

OUTPUT

Server.py

```
>>>
===== RESTART: D:/Study/SEM 5 Notes/CN/LAB/LAB EXP PGM/Ex no 4 a/Server.py =====
Listening on port 8000...
```

Client.py

```
===== RESTART: D:/Study/SEM 5 Notes/CN/LAB/LAB EXP PGM/Ex no 4 a/Client
Enter a number:2
Enter b number:4

2+4
6
```

RESULT

Thus the program for study of remote procedure call- XMLRPC is executed successfully

EX NO 4 b REMOTE PROCEDURE CALL FOR LIST OPERATIONS- XMLRPC

Date:

AIM:

To Implement an XML RPC code for the following functions,

- a. No of items in a list
- b. Smallest element in a list
- c. Largest element in the list
- d. Converting a list to a set.

ALGORITHM:

1. The client calls the client stub. The call is a local procedure call, with parameters pushed on to the stack in the normal way.
2. The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called marshaling.
3. The client's local operating system sends the message from the client machine to the server machine.
4. The local operating system on the server machine passes the incoming packets to the server stub.
5. The server stub unpacks the parameters from the message. Unpacking the parameters is called unmarshalling.
6. Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

PROGRAM

Server Side:

```
from xmlrpc.server import SimpleXMLRPCServer
def list_length(a):
    return len(a)
def list_maximum(a):
    return max(a)
def list_minimum(a):
    return min(a)
def list_to_set(a):
    f=list(set(a))
    return f
def list_concate(a,b):
    return a+b

server = SimpleXMLRPCServer(("localhost", 8000))
print("Listening on port 8000...")
server.register_function(list_length,"list_length")
server.register_function(list_maximum, "list_maximum")
server.register_function(list_minimum, "list_minimum")
```

```

server.register_function(list_to_set, "list_to_set")
server.register_function(list_concat, "list_concat")
server.serve_forever()

```

Client Side:

```

import xmlrpc.client
proxy= xmlrpc.client.ServerProxy('http://localhost:8000/')
while True:
    print("PRESS 1-->STRAT || 2--> STOP ")
    c=int(input("ENTER YOUR CHOICE"))
    a=[]
    b=[]
    if c==1:
        print("ENTER THE ELEMENTS TO ADD FIRST LIST")
        print("PRESS -1 TO EXIT THIS LIST")
        while True:
            d=int(input("--->"))
            if d== -1:
                break
            a.append(d)
        print("ENTER THE ELEMENTS TO ADD SECOND LIST")
        print("PRESS -2 TO EXIT THIS LIST")
        while True:
            e=int(input("--->"))
            if e== -2:
                break
            b.append(e)
    if c==2:
        break
    print(a)
    print(b)
    print("list_length",proxy.list_length(a))
    print("list_maximum",proxy.list_maximum(a))
    print("list_minimum",proxy.list_minimum(a))
    print("list_to_set",proxy.list_to_set(a))
    print("list_concat",proxy.list_concat(a,b))

```

OUTPUT

Server.py

```
Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (tags/v3.5.0:9cf6752, Oct 5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/Study/SEM 5 Notes/CN/LAB/LAB EXP PGM/Ex no 4 b/Server.py =====
Listening on port 8000...
127.0.0.1 -- [10/Oct/2022 05:38:23] "POST / HTTP/1.1" 200 -
127.0.0.1 -- [10/Oct/2022 05:38:25] "POST / HTTP/1.1" 200 -
127.0.0.1 -- [10/Oct/2022 05:38:27] "POST / HTTP/1.1" 200 -
127.0.0.1 -- [10/Oct/2022 05:38:29] "POST / HTTP/1.1" 200 -
127.0.0.1 -- [10/Oct/2022 05:38:31] "POST / HTTP/1.1" 200 -
```

Client.py

```
===== RESTART: D:/Study/SEM 5 Notes/CN/LAB/LAB EXP PGM/Ex no 4 b/Client.py =====
PRESS 1-->STRAT || 2--> STOP
ENTER YOUR CHOICE1
ENTER THE ELEMENTS TO ADD FIRST LIST
PRESS -1 TO EXIT THIS LIST
-->10
-->54
-->54
-->78
-->20
-->-1
ENTER THE ELEMENTS TO ADD SECOND LIST
PRESS -2 TO EXIT THIS LIST
-->20
-->10
-->45
-->54
-->70
-->-2
[10, 54, 54, 78, 20]
[20, 10, 45, 54, 70]
list_length 5
list_maximum 78
list_minimum 10
list_to_set [78, 10, 20, 54]
list_concat [10, 54, 54, 78, 20, 20, 10, 45, 54, 70]
PRESS 1-->STRAT || 2--> STOP
ENTER YOUR CHOICE2
>>>
```

RESULT

Thus the program for RPC for list operations- XMLRPC

Ex no 4 c REMOTE PROCEDURE CALL FOR SORTING AN ARRAY – XMLRPC**Date:****AIM:**

To Implement an XML RPC code for sorting array

ALGORITHM

1. Write a server code first and execute it.
2. Now write client program and execute
3. The server is listening to port waiting for client to connect
4. Now , connection is successful and given array in client side is executed successfully.
5. The array is sorted. Connection is completed

PROGRAM**Server.py**

```
from xmlrpc.server import SimpleXMLRPCServer
def selection_sort(x):
    x.sort()
server = SimpleXMLRPCServer(("localhost", 8000))
print("Listening on port 8000...")
server.register_function(selection_sort,"selection_sort")
server.serve_forever()
```

Client.py

```
import xmlrpc.client
proxy= xmlrpc.client.ServerProxy('http://localhost:8000/')
while True:
    a = [1, 3, 4, 2]
    print("Sorted",proxy.selection_sort(a))
```

OUTPUT

Server.py

```
>>>
===== RESTART: D:\Study\SEM 5 Notes\CN\LAB\LAB EXP PGM\Ex no 4 c\Server.py =====
Listening on port 8000...
```

Client.py

```
>>>
===== RESTART: D:/Study/SEM 5 Notes/CN/LAB/LAB EXP PGM/Ex no 4 c/Client.py =====
Sorted 1,2,3,4
```

RESULT

Thus the remote procedure call for sorting an array

Ex No: 5 STUDY OF PACKET TRACER-Installation and User Interface Overview**Date:****AIM:**

To study the Packet tracer tool Installation and User Interface Overview.

INTRODUCTION:

A simulator, as the name suggests, simulates network devices and its environment. Packet Tracer

is an exciting network design, simulation and modelling tool.

1. It allows you to model complex systems without the need for dedicated equipment.
2. It helps you to practice your network configuration and troubleshooting skills via computer or an Android or iOS based mobile device.
3. It is available for both the Linux and Windows desktop environments.
4. Protocols in Packet Tracer are coded to work and behave in the same way as they would on real hardware.

INSTALLING PACKET TRACER:

To download Packet Tracer, go to <https://www.netacad.com> and log in with your Cisco Networking

Academy credentials; then, click on the Packet Tracer graphic and download the package appropriate for your operating system. (Can be used to download in your laptop).

Windows

Installation in Windows is pretty simple and straightforward; the setup comes in a single file named Packettracer_Setup6.0.1.exe. Open this file to begin the setup wizard, accept the license agreement, choose a location, and start the installation.

Linux

Linux users with an Ubuntu/Debian distribution should download the file for Ubuntu, and those

using Fedora/Redhat/CentOS must download the file for Fedora. Grant executable permission to

this file by using chmod, and execute it to begin the installation.

```
chmod +x PacketTracer601_i386_installer-rpm.bin
```

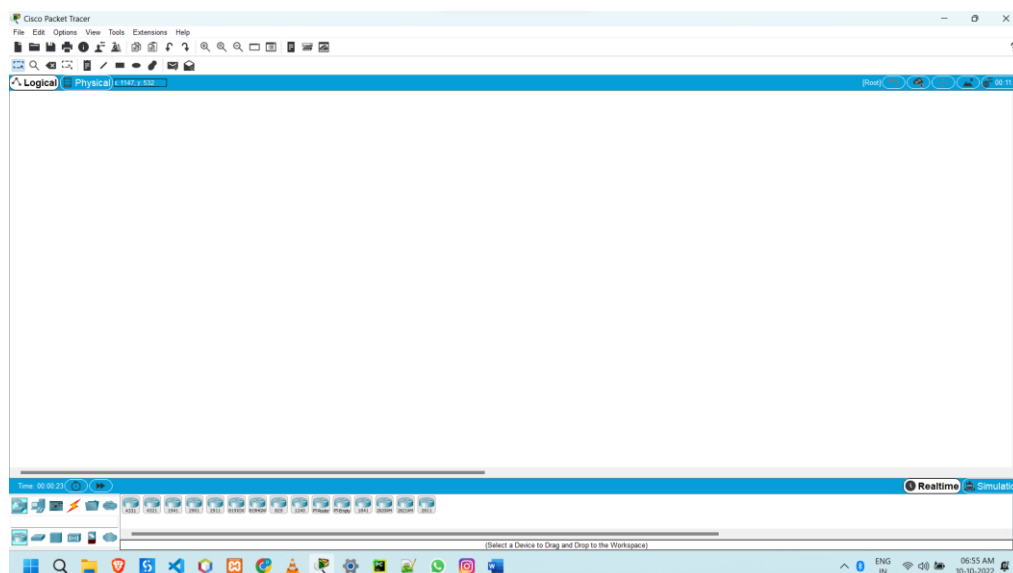
```
./PacketTracer601_i386_installer-rpm.bin
```

USER INTERFACE OVERVIEW:

The layout of Packet Tracer is divided into several components. The components of the Packet Tracer interface are as follows: match the numbering with explanations.

1. Menu bar – This is a common menu found in all software applications; it is used to open, save, print, change preferences, and so on.
2. Main toolbar – This bar provides shortcut icons to menu options that are commonly accessed, such as open, save, zoom, undo, and redo, and on the right-hand side is an icon for entering network information for the current network.
3. Logical/Physical workspace tabs – These tabs allow you to toggle between the Logical and Physical work areas.
4. Workspace – This is the area where topologies are created and simulations are displayed.
5. Common tools bar – This toolbar provides controls for manipulating topologies, such as select, move layout, place note, delete, inspect, resize shape, and add simple/complex PDU.
6. Real-time/Simulation tabs – These tabs are used to toggle between the real and simulation modes. Buttons are also provided to control the time, and to capture the packets.
7. Network component box – This component contains all of the network and end devices available with Packet Tracer, and is further divided into two areas: Area 7a: Device-type selection box – This area contains device categories Area 7b: Device-specific selection box – When a device category is selected, this selection box displays the different device models within that category
8. User-created packet box – Users can create highly-customized packets to test their topology from this area, and the results are displayed as a list.

OUTPUT



RESULT

Thus the installation of cisco packet tracer is completed and installed successfully.

Ex no : 6a CUSTOMIZING A NETWORK DEVICE WITH NETWORK MODULES**Date:****AIM:**

To create a simple network topology using packet tracer and customize the switch with network modules.

Network Device Modules:

Network devices form the core of networking. Any network device like switch or router, comes with some default interfaces and some empty slots. Default interfaces are usually Ethernet interfaces; however, router or switch support more different kinds of interfaces. New interfaces come in the form of modules which can be put inside those empty slots.

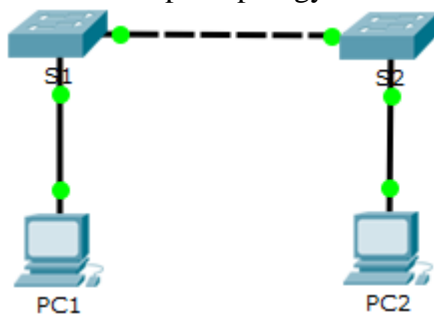
A device module is a piece of hardware containing several device interfaces. For example, a HWIC-4ESW module contains four Ethernet (10 MBps) ports. Suppose you need to establish a WAN connection between two different locations. So you need to buy a WAN module.

Creating a custom device by adding modules:

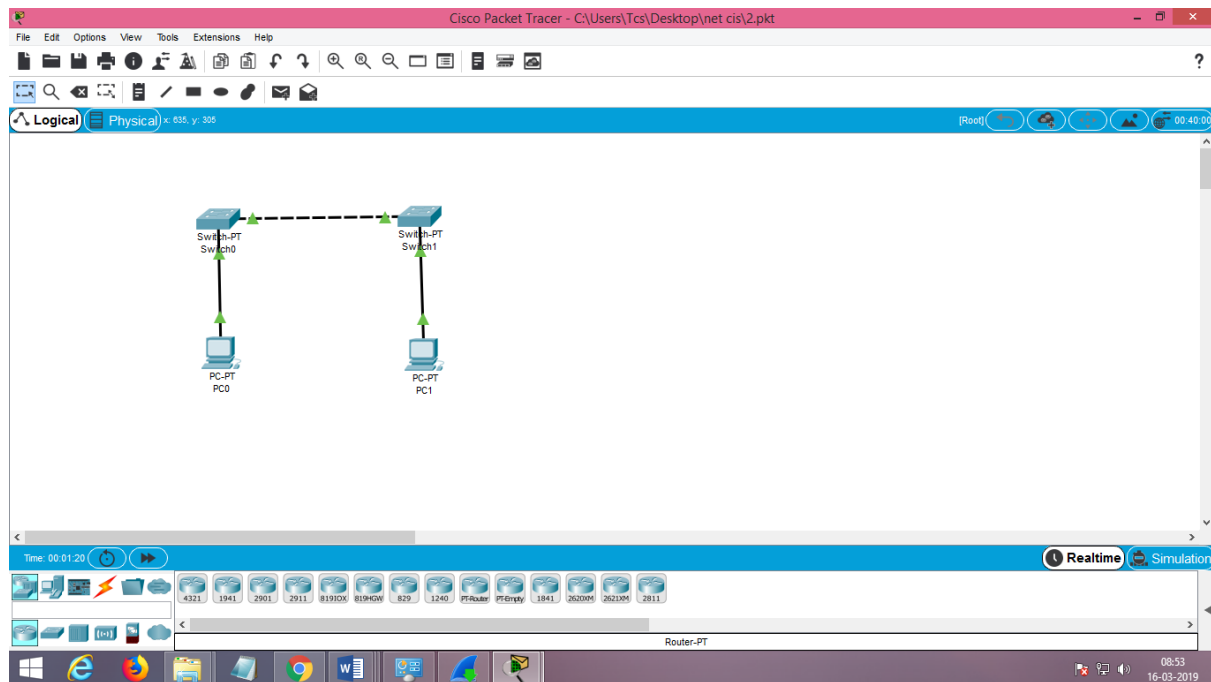
Packet Tracer offers a feature to save a device that is customized by adding modules and termed as custom-made device. The following steps create a custom network device:

1. Drag-and-drop a network device into the work area. For this example, Generic switch: Switch-PT-Empty.
 2. Click on the switch to open its configuration dialog box, and turn the device off.
 3. Add the most-used modules to this switch.
 4. Navigate to Tools | Custom Devices Dialog, or press Ctrl + E.
 5. Click on the Select button, and then click on the switch that was just customized.
 6. Provide a name and description, and then click on Add and Save.
- This custom device is saved with a .ptd extension and will be listed in the Device-specific selection box.

1. Create a Simple topology as shown below and configure the PCs and switch s1 and s2.



OUTPUT:



1. Drag-and-drop a network device into the work area. For this exercise, Generic switch: Switch-PT-Empty.
2. Click on the switch to open its configuration dialog box, and turn the device off.
3. Add the most-used modules to this switch.
4. Navigate to Tools | Custom Devices Dialog, or press Ctrl + E.
5. Click on the Select button, and then click on the switch that was just customized.
6. Provide a name and description, and then click on Add and Save

RESULT

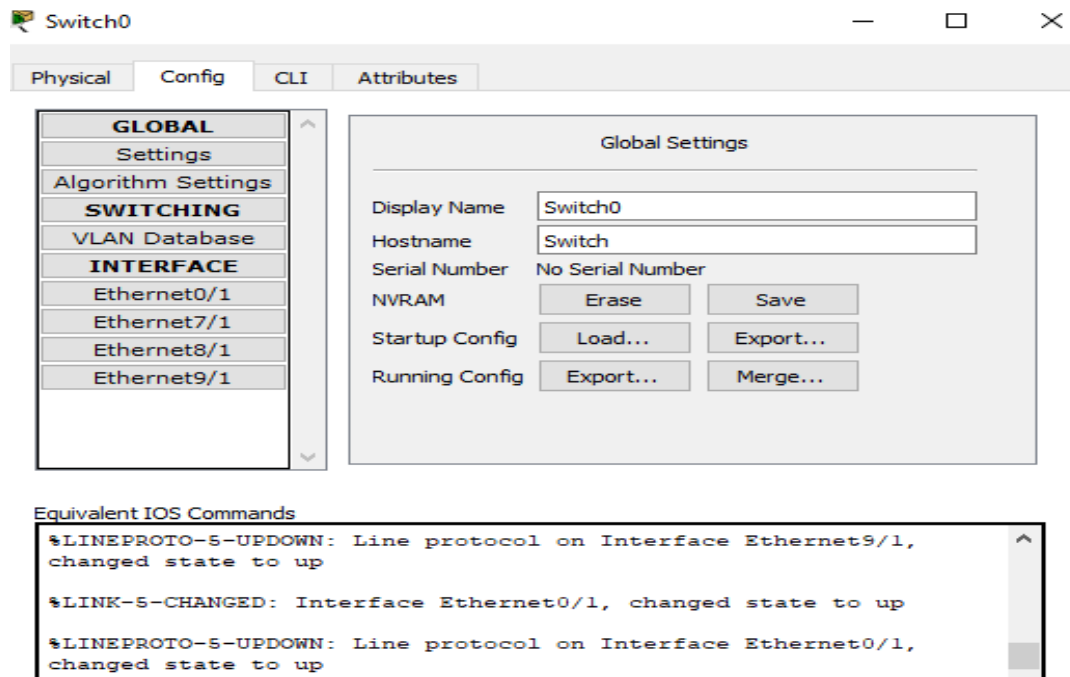
Thus the program for configuring the network is successfully executed and the output is verified.

Ex No :6b CONFIGURING A NETWORK DEVICE (ROUTERS AND SWITCHES)**Date:****AIM:**

To configure network devices (switches and routers)

DESCRIPTION:

To configure Cisco routers and switches, Packet Tracer provides a Config tab that contains GUI options for the most common configurations. Config Tab details of a switch is shown in the screen shot.



Using the Config tab, the following can be configured:

1. Global settings
2. Routing (on a router and a layer 3 switch)
3. VLAN database (on a switch)
4. Interface settings

Global settings

The first part of Global settings allows you to change the Display name and Hostname of the device. The display name can also be changed by clicking on the name below the device icon. The configuration file for the device can also be saved, erased, or exported for later use.

The Algorithm Settings section contains settings meant for advanced users who want to minutely tweak their device to see how it responds to certain situations. These settings can also be globally set for all network devices by navigating to Options | Algorithm Settings, or by using the shortcut Ctrl + Shift + M.

Interface settings

This section slightly differs from the switch and the router. Switches have options for modifying the speed and duplex setting and for assigning a port to VLAN. On routers, the VLAN section is replaced by the IP address configuration.

Configuring a Router is done by performing the following steps:

- Click on a router icon, go to the Config tab, select an interface, and configure the IP address. Make sure that you select the On checkbox in this section to bring the port state up. For example, if there are four router connected to each other then, the following IP addresses can be assigned to the Routers.

Router	Interface	IP Address
R1	FastEthernet0/0	192.168.10.1
	FastEthernet0/1	192.168.20.1
R2	FastEthernet0/0	192.168.10.2
	FastEthernet0/1	192.168.30.1
R3	FastEthernet0/0	192.168.20.2
	FastEthernet0/1	192.168.40.1
R4	FastEthernet0/0	192.168.30.2
	FastEthernet0/1	192.168.40.2

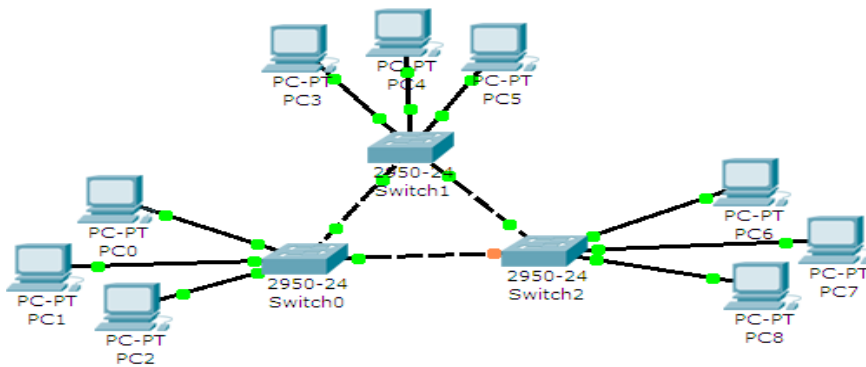
Routing

This section has options for configuring Static and dynamic routing (RIP). To configure static routing, enter the network address, netmask, and its next hop address, and then click on Add.

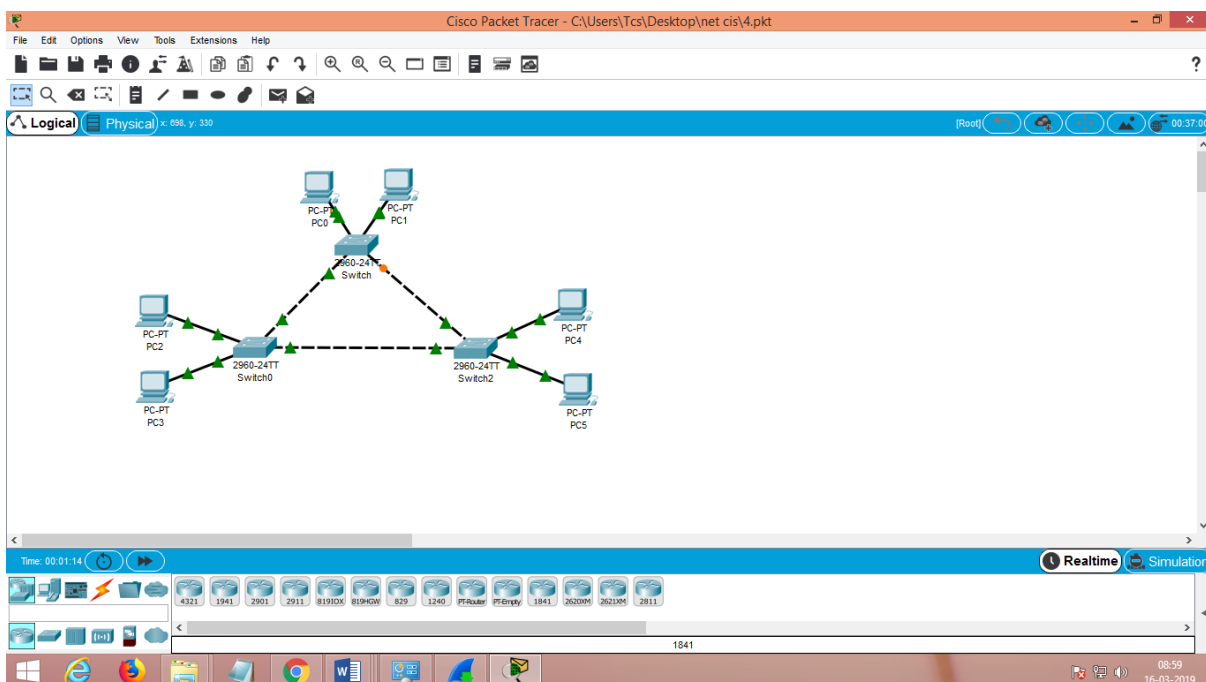
The screenshot shows the R1 configuration window with the 'Config' tab selected. The left sidebar shows the configuration tree with 'ROUTING' > 'Static' selected. The main area displays the 'Static Routes' configuration. The 'Network' field is set to 192.168.40.0, the 'Mask' is 255.255.255.0, and the 'Next Hop' is 192.168.20.2. An 'Add' button is present. Below this, a list of configured routes shows '192.168.30.0/24 via 192.168.10.2' with a 'Remove' button. At the bottom, the 'Equivalent IOS Commands' section shows a list of commands: Router(config)#, Router(config)#, Router(config)#, and Router(config)#.

To configure Routing Information Protocol (RIP), it is enough to add only network IP.

1. Create a topology shown below and configure the Initial Setting. (PC and Switch Configuration)

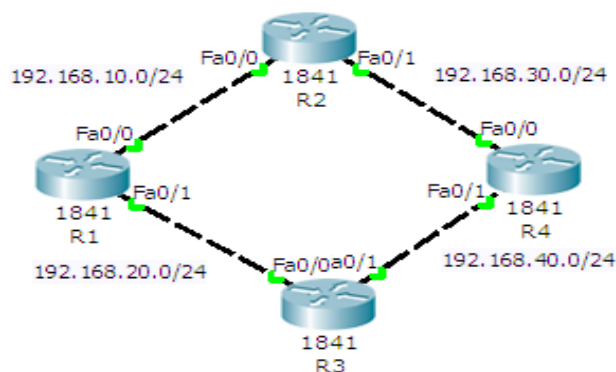


OUTPUT

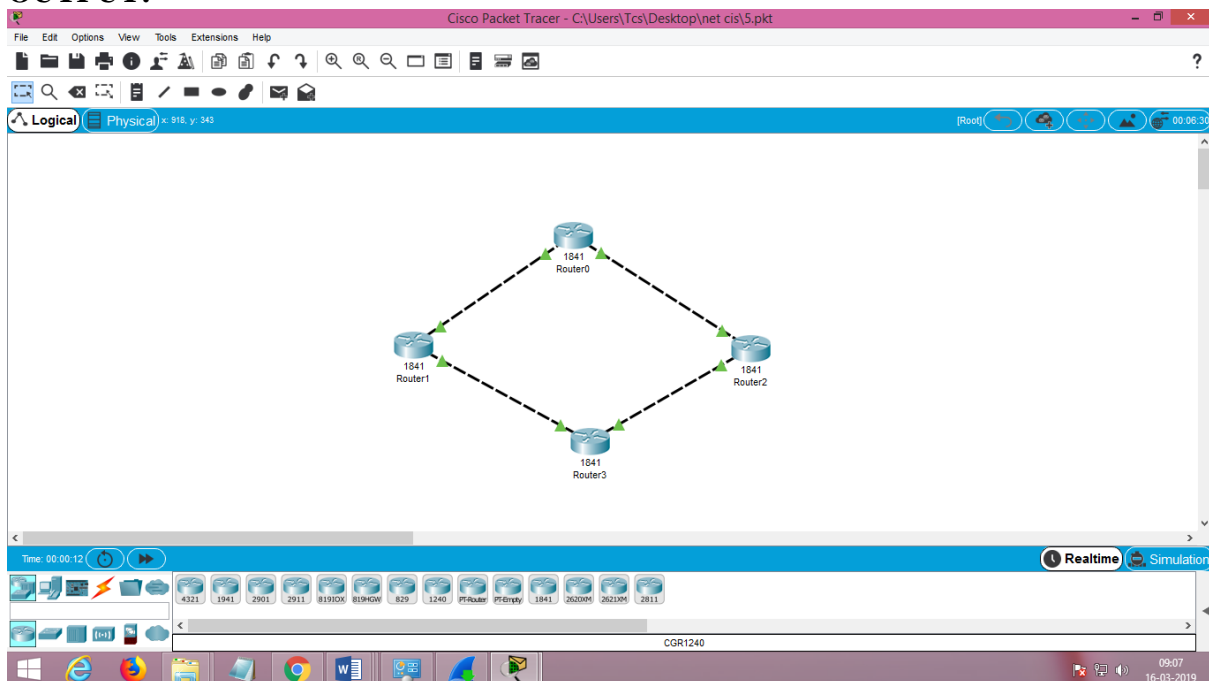


1. The topology is created as shown above.
2. The PC'S are configured.
3. The switches are configured.
4. Using the Config tab, the following can be configured:
 5. Global settings
 6. Routing (on a router and a layer 3 switch)
 7. VLAN database (on a switch)
 8. Interface setting

2. Perform Static Routing Configuration of Routers connected as shown below



OUTPUT:



Configuring a Router is done by performing the following steps:

1. Click on a router icon, go to the Config tab, select an interface, and configure the IP address. Make sure that you select the On checkbox in this section to bring the port state up. For example, if there are four routers connected to each other then, the following IP addresses can be assigned to the Routers.

Router	Interface	IP Address
R1	FastEthernet0/0	192.168.10.1
	FastEthernet0/1	192.168.20.1
R2	FastEthernet0/0	192.168.10.2
	FastEthernet0/1	192.168.30.1
R3	FastEthernet0/0	192.168.20.2
	FastEthernet0/1	192.168.40.1
R4	FastEthernet0/0	192.168.30.2
	FastEthernet0/1	192.168.40.2

RESULT

Thus the program for configuring a network device (routers and switches) is executed successfully

Ex no 6c TESTING CONNECTIVITY USING SIMPLE AND COMPLEX PDUs OPTIONS

Date:

AIM:

To test the connectivity of a network using simple and complex PDUs.

DESCRIPTION:



Once a topology has been created, connectivity can be tested between devices by using either simple or complex PDUs. Although it is possible to do the same by pinging devices from their command-line interface, using the PDU option is quicker for large topologies.

REAL-TIME MODE

Simple PDU

The Add Simple PDU option uses only ICMP (Internet Control Message Protocol). Create a topology with a PC and a server.

1. Add a PC and a server to the workspace and connect them using a copper crossover cable.
2. Assign IP addresses to both of them in the same subnet. Example, PC1: 192.168.0.1/255.255.255.0 and PC2: 192.168.0.2/255.255.255.0.
3. From the common tools bar, click on the closed envelope icon or use the shortcut key P.
4. The pointer will change to an envelope symbol. Click on the PC first and then on the server. Now look at the User Created Packet box. Status Successful, the source, the destination, and the type of packet that was sent will be shown.

Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num	Edit
	Successful	PC0	Server0	ICMP		0.000	N	0	(edit)

Complex PDU

Complex PDUs is also shown with the same PC-Server topology:

1. Click on the open envelope icon or press C; this is the Add Complex PDU option.
2. Click on the PC and the Create Complex PDU dialog box opens. Select the application and fill the Destination IP address (IP of the server), Starting Source Port, and Time fields, and then click on the Create PDU button.

Create Complex PDU

Source Settings

Source Device: PC0
 Outgoing Port:
 ☒ Auto Select Port

PDU Settings

Select Application:

Destination IP Address:

Source IP Address:

TTL:

TOS:

Starting Source Port:

Destination Port:



Size:

Simulation Settings

☒ One Shot Time: Seconds

☐ Periodic Interval: Seconds

Now click on the server and then look at the user-created packet box. An entry indicates a successful TCP three-way handshake as shown in the following screenshot:

Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num	Edit
	Successful	PC0	10.0.0.2	TCP		0.000	N	0	(edit)

SIMULATION MODE:

Use the real time/simulation tab to switch to the simulation mode. Using simulation mode, packets flowing from one node to can be seen.

Click on the Auto Capture / Play button to begin packet capture. Try a Simple PDU, as described in the previous section, and the event list will be populated with three entries, indicating the creation of an ICMP packet, ICMP echo sent, and ICMP reply received

Event List					
Vis.	Time (sec)	Last Device	At Device	Type	Info
	0.000	--	PC0	ICMP	
	0.001	PC0	Server0	ICMP	
	0.002	Server0	PC0	ICMP	

If you click on a packet (the envelope icon), you'll be presented with the packet information categorized according to OSI layers. The Outbound PDU Details tab lists each layer's information in a packet format:

PDU Information at Device: PC0

OSI Model Outbound PDU Details

PDU Formats

Ethernet II

0	4	8	14	19	Bytes
PREAMBLE: 101010...1011		DEST MAC: 0001.4279.6E10		SRC MAC: 00E0.F9CC.7B39	
TYPE: 0x800		DATA (VARIABLE LENGTH)		FCS: 0x0	

IP

0	4	8	16	19	31	Bits
4		IHL	DSCP: 0x0		TL: 28	
ID: 0x44			0x0		0x0	
TTL: 255		PRO: 0x1		CHKSUM		
SRC IP: 10.0.0.1						
DST IP: 10.0.0.2						
OPT: 0x0					0x0	
DATA (VARIABLE LENGTH)						

ICMP

0	8	16	31	Bits
TYPE: 0x8		CODE: 0x0		CHECKSUM
ID: 0xa		SEQ NUMBER: 9		

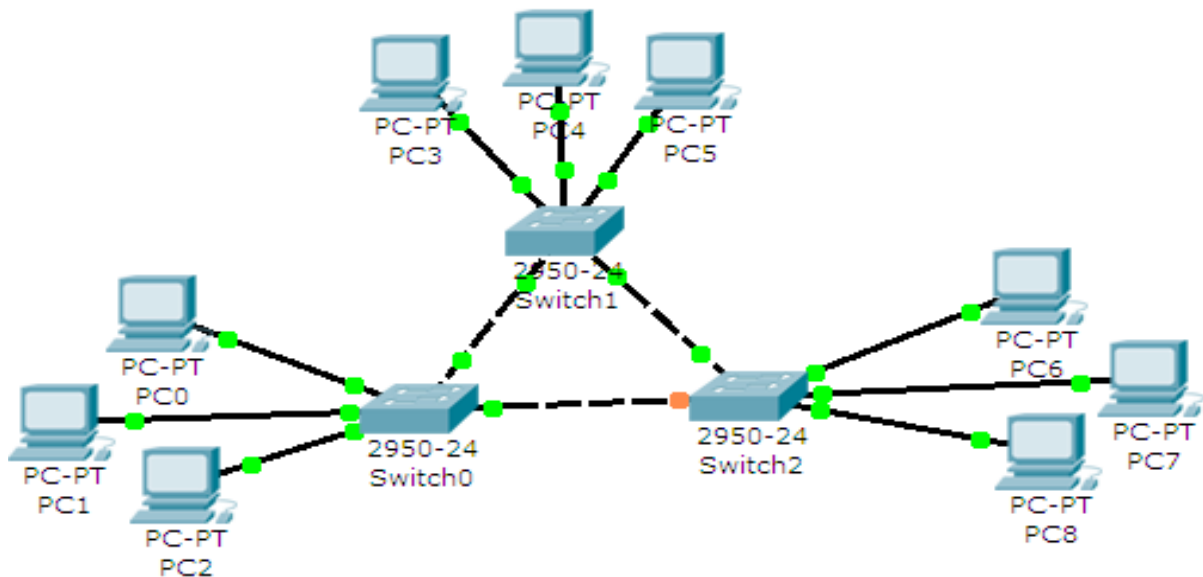
The simulation mode has a Play Controls section that works similar to the controls of a media player and is as follows:

Back: This button moves the process one step back each time it is clicked on.

Auto Capture / Play: Pressing this button results in all of the network traffic (chosen under event filters) being continuously captured until this button is pressed again.

Capture/Forward: This is the manual mode of the previous button. This has to be pressed each time to move the packet from one place to another.

1. Test the connectivity by sending simple PDU between PC1 and PC7.

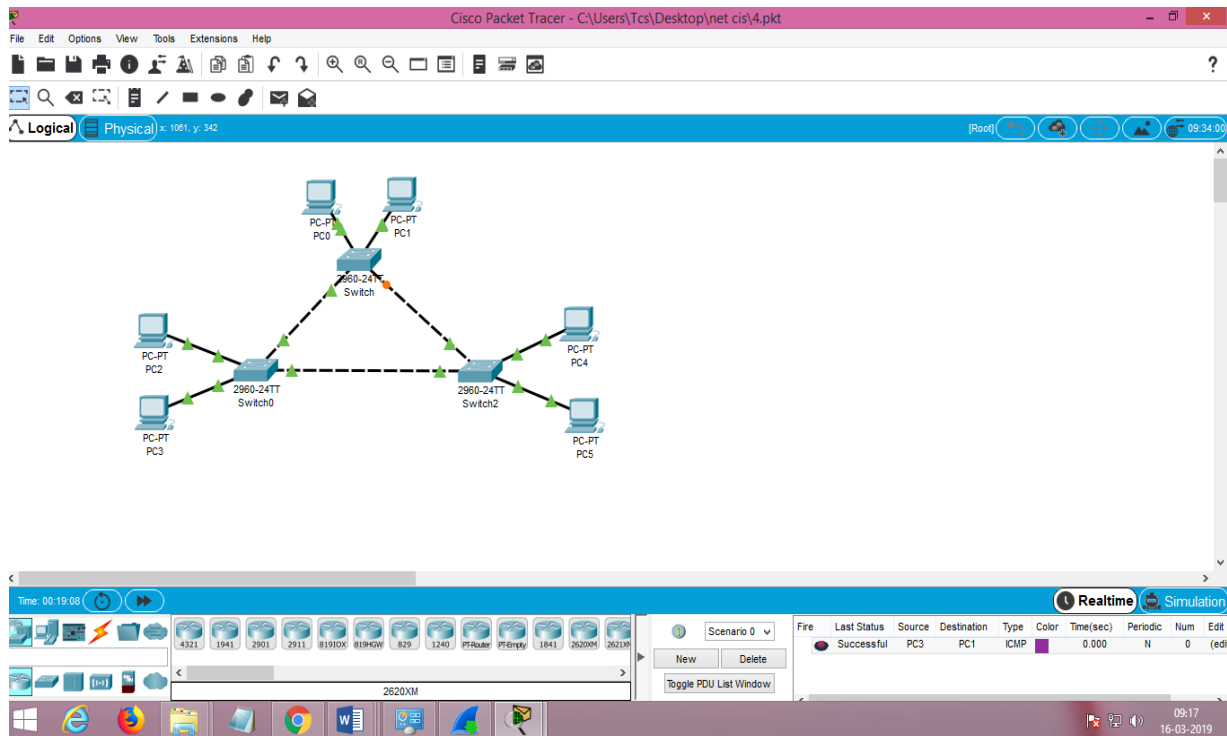


OUTPUT:

Simple PDU

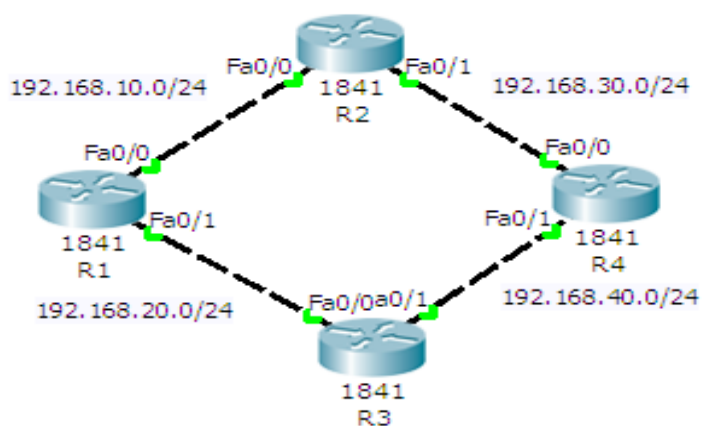
The Add Simple PDU option uses only ICMP (Internet Control Message Protocol). Create a topology with a PC and a server.

5. Add a PC and a server to the workspace and connect them using a copper crossover cable.
6. Assign IP addresses to both of them in the same subnet. Example, PC1: 192.168.0.1/255.255.255.0 and PC2: 192.168.0.2/255.255.255.0.
7. From the common tools bar, click on the closed envelope icon or use the shortcut key P. The pointer will change to an envelope symbol. Click on the PC first and then on the server.



Fire	Last Status	Source	Destination	Type	Color	Time (sec)	Periodic	Num	Edit
	Successful	PC0	Server0	ICMP		0.000	N	0	(edit)

- Test the connectivity by sending simple PDU between R2 and R3.



OUTPUT:

The screenshot displays the Cisco Packet Tracer interface. The main workspace shows a network topology with four routers: Router0 (top), Router1 (left), Router2 (right), and Router3 (bottom). They are interconnected in a diamond topology. The Event List panel on the right shows the following events:

Vis.	Time(sec)	Last Device	AI Device	Type
	0.000	--	Router2	ICMP
	0.001	Router2	Router0	ICMP
	0.002	Router0	Router2	ICMP
	54.641	--	Router1	CDP
	54.641	--	Router1	CDP

The bottom status bar indicates the simulation is running in Realtime mode. The Event List panel also shows a table of events:

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit
	Successful	Router2	Router0	ICMP		0.000	N	0	(edit)

RESULT

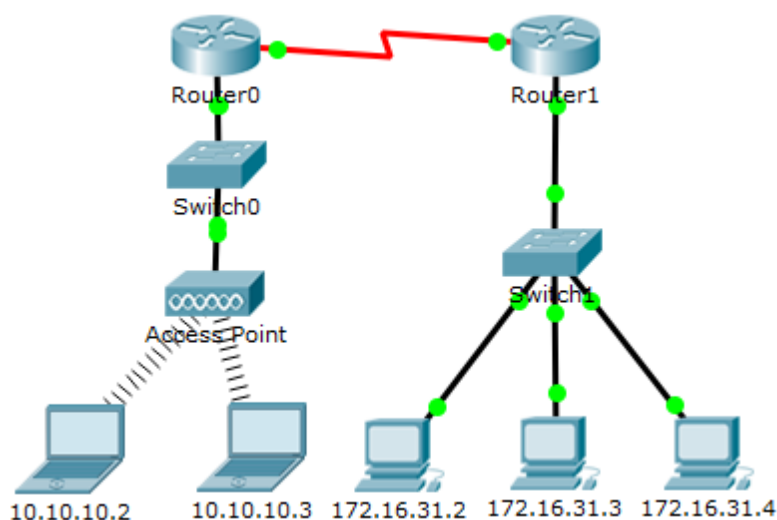
Thus the program testing connectivity using simple and complex PDUS options is executed successfully

Ex no 6d**EXAMINE THE ARP TABLE****Date:****AIM:**

To test the connectivity of a network using simple and complex PDUs.

DESCRIPTION:

TOPOLOGY: Create a topology as shown below.



Address the devices as given in the table

Device	Interface	MAC Address	Switch Interface
Router0	Gg0/0	0001.6458.2501	G0/1
S0/0/0	N/A	N/A	
Router1	G0/0	00E0.F7B1.8901	G0/1
S0/0/0	N/A	N/A	
10.10.10.2	Wireless	0060.2F84.4AB6	F0/2
10.10.10.3	Wireless	0060.4706.572B	F0/2
172.16.31.2	F0	000C.85CC.1DA7	F0/1
172.16.31.3	F0	0060.7036.2849	F0/2
172.16.31.4	G0	0002.1640.8D75	F0/3

Objectives

1. Examine an ARP Request
2. Examine a Switch MAC Address Table

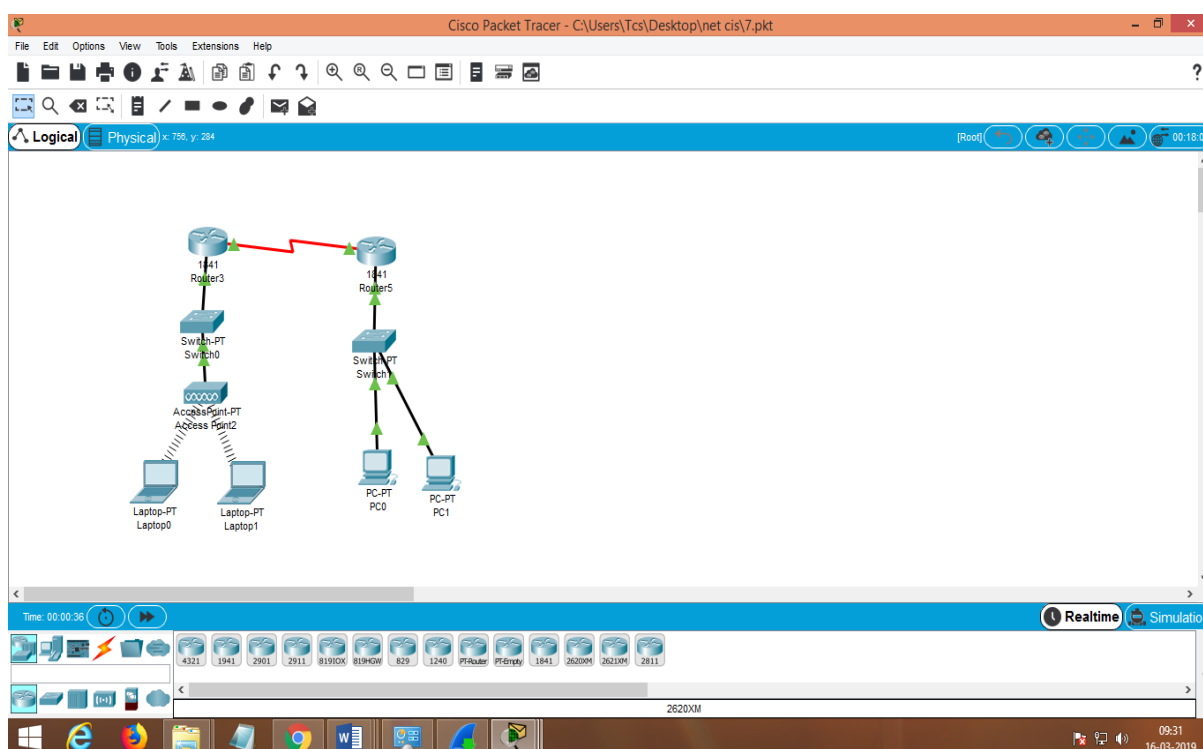
Examine an ARP Request

- Enter Simulation mode.
- Click 172.16.31.2 and open the Command Prompt.
- Enter the `arp -d` command to clear the ARP table.
- Enter the command `ping 172.16.31.3`.
- Two PDUs will be generated.
- The ping command cannot complete the ICMP packet without knowing the MAC address of the destination.
- So the computer sends an ARP broadcast frame to find the MAC address of the destination.
- Click Capture/Forward once. The ARP PDU moves Switch1 while the ICMP PDU disappears, waiting for the ARP reply.
- Click Capture/Forward to move the PDU to the next device.
- Click Capture/Forward until the PDU returns to 172.16.31.2.
- The ICMP packet reappears.
- Switch back to Realtime and the ping completes.
- Click 172.16.31.2 and enter the `arp -a` command. The Mac address of 172.16.31.3 will be added to the ARP table.

Examine a Switch MAC Address Table

- From 172.16.31.2, enter the `ping 172.16.31.4` command.
- Click 10.10.10.2 and open the Command Prompt.
- Enter the `ping 10.10.10.3` command.
- Click Switch1 and then the CLI tab. Enter the `show mac-address-table` command.
- Click Switch0, then the CLI tab. Enter the `show mac-address-table` command.

OUTPUT:



RESULT

Thus the program for ARP table is executed successfully

Ex no 7**SCAN PORTS USING NMAP TOOL****Date:****Aim:**

To scan various ports using the NMAP tool.

Nmap

- Network Mapper, is a **free, open-source tool for vulnerability scanning and network discovery**. Network administrators use Nmap to identify what devices are running on their systems, discovering hosts that are available and the services they offer, finding open ports and detecting security risks.
- Nmap can be used to **monitor** single hosts as well as vast networks that encompass **hundreds of thousands of devices and multitudes of subnets**.
- **It's a port-scan tool**, gathering information by sending raw packets to system ports. It **listens for responses and determines whether ports are open, closed or filtered in some way by, for example, a firewall**. Other terms used for port scanning include port discovery or enumeration.
- Nmap uses **raw IP packets to determine**:
 - what hosts are available on the network,
 - what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running,
 - what type of packet filters/firewalls are in use, and dozens of other characteristics.
- To ensure that the router is protected from unwanted intruders one of the basic tools for this job is Network Mapper. This program will scan a target and report which ports are open and which are closed, among other things.
- Security specialists use this program to test the security of a network.
- Basis of Nmap's functionality is port scanning, it allows for a variety of related capabilities including:
 - **Network mapping**: Nmap can identify the devices on a network (also called host discovery), including servers, routers and switches, and how they're physically connected.
 - **OS detection**: Nmap can detect the operating systems running on network devices (also called OS fingerprinting), providing the vendor name, the underlying operating system, the version of the software and even an estimate of devices' uptime.
 - **Service discovery**: Nmap can not only identify hosts on the network, but whether they're acting as mail, web or name servers, and the particular applications and versions of the related software they're running.
 - **Security auditing**: Figuring out what versions of operating systems and applications are running on network hosts lets network managers determine their vulnerability to specific flaws. If a network admin receives an alert about a vulnerability in a particular version of an application, for example, she can scan her network to identify whether that software version is running on the network and take steps to patch or update

the relevant hosts. Scripts can also automate tasks such as detecting specific vulnerabilities..

- To get started, download and install Nmap from the nmap.org website and then launch a command prompt.
- Inside command prompt type:
nmap [hostname] or nmap [ip_address]
- **nmap [hostname] or nmap [ip_address]** will initiate a default scan.
- A default scan uses 1000 common TCP ports and has Host Discovery enabled.

Host Discovery performs a check to see if the host is online. In a large IP range, this is useful for identifying only active or interesting hosts, rather than scanning every single port on every single IP in the range (a lot of which may not even be there).

The information returned is PORT | STATE | SERVICE.

STATE	Description
Open	The target port actively responds to TCP/UDP/SCTP requests.
Closed	The target port is active but not listening.
Filtered	A firewall or packet filtering device is preventing the port state being returned.
Unfiltered	The target port is reachable but Nmap cannot determine if it is open or closed.
Open/Filtered	Nmap cannot determine if the target port is open or filtered.
Closed/Filtered	Nmap cannot determine if the target port is closed or filtered.

Nmap Port Scanning Commands:

Syntax: nmap -open [ip_address]

The “-open” parameter “-open” parameter shows you ports with an “Open” state.

Scanning a single port

Syntax: nmap -p 80 [ip_address]

This command will initiate a default scan against the target host and look for port 80.

Scanning a specific range of ports

Syntax: nmap -p 1-200 [ip_address]

This command will initiate a default scan against the target host and look for ports between the range of 1-200.

Scanning the entire port range

Syntax: nmap -p- [ip_address]

This command will initiate a scan against the target host looking for all ports (1-65535).

Scanning the top 100 ports (fast scan)

Syntax: nmap -F [ip_address]

This command will initiate a fast scan against the target host looking only for the top 100 common TCP ports.

Scanning multiple TCP/UDP ports

Syntax: nmap -p U:53,67-68,T:21-25,80,135 [ip_address]

This command will initiate a scan against the target host looking only for specified UDP and TCP ports.

Scanning for specific service names

Syntax: nmap -p http,ssh,msrpc,microsoft-ds [ip_address]

This command will initiate a scan against the target host looking for ports associated with specified service names.

TCP SYN scan (default)

Syntax: nmap -sS [ip_address]

This command will initiate a TCP SYN scan against the target host. A TCP SYN scan sends a SYN packet to the target host and waits for a response. If it receives an ACK packet back, this indicates the port is open. If an RST packet is received, this indicates the port is closed. If no response is received after multiple transmissions, the port is considered filtered (a device or application between the source and the target is filtering the packets).

TCP connect scan

Syntax: nmap -sT [ip_address]

This command will initiate a TCP connect scan against the target host. A TCP connect scan is the default scan performed if a TCP SYN scan is not possible. This type of scan requests that the underlying operating system try to connect with the target host/port using the 'connect' system call.

UDP port scan

Syntax: nmap -sU [ip_address]

This command will initiate a UDP port scan against the target host. A UDP scan sends a UDP packet to the target port(s). If a response is received, the port is classified as Open. If no response is received after multiple transmissions, the port is classified as open/filtered.

SCTP INIT scan

Syntax: nmap -sY [ip_address]

This command will initiate an SCTP INIT scan against the target host. An SCTP INIT scan is similar to the TCP SYN scan but specific to the SCTP protocol. An INIT chunk is sent to the target port(s). If an INIT-ACK chunk is received back, the port is classified as open. If an ABORT chunk is received, the port is classified as closed. If no response is received after multiple transmissions, the port is classified as filtered.

OUTPUT

1. Perform a port scan over DNS and POP3 ports

command:

DNS: `nmap -p 53 127.0.0.1`

POP3: `nmap -p 110 127.0.0.1`

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\tcs>nmap -p 53 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-16 20:55 Pacific Daylight Time
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00s latency).

PORT      STATE SERVICE
53/tcp    closed domain

Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds

C:\Users\tcs>nmap -p 110 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-16 20:55 Pacific Daylight Time
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00s latency).

PORT      STATE SERVICE
110/tcp   closed pop3

Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds
C:\Users\tcs>
```

2. Perform a port scan over SMTP and IMAP ports

Command :

SMTP: `nmap -p 25 127.0.0.1`

IMAP: `nmap -p 143 127.0.0.1`

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\tcs>nmap -p 25 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-16 20:55 Pacific Daylight Time
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00s latency).

PORT      STATE SERVICE
25/tcp    closed smtp

Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds

C:\Users\tcs>nmap -p 143 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-16 20:56 Pacific Daylight Time
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00s latency).

PORT      STATE SERVICE
143/tcp   closed imap

Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds
C:\Users\tcs>_
```

3. Perform a port scan over HTTP and Telnet port

Command:

HTTP : nmap -p 80 127.0.0.1

Telnet port: : nmap -p 23 127.0.0.1

```
C:\Users\tcs>nmap -p 80 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-16 20:56 Pacific Daylight Time
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0010s latency).
PORT      STATE SERVICE
80/tcp    closed http
Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds
C:\Users\tcs>nmap -p 23 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-16 20:57 Pacific Daylight Time
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000s latency).
PORT      STATE SERVICE
23/tcp    closed telnet
Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds
C:\Users\tcs>
```

4. Perform a port scan over any network application created by you

Command: *nmap -p 8000 127.0.0.1*

OUTPUT

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.
C:\Users\tcs>nmap -p 8000 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-16 21:26 Pacific
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000s latency).
PORT      STATE SERVICE
8000/tcp  closed http-alt
Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds
C:\Users\tcs>
```

5. Perform a scan over the target host 172.16.9.83.

Command:

nmap -p 172.16.8.133

```
C:\Users\tcs>nmap 172.16.8.133
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-16 21:41 Pacific Day
Nmap scan report for 172.16.8.133
Host is up (0.0011s latency).
Not shown: 991 closed ports
PORT      STATE SERVICE
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds
49152/tcp  open  unknown
49153/tcp  open  unknown
49154/tcp  open  unknown
49155/tcp  open  unknown
49156/tcp  open  unknown
49158/tcp  open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.33 seconds
C:\Users\tcs>
```

6. Perform a port scan over TCP ports and UDP ports.

Command:

nmap -p T:80 127.0.0.1

nmap -p U:53 127.0.0.1

```
Microsoft Windows [Version 6.0.6002]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\tcs>nmap -p T:80 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-16 21:02 Pacific Daylight Ti
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00s latency).
PORT      STATE SERVICE
80/tcp    closed http
Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds

C:\Users\tcs>nmap -v -sU -sT -p U:53,111,137,T:21-25,80,139,8080 127.0.0.1
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-16 21:02 Pacific Daylight Ti
Initiating Parallel DNS resolution of 1 host. at 21:02
Completed Parallel DNS resolution of 1 host. at 21:02. 0.00s elapsed
Initiating UDP Scan at 21:02
Scanning localhost (127.0.0.1) [3 ports]
Completed UDP Scan at 21:02. 0.03s elapsed (3 total ports)
Initiating Connect Scan at 21:02
Scanning localhost (127.0.0.1) [8 ports]
Completed Connect Scan at 21:02. 2.11s elapsed (8 total ports)
Nmap scan report for localhost (127.0.0.1)
Host is up (0.66s latency).
PORT      STATE SERVICE
11/tcp    closed ftp
22/tcp    closed ssh
23/tcp    closed telnet
24/tcp    closed priv-mail
25/tcp    closed smtp
80/tcp    closed http
139/tcp   closed netbios-ssn
8080/tcp  closed http-proxy
137/udp   closed domain
111/udp   closed rshind
137/udp   closed netbios-ns

Read data files from: C:\Program Files\Nmap
Nmap done: 1 IP address (1 host up) scanned in 2.32 seconds
Raw packets sent: 3 (106B) 1 Rcvd: 6 (456B)
C:\Users\tcs>
```

RESULT

Thus the program for Nmap is executed successfully.

Ex no 8a

STUDY OF WIRESHARK TOOL

Date:

AIM:

To study packet sniffing concepts using Wireshark Tool.

DESCRIPTION:

WIRESHARK

Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and displays them in human-readable format. Wireshark includes filters, color coding, and other features that let you dig deep into network traffic and inspect individual packets. You can use Wireshark to inspect a suspicious program's network traffic, analyze the traffic flow on your network, or troubleshoot network problems.

What we can do with Wireshark:

- Capture network traffic
- Decode packet protocols using dissectors
- Define filters – capture and display
- Watch smart statistics
- Analyze problems
- Interactively browse that traffic

Wireshark used for:

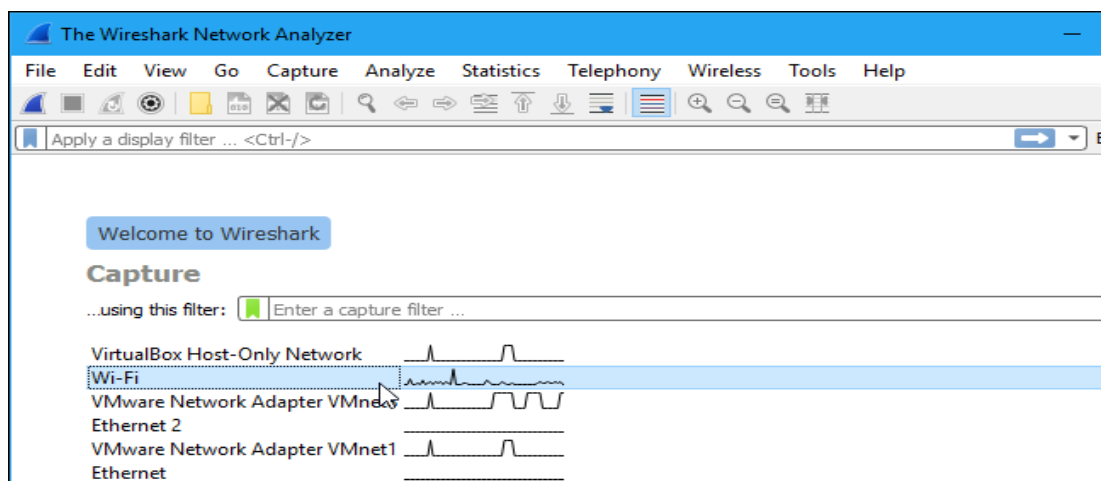
- Network administrators: troubleshoot network problems
- Network security engineers: examine security problems
- Developers: debug protocol implementations
- People: learn **network protocol internals**

Getting Wireshark

Wireshark can be downloaded for Windows or macOS from its official website. For Linux or another UNIX-like system, Wireshark will be found in its package repositories. For Ubuntu, Wireshark will be found in the Ubuntu Software Center.

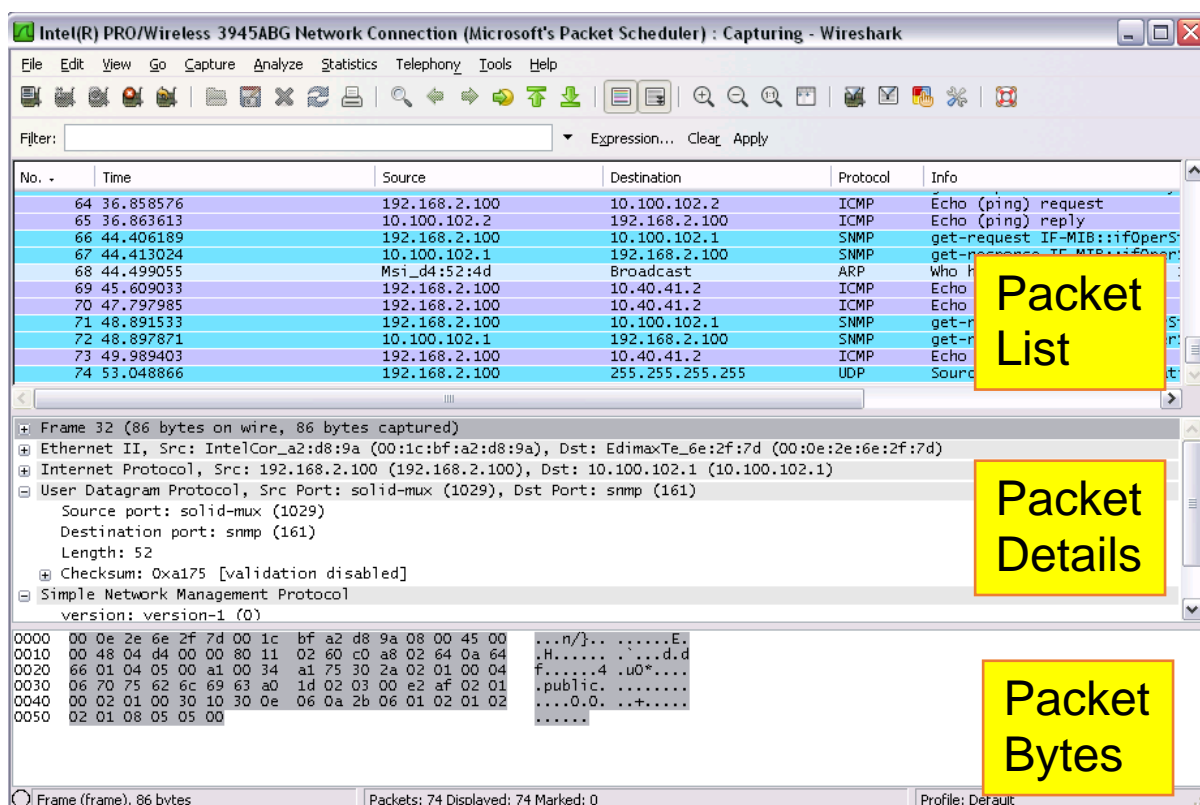
Capturing Packets

After downloading and installing Wireshark, launch it and double-click the name of a network interface under Capture to start capturing packets on that interface



As soon as you click the interface's name, you'll see the packets start to appear in real time. Wireshark captures each packet sent to or from your system.

If you have promiscuous mode enabled—it's enabled by default—you'll also see all the other packets on the network instead of only packets addressed to your network adapter. To check if promiscuous mode is enabled, click Capture > Options and verify the "Enable promiscuous mode on all interfaces" checkbox is activated at the bottom of this window.



Click the red "Stop" button near the top left corner of the window when you want to stop capturing traffic.

The “Packet List” Pane

The packet list pane displays all the packets in the current capture file. The “Packet List” pane Each line in the packet list corresponds to one packet in the capture file. If you select a line in this pane, more details will be displayed in the “Packet Details” and “Packet Bytes” panes.

The “Packet Details” Pane

The packet details pane shows the current packet (selected in the “Packet List” pane) in a more detailed form. This pane shows the protocols and protocol fields of the packet selected in the “Packet List” pane. The protocols and fields of the packet shown in a tree which can be expanded and collapsed.

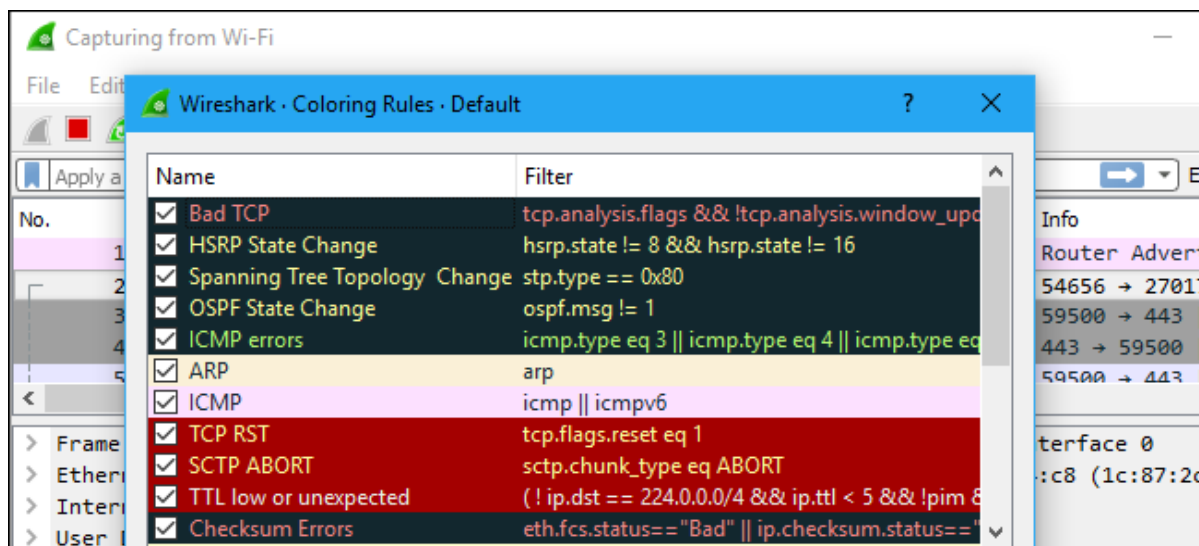
The “Packet Bytes” Pane

The packet bytes pane shows the data of the current packet (selected in the “Packet List” pane) in a hexdump style.

Color Coding

You’ll probably see packets highlighted in a variety of different colors. Wireshark uses colors to help you identify the types of traffic at a glance. By default, light purple is TCP traffic, light blue is UDP traffic, and black identifies packets with errors—for example, they could have been delivered out of order.

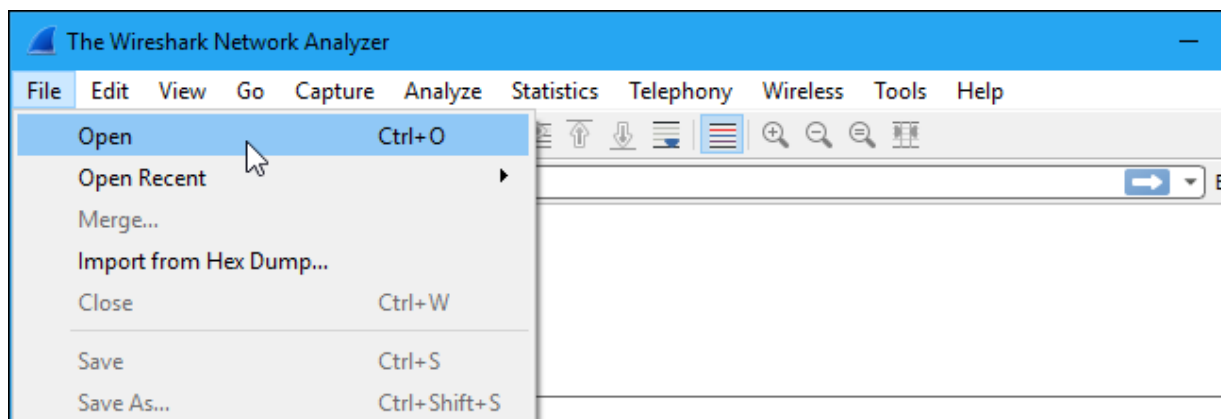
To view exactly what the color codes mean, click View > Coloring Rules. You can also customize and modify the coloring rules from here, if you like.



Sample Captures

If there's nothing interesting on your own network to inspect, Wireshark's wiki has you covered. The wiki contains a page of sample capture files that you can load and inspect. Click File > Open in Wireshark and browse for your downloaded file to open one.

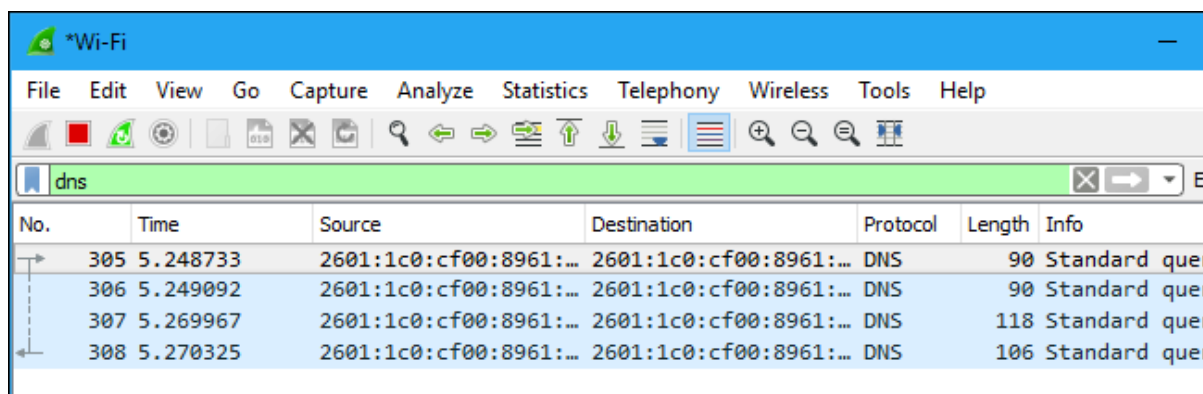
You can also save your own captures in Wireshark and open them later. Click File > Save to save your captured packets.



Filtering Packets

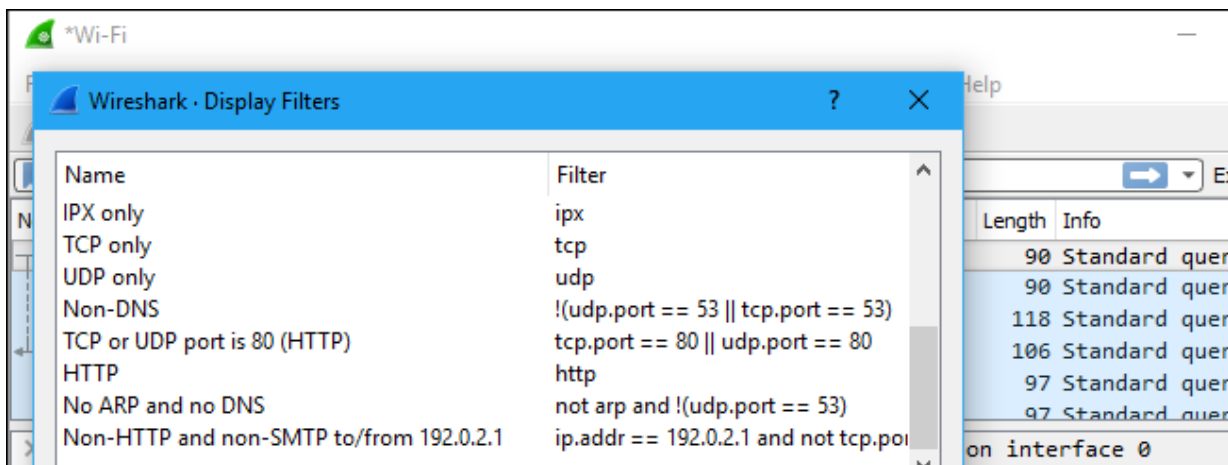
If you're trying to inspect something specific, such as the traffic a program sends when phoning home, it helps to close down all other applications using the network so you can narrow down the traffic. Still, you'll likely have a large amount of packets to sift through. That's where Wireshark's filters come in.

The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking Apply (or pressing Enter). For example, type "dns" and you'll see only DNS packets. When you start typing, Wireshark will help you autocomplete your filter.



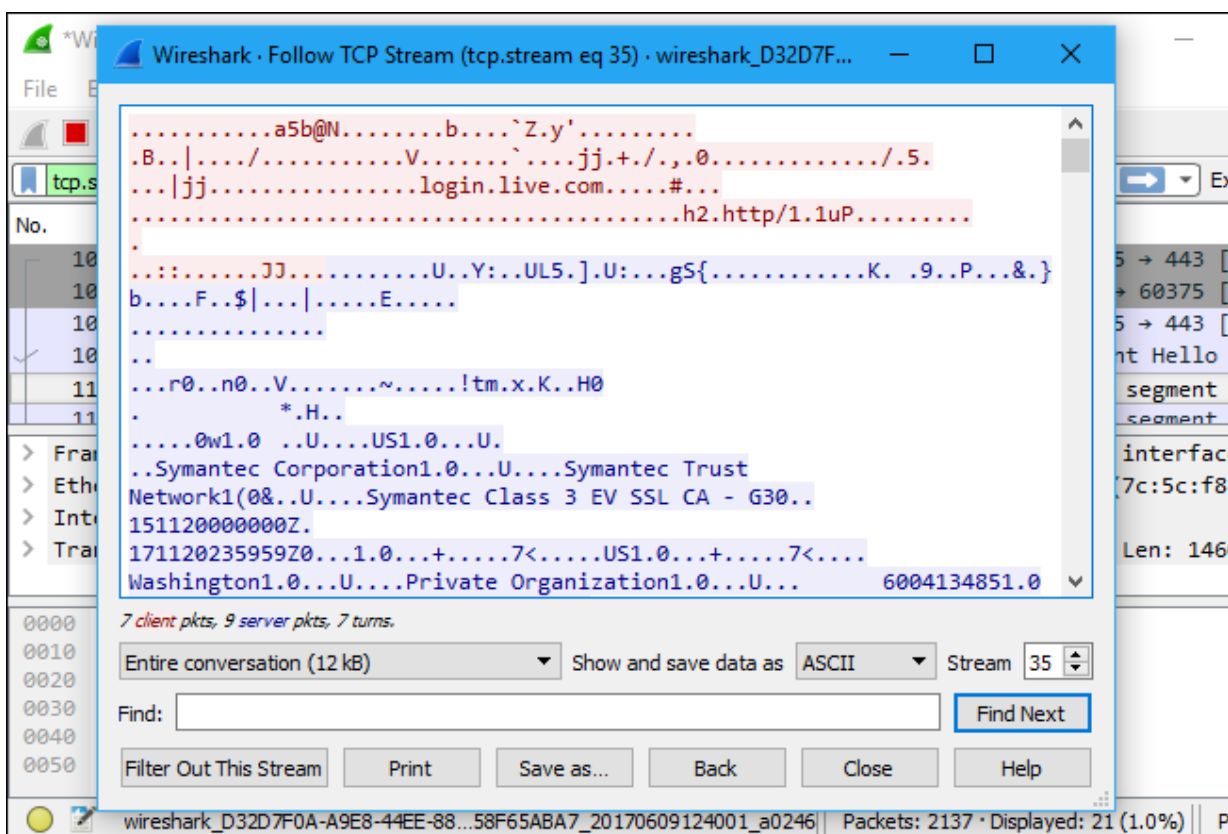
You can also click Analyze > Display Filters to choose a filter from among the default filters included in Wireshark. From here, you can add your own custom filters and save them to easily access them in the future.

For more information on Wireshark's display filtering language, read the [Building display filter expressions](#) page in the official Wireshark documentation.

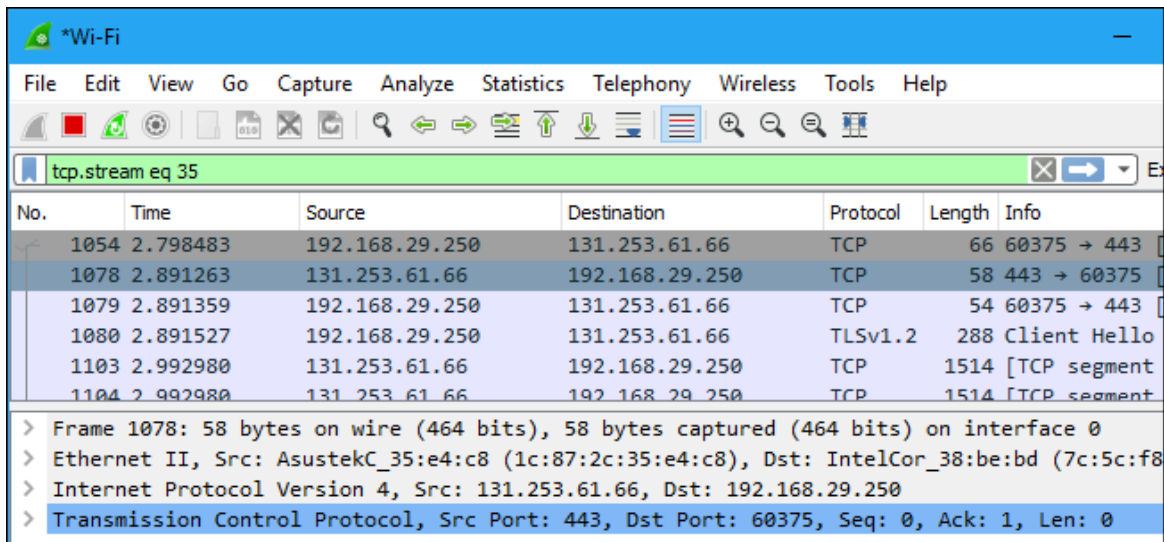


Another interesting thing you can do is right-click a packet and select Follow > TCP Stream.

You'll see the full TCP conversation between the client and the server. You can also click other protocols in the Follow menu to see the full conversations for other protocols, if applicable.



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

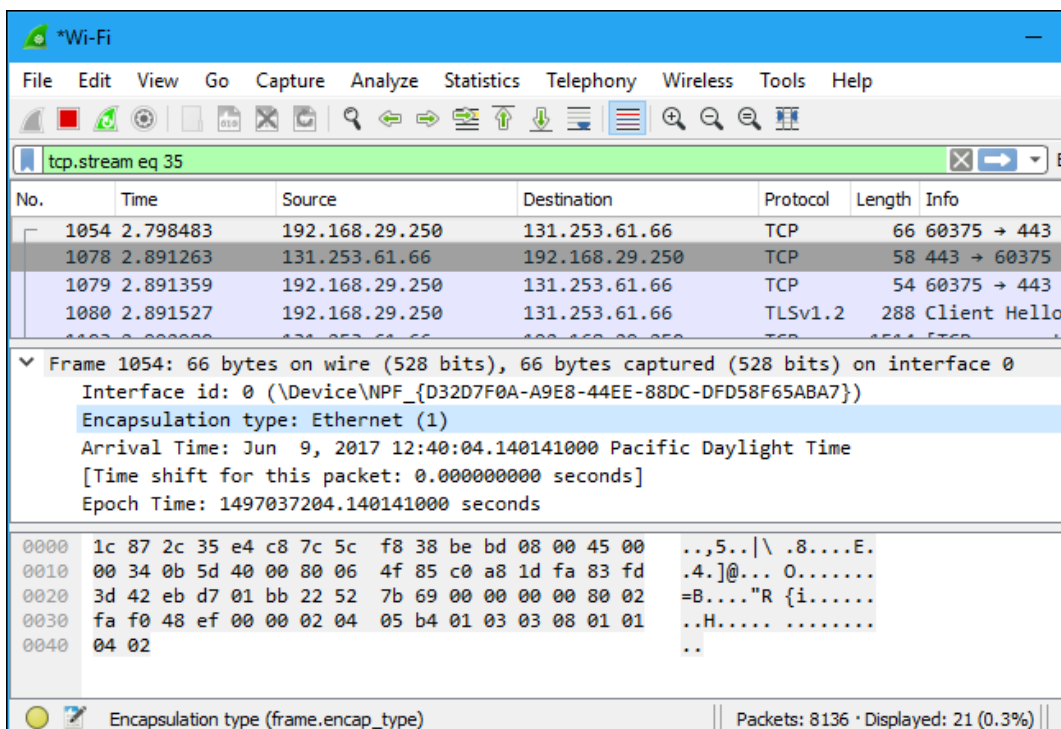


No.	Time	Source	Destination	Protocol	Length	Info
1054	2.798483	192.168.29.250	131.253.61.66	TCP	66	60375 → 443
1078	2.891263	131.253.61.66	192.168.29.250	TCP	58	443 → 60375
1079	2.891359	192.168.29.250	131.253.61.66	TCP	54	60375 → 443
1080	2.891527	192.168.29.250	131.253.61.66	TLSv1.2	288	Client Hello
1103	2.992980	131.253.61.66	192.168.29.250	TCP	1514	[TCP segment
1104	2.992980	131.253.61.66	192.168.29.250	TCP	1514	[TCP segment

> Frame 1078: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0
 > Ethernet II, Src: AsustekC_35:e4:c8 (1c:87:2c:35:e4:c8), Dst: IntelCor_38:be:bd (7c:5c:f8
 > Internet Protocol Version 4, Src: 131.253.61.66, Dst: 192.168.29.250
 > Transmission Control Protocol, Src Port: 443, Dst Port: 60375, Seq: 0, Ack: 1, Len: 0

Inspecting Packets

Click a packet to select it and you can dig down to view its details.



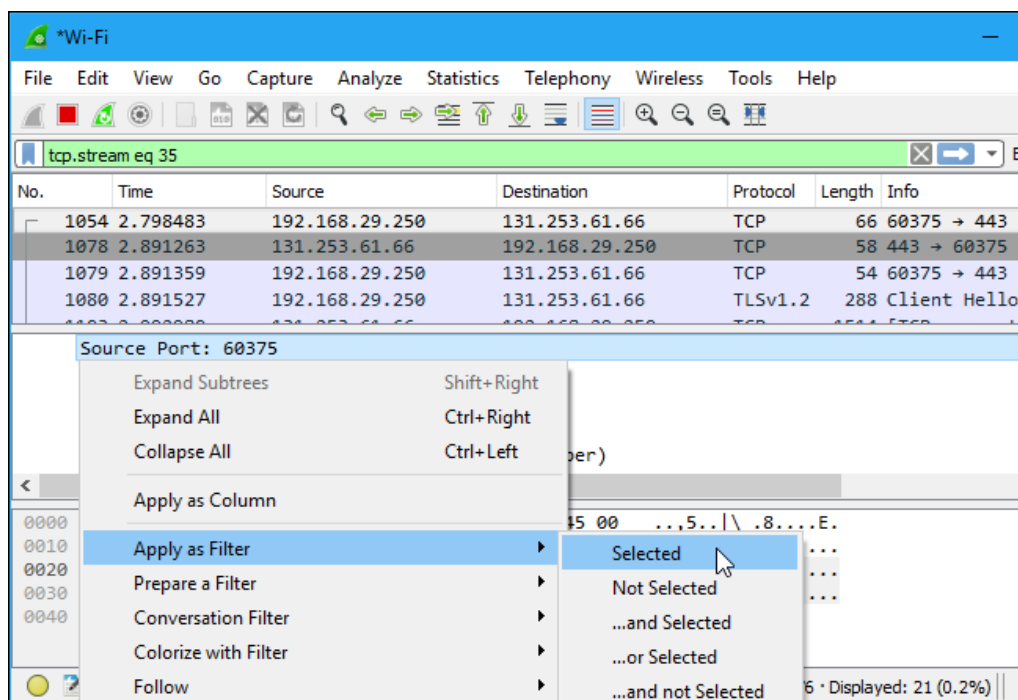
No.	Time	Source	Destination	Protocol	Length	Info
1054	2.798483	192.168.29.250	131.253.61.66	TCP	66	60375 → 443
1078	2.891263	131.253.61.66	192.168.29.250	TCP	58	443 → 60375
1079	2.891359	192.168.29.250	131.253.61.66	TCP	54	60375 → 443
1080	2.891527	192.168.29.250	131.253.61.66	TLSv1.2	288	Client Hello

Frame 1054: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
 Interface id: 0 (\Device\NPF_{D32D7F0A-A9E8-44EE-88DC-DFD58F65ABA7})
 Encapsulation type: Ethernet (1)
 Arrival Time: Jun 9, 2017 12:40:04.140141000 Pacific Daylight Time
 [Time shift for this packet: 0.000000000 seconds]
 Epoch Time: 1497037204.140141000 seconds

Offset	Hex	ASCII
0000	1c 87 2c 35 e4 c8 7c 5c f8 38 be bd 08 00 45 00	..,5.. \ .8....E.
0010	00 34 0b 5d 40 00 80 06 4f 85 c0 a8 1d fa 83 fd	.4.]@... 0.....
0020	3d 42 eb d7 01 bb 22 52 7b 69 00 00 00 00 80 02	=B...."R {i.....
0030	fa f0 48 ef 00 00 02 04 05 b4 01 03 03 08 01 01	..H.....
0040	04 02	..

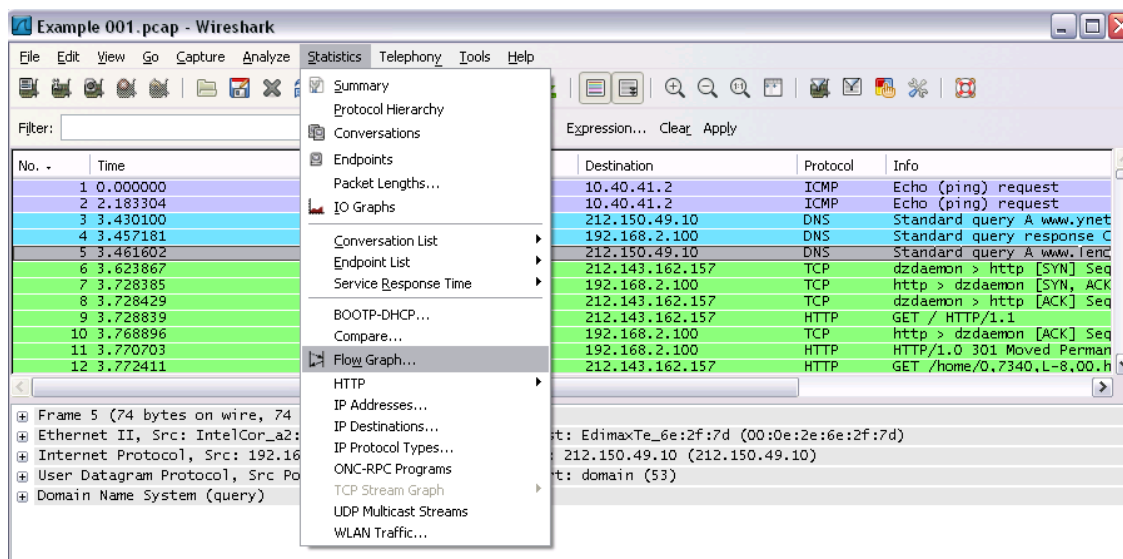
Encapsulation type (frame.encap_type) | Packets: 8136 · Displayed: 21 (0.3%)

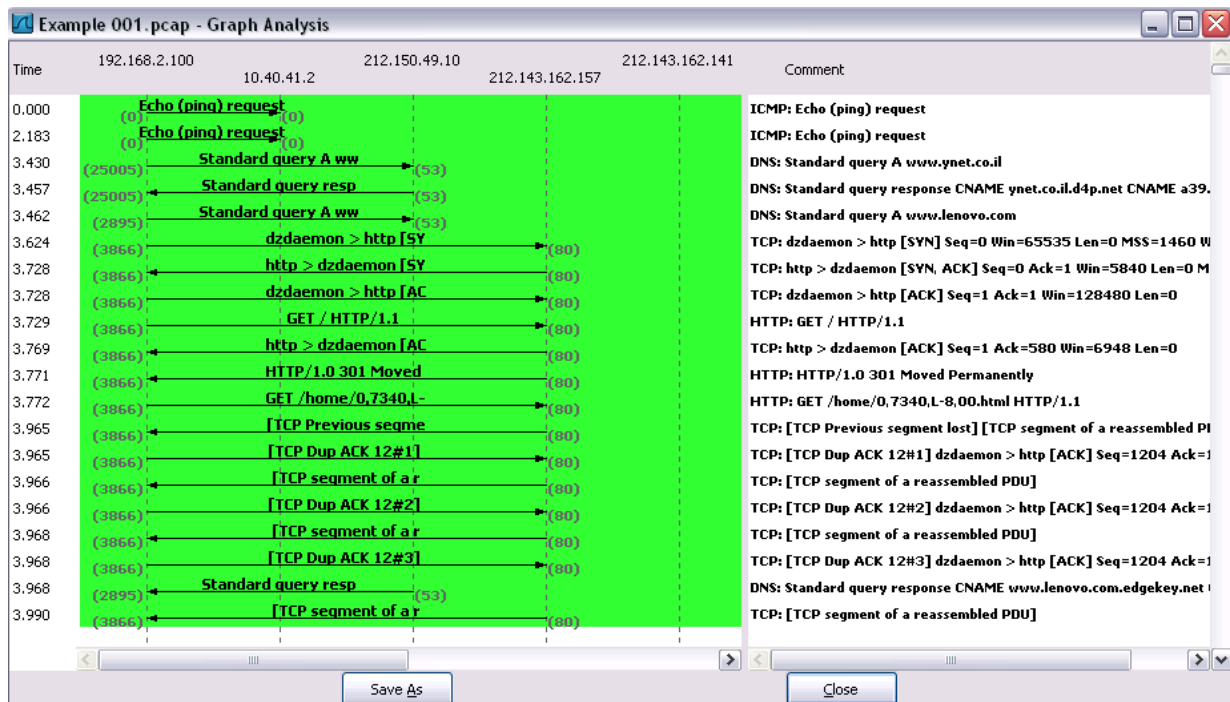
You can also create filters from here — just right-click one of the details and use the Apply as Filter submenu to create a filter based on it.



Wireshark is an extremely powerful tool, and this tutorial is just scratching the surface of what you can do with it. Professionals use it to debug network protocol implementations, examine security problems and inspect network protocol internals.

Flow Graph: Gives a better understanding of what we see.





RESULT

Thus the study for Wireshark tool has been successfully executed.

Ex no 8b: CAPTURING AND ANALYSING PACKETS USING WIRESHARK TOOL**Date:****Aim**

To filter, capture, view, packets in Wireshark Tool.

1. Capture 100 packets from the Ethernet: IEEE 802.3 LAN Interface and save it.**Procedure**

- Select Local Area Connection in Wireshark.
- Go to capture ☐ option
- Select stop capture automatically after 100 packets.
- Then click Start capture.
- Save the packets.

Output

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Pegatron_e0:87:9e	Broadcast	ARP	60	Who has 172.16.9.94? Tell 172.16.9.138
2	0.000180	RealtekS_55:2c:b8	Broadcast	ARP	60	Who has 172.16.10.36? Tell 172.16.10.50
3	0.000294	RealtekS_55:2c:b8	Broadcast	ARP	60	Who has 172.16.11.36? Tell 172.16.10.50
4	0.000295	RealtekS_55:2c:b8	Broadcast	ARP	60	Who has 172.16.8.37? Tell 172.16.10.50
5	0.000296	RealtekS_55:2c:b8	Broadcast	ARP	60	Who has 172.16.9.37? Tell 172.16.10.50
6	0.000296	RealtekS_55:2c:b8	Broadcast	ARP	60	Who has 172.16.11.37? Tell 172.16.10.50
7	0.001460	fe80::4968:12a7:5e3...	ff02::1:3	LLMNR	95	Standard query 0xae2b A TLFL3-HDC101701
8	0.001622	172.16.8.95	224.0.0.252	LLMNR	75	Standard query 0xae2b A TLFL3-HDC101701
9	0.001623	172.16.8.95	224.0.0.252	LLMNR	75	Standard query 0x28c0 AAAA TLFL3-HDC101701
10	0.001625	fe80::4968:12a7:5e3...	ff02::1:3	LLMNR	95	Standard query 0x28c0 AAAA TLFL3-HDC101701
11	0.045051	fe80::2d3b:4d33:71e0...	ff02::1:3	LLMNR	95	Standard query 0xae2b A TLFL3-HDC101701

▶ Frame 7: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface 0 ▶ Ethernet II, Src: Dell_35:10:a8 (50:9a:4c:35:10:a8), Dst: IPv6mcast_01:00:03 (33:33:00:01:00:03) ▶ Internet Protocol Version 6, Src: fe80::4968:12a7:5e36:523e, Dst: ff02::1:3 ▶ User Datagram Protocol, Src Port: 62374, Dst Port: 5355
Source Port: 62374 Destination Port: 5355 Length: 41 Checksum: 0x90e0 [unverified] [Checksum Status: Unverified] [Stream index: 0]
▶ Link-local Multicast Name Resolution (query)

0000	33 33 00 01 00 03 50 9a 4c 35 10 a8 86 dd 60 00	33....P L5....`
0010	00 00 00 29 11 01 fe 80 00 00 00 00 00 00 49 68).....Ih
0020	12 a7 5e 36 52 3e ff 02 00 00 00 00 00 00 00 00	..^6R>.....
0030	00 00 00 01 00 03 f3 a6 14 eb 00 29 90 e0 ae 2b)....+
0040	00 00 00 01 00 00 00 00 00 00 0f 54 4c 46 4c 33TLFL3
0050	2d 48 44 43 31 30 31 37 30 31 00 00 01 00 01	-HDC1017 01....

Procedure

- Select Local Area Connection in Wireshark.
- Go to capture ☐ option
- Select stop capture automatically after 100 packets.
- Then click Start capture.
- Search TCP packets in the search bar.
- To see flow graph click Statistics ☐ Flow graph.
- Save the packets.

No.	Time	Source	Destination	Protocol	Length	Info
123	4.557832	fe80::8532:3a9f:aff...	fe80::5c2b:19eb:d33...	TCP	74	1509 → 2869 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
126	4.557993	172.16.9.106	172.16.9.96	TCP	60	1506 → 2869 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1095	30.718732	172.16.8.83	172.16.9.96	TCP	66	51526 → 2869 [SYN, ECN, CW] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
1096	30.718794	172.16.9.96	172.16.8.83	TCP	66	2869 → 51526 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
1097	30.719129	172.16.8.83	172.16.9.96	TCP	60	51526 → 2869 [ACK] Seq=1 Ack=1 Win=65536 Len=0
1099	30.719919	172.16.9.96	172.16.8.83	TCP	278	2869 → 51526 [PSH, ACK] Seq=1 Ack=133 Win=65536 Len=224 [TCP segment of a reassembled PDU]
1100	30.719986	172.16.9.96	172.16.8.83	TCP	1514	2869 → 51526 [ACK] Seq=225 Ack=133 Win=65536 Len=1460 [TCP segment of a reassembled PDU]
1101	30.720279	172.16.8.83	172.16.9.96	TCP	60	51526 → 2869 [ACK] Seq=133 Ack=1685 Win=65536 Len=0
1103	30.730003	173.16.9.83	173.16.9.96	TCP	60	51526 → 2869 [ACK] Seq=133 Ack=1685 Win=65536 Len=0
▶ Frame 123: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0 ▶ Ethernet II, Src: Realtek5_b2:60:90 (00:e0:4c:b2:60:90), Dst: IntelCor_13:ed:7c (00:27:0e:13:ed:7c) ▶ Internet Protocol Version 6, Src: fe80::8532:3a9f:aff1:b3ca, Dst: fe80::5c2b:19eb:d33d:a1cd ▶ Transmission Control Protocol, Src Port: 1509, Dst Port: 2869, Seq: 1, Ack: 1, Len: 0						
0000	00 27 0e 13 ed 7c 00 e0	4c b2 60 90 86 d0 60 00L.....			
0010	00 00 00 14 06 80 fe 80	00 00 00 00 00 00 85 322			
0020	3a 9f af f1 b3 ca fe 80	00 00 00 00 00 00 5c 2b\+			
0030	19 eb d3 d3 a1 cd 05 e5	0b 35 3b ef f1 2f bf 405;./...			
0040	67 35 50 14 00 00 3e de	00 00	eSP.....			

3.Create a Filter to display only ARP packets and inspect the packets.

Procedure

- Select Local Area Connection in Wireshark.Go to capture ☐ option
- Select stop capture automatically after 100 packets.
- Then click Start capture.
- Search ARP packets in the search bar.
- Save the packets.

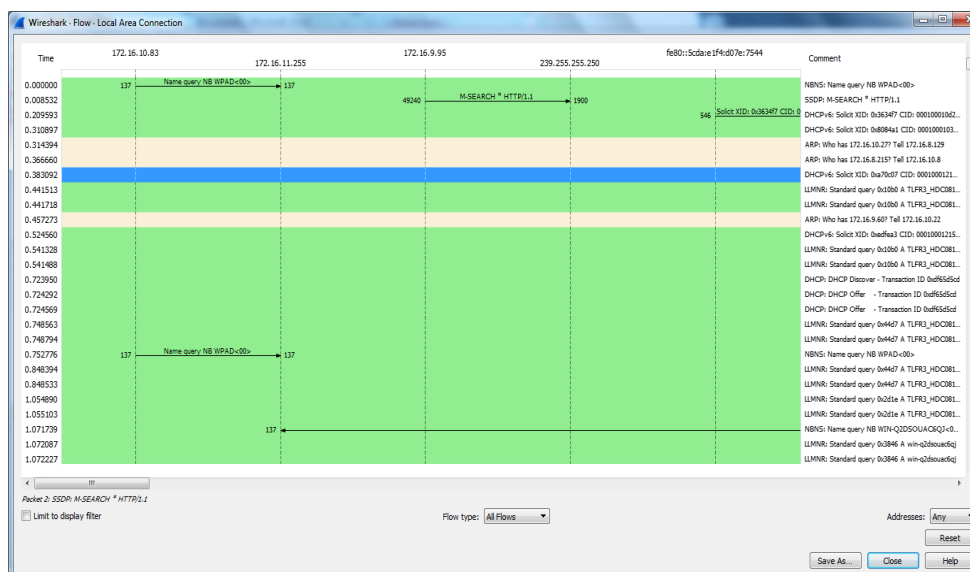
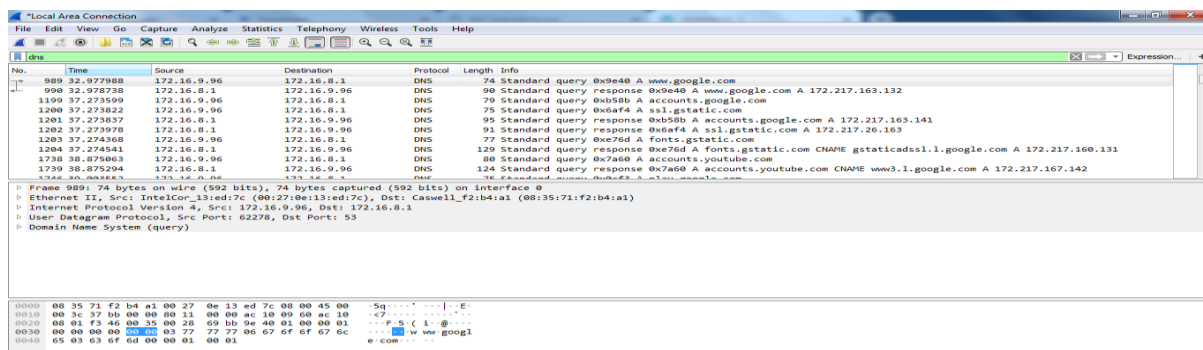
Output

arp						
No.	Time	Source	Destination	Protocol	Length	Info
6	0.255305	Foxconn_c9:c5:f0	Broadcast	ARP	60	Who has 172.16.10.15? Tell 172.16.10.3
14	0.692936	Foxconn_d0:ac:46	Broadcast	ARP	60	Who has 172.16.8.39? Tell 172.16.10.8
19	1.418424	Foxconn_c9:c9:91	Broadcast	ARP	60	Who has 172.16.8.106? Tell 172.16.10.26
24	1.880729	Foxconn_d0:ac:46	Broadcast	ARP	60	Who has 172.16.8.40? Tell 172.16.10.8
27	2.029517	Giga-Byt_92:d2:ef	Broadcast	ARP	60	Who has 172.16.10.33? Tell 172.16.10.1
41	2.509905	Giga-Byt_7c:c5:34	Broadcast	ARP	60	Who has 172.16.9.82? Tell 172.16.9.111
44	2.602358	Foxconn_c9:c8:24	Broadcast	ARP	60	Who has 172.16.8.139? Tell 172.16.10.22
46	2.743021	Dell_35:11:11	Broadcast	ARP	60	Who has 172.16.8.118? Tell 172.16.10.195
56	3.201822	Giga-Byt_92:d2:ef	Broadcast	ARP	60	Who has 172.16.10.34? Tell 172.16.10.1
60	3.237061	Giga-Byt_7c:c5:34	Broadcast	ARP	60	Who has 172.16.9.82? Tell 172.16.9.111
71	3.438062	Dell_35:11:11	Broadcast	ARP	60	Who has 172.16.8.118? Tell 172.16.10.195
▶ Frame 119: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0 ▶ Ethernet II, Src: IntelCor_13:ed:7c (00:27:0e:13:ed:7c), Dst: RealtekS_b2:60:90 (00:e0:4c:b2:60:90) ▶ Address Resolution Protocol (reply)						
0000	00 e0 4c b2 60 90 00 27 0e 13 ed 7c 08 06 00 01	..L....				
0010	08 00 06 04 00 02 00 27 0e 13 ed 7c ac 10 09 60				
0020	00 e0 4c b2 60 90 ac 10 09 6a	..L....j				

4. Create a Filter to display only DNS packets and provide the flow graph.

Procedure

- Select Local Area Connection in Wireshark.
- Go to capture ☐ option
- Select stop capture automatically after 100 packets.
- Then click Start capture.
- Search DNS packets in the search bar.
- To see flow graph click Statistics ☐ Flow graph.
- Save the packets.



5. Create a Filter to display only HTTP packets and inspect the packets

Procedure

- Select Local Area Connection in Wireshark.
- Go to capture ☐ option
- Select stop capture automatically after 100 packets.
- Then click Start capture.
- Search HTTP packets in the search bar.
- Save the packets.

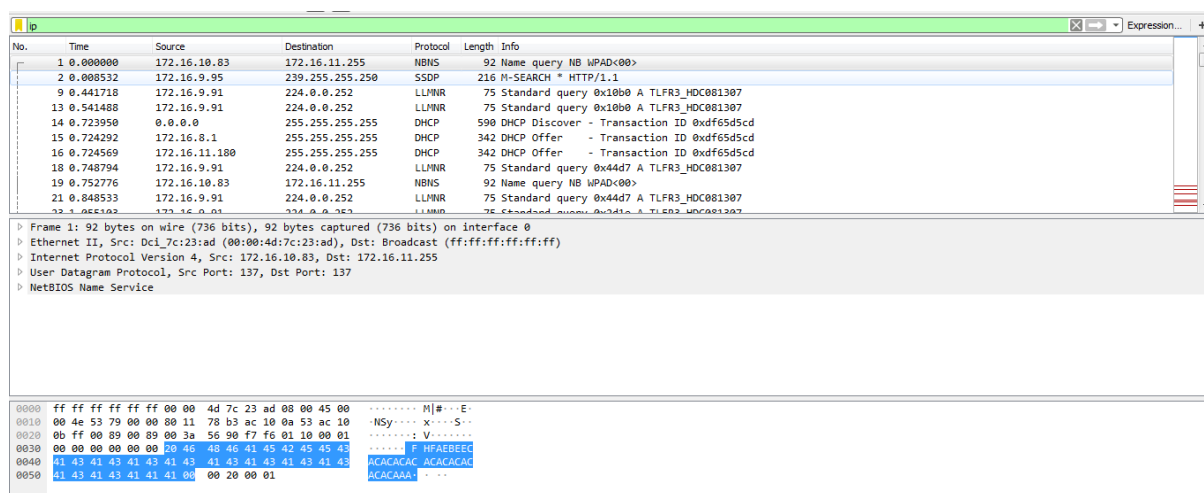
The screenshot shows the Wireshark interface with the following details:

- Packet List:** A table of captured packets. Packet 357 is selected, showing it is an HTTP packet of 5480 bytes.
- Packet Details:**
 - Frame 357: 5480 bytes on wire (43840 bits), 5480 bytes captured (43840 bits) on interface 0
 - Ethernet II, Src: IntelCor_13:ed:7c (00:27:0e:13:ed:7c), Dst: Dci_7c:23:64 (00:00:4d:7c:23:64)
 - Internet Protocol Version 4, Src: 172.16.9.96, Dst: 172.16.18.96
 - Transmission Control Protocol, Src Port: 2869, Dst Port: 49320, Seq: 11420, Ack: 574, Len: 5426
 - [2 Reassembled TCP Segments (5615 bytes): #356(189), #357(5426)]
 - Hypertext Transfer Protocol
 - Portable Network Graphics
- Packet Bytes:** A hex dump of the packet data, showing the start of a PNG image header.

6. Create a Filter to display only IP/ICMP packets and inspect the packets.

Procedure

- Select Local Area Connection in Wireshark.
- Go to capture ☐ option
- Select stop capture automatically after 100 packets.
- Then click Start capture.
- Search ICMP/IP packets in the search bar.
- Save the packets



7. Create a Filter to display only DHCP packets and inspect the packets.

Procedure

- Select Local Area Connection in Wireshark.
- Go to capture ☐ option
- Select stop capture automatically after 100 packets.
- Then click Start capture.
- Search DHCP packets in the search bar.
- Save the packets

Output

No.	Time	Source	Destination	Protocol	Length	Info
3	0.209593	fe80::5cda:e1f4:d07...	ff02::1:2	DHCPv6	150	Solicit XID: 0x3634f7 CID: 000100010d2a7d3800004d7c6d3e
4	0.310897	fe80::d40:4448:5018...	ff02::1:2	DHCPv6	153	Solicit XID: 0x0084a1 CID: 0001000103c2673c00e04c552cb8
7	0.383092	fe80::5147:5612:a48...	ff02::1:2	DHCPv6	158	Solicit XID: 0xa70c07 CID: 00010001215e53494d3d7ec887b3
11	0.524560	fe80::f821:1889:190...	ff02::1:2	DHCPv6	156	Solicit XID: 0xedfea3 CID: 000100012153c05f00270e13f6d9
31	1.215571	fe80::5cda:e1f4:d07...	ff02::1:2	DHCPv6	150	Solicit XID: 0x3634f7 CID: 000100010d2a7d3800004d7c6d3e
85	3.228007	fe80::5cda:e1f4:d07...	ff02::1:2	DHCPv6	150	Solicit XID: 0x3634f7 CID: 000100010d2a7d3800004d7c6d3e
96	3.649834	fe80::c159:d7a6:a74...	ff02::1:2	DHCPv6	156	Solicit XID: 0x8236e6 CID: 00010001210226a5001fd0e2a391
109	4.096915	fe80::c4b5:5e2c:922...	ff02::1:2	DHCPv6	156	Solicit XID: 0xd02548 CID: 0001000121d5ad5200270e13eec4
118	4.311356	fe80::d40:4448:5018...	ff02::1:2	DHCPv6	153	Solicit XID: 0x0084a1 CID: 0001000103c2673c00e04c552cb8
211	7.240392	fe80::5cda:e1f4:d07...	ff02::1:2	DHCPv6	150	Solicit XID: 0x3634f7 CID: 000100010d2a7d3800004d7c6d3e

Frame 3: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits) on interface 0
 Ethernet II, Src: Giga-Byt_7c:6d:3e (00:1a:4d:7c:6d:3e), Dst: IPv6mcast_01:00:02 (33:33:00:01:00:02)
 Internet Protocol Version 6, Src: fe80::5cda:e1f4:d07e:7544, Dst: ff02::1:2
 User Datagram Protocol, Src Port: 546, Dst Port: 547
 DHCPv6

```

0000  33 33 00 01 00 02 00 1a 4d 7c 6d 3e 8e dd 60 00  33.....M|>...
0010  00 00 00 60 11 01 fe 80 00 00 00 00 00 00 5c da  .....\
0020  e1 f4 d0 7e 75 44 ff 02 00 00 00 00 00 00 00 00  ..uD...
0030  00 00 00 01 00 02 02 22 02 23 00 60 7d b0 01 36  ....*#]...6
0040  34 f7 00 00 00 02 00 00 00 01 00 0e 00 01 00 01  4.....M| m>...
0050  0d 2a 7d 38 00 00 4d 7c 6d 3e 00 03 00 0c 0e 00  ..*)s--M| m>...
0060  00 4d 00 00 00 00 00 00 00 00 27 00 0a 00 08  ..M.....
0070  c1 54 6d 69 6e 2d 50 43 00 10 00 0e 00 00 01 37  admin-PC.....7
0080  00 00 4d 53 46 54 20 35 2e 30 00 06 00 00 00 18  ....MSFT 5.0....
0090  00 17 00 11 00 27  .....
```

RESULT

Thus the program for capturing and analyzing packets using Wireshark tool is successfully executed.

EX NO :9**ANALYZE WEB LOGS USING WEBALIZER****Date:****Aim**

To analyze the different types of web logs using Webalizer tool.

Description

This Manual is intended to provide the necessary background and insight to how web server analysis works, things to look for and things to watch out for. Specifically, this Manual is intended for the users of the Webalizer.

You have a website and you want to know if anybody is looking at it, and if so, what they are looking at and how many times. Lucky for you, (most) every web server keeps a log of what it's doing, so you can just go look and see. The logs are just plain ASCII text files, so any text editor or viewer would work just fine. Each time someone (using a web browser) asks for one of your web pages, or any component thereof (known as URLs, or *Uniform Resource Locator*), the web server will write a line to the end of the log representing that request. Unfortunately, the raw logs are rather cryptic for everyday humans to read. While you might be able to determine *if* anybody was looking at your web site, any other information would require some sort of processing to determine

A typical log entry might look something like the following:

192.168.45.13 - - 21/NOv/2018:11:20:39 -0400] "GET /mypage.html HTTP/1.1" 200 117

This represents a request from a computer with the IP address 192.168.45.13 for the URL **/mypage.html** on the web server. It also gives the time and date the request was made, the type of request, the result code for that request and how many bytes were sent to the remote browser. There will be a line similar to this one for each and every request made to the web server over

the period covered by the log. A '**Hit**' is another way to say '*request made to the server*', so as you may have noticed, each line in the log represents a '**Hit**'. If you want to know how many Hits your server received, just count the number of lines in the log. And since each log line represents a request for a *specific* URL, from a *specific* IP address, you can easily figure out how many hits you got for each of your web pages or how many hits you received from a particular IP address by just counting the lines in the log that contain them. Yes, it really is that simple. And while you *could* do this manually with a text editor or other simple text processing tools, it is much more practical and easier to use a program specifically designed to analyze the logs for you, such as the Webalizer. They take the work out of it for you, provide totals for many other aspects of your server, and allow you to visualize the data in a way not possible by just looking at the raw logs.

This represents a request from a computer with the IP address 192.168.45.13 for the URL **/mypage.html** on the web server. It also gives the time and date the request was made, the type of request, the result code for that request and how many bytes were sent to the remote browser. There will be a line similar to this one for each and every request made to the web server over the period covered by the log A typical server/browser interaction might

go something like this:

- The web browser asks for the URL **mypage.html**.
- The server sees the request and sends back the HTML page.
- The web browser notices that there are two inline graphic links in the HTML page, so it asks for the first one, **myimage1.jpg**.
- The server sees the request and sends back the graphic image.
- The web browser then asks for the second image, **myimage2.jpg**.
- The server sees the request and sends back the graphic image.
- The browser displays the web page and graphics for the user.

In the web server log, the following lines would be added:

192.168.45.13 - - [24/May/2005:11:20:39 -0400] "GET /mypage.html HTTP/1.1" 200 117

192.168.45.13 - - [24/May/2005:11:20:40 -0400] "GET /myimage1.jpg HTTP/1.1" 200 231

192.168.45.13 - - [24/May/2005:11:20:41 -0400] "GET /myimage2.jpg HTTP/1.1" 200 432

So what can we gather from this exchange? Well, based on the what we learned above, we can count the number of lines in the log file and determine that the server received 3 hits during the period that this log file covers. We can also calculate the number of hits each URL received (in this case, 1 hit each). Along the same lines, we can see that the server received 3 hits from the IP address 192.168.45.13, and when those requests were received. The two numbers at the end of each line represent the *response code* and the *number of bytes* sent back to the requestor. The response code is how the web server indicates how it handled the request, and the codes are defined as part of the HTTP protocol. In this example, they are all 200, which means everything went OK. One response code you may be very familiar with is the all too common '**404 - Not Found**', which means that the requested URL could not be found on the server. There are several other response codes defined, however these two are the most common.

And that, in a nutshell, is about all you can accurately determine from the logs. "But wait!" you might be screaming, "most analysis program have lots of other numbers displayed!", and you would be right. Some more obscure numbers can be calculated, like the number of different response codes, number of hits within a given time period, total number of bytes sent to remote browsers, etc.. Other numbers can be implied based on certain assumptions, however those cannot be considered entirely accurate, and some can even be wildly inaccurate. Other log formats might be used by a web server as well, which provide additional information above what the CLF format does, and those will be discussed shortly. For now, just realize that the only thing you can really, accurately determine is what IP address requested which URL, and when it requested that URL, as shown in the example above.

The Good, the Bad and the Ugly

So now you have a good grasp of how your web server works and what information can be obtained from its logs, like number of hits (to the server and to individual URLs), number of IP addresses making the requests (and how many hits each IP address made), and when those requests were made. Given just that information, you can answer questions such as "What is the most popular URL on my site?", "What was the next most popular URL?", "What IP address

made the most requests to my server?", and "How busy was my server during this time period?". Most analysis programs will also make it easy to answer such questions as "What time of day is my web server the most active?", or "What day of the week is the busiest?". They can give you an insight into usage patterns that may not be apparent by just looking at the raw logs. All of these questions can be answered with completely accurate answers, based just on the simple analysis of your web server logs. That's the good news!

The bad news? Well, with all the things you can determine by looking at your logs, there are a lot of things you can't accurately calculate. Unfortunately, some analysis programs lead you to believe otherwise, and forget to mention (particularly commercial packages) that these are not much more than assumptions and cannot be considered at all accurate. Like what? You ask. Well, how about those things that some programs call 'user trails' or 'paths', that are supposed to tell you what pages and in what order a user travelled through your site. Or how about the length of time a user spends on your site. Another less than accurate metric would be that of 'visits', or how many users 'visited' your site during a given time period. All of these cannot be accurately calculated, for a couple of different reasons.. lets look at some of them:

The HTTP protocol is stateless

In a typical computer program that you run on your own machine, you can always determine what the user is doing. They log in, do some stuff, and when finished, they log out. The HTTP protocol however is different. Your web server only sees requests from some remote IP address. The remote address connects, sends a request, receives a response and then disconnects. The web server has no idea what the remote side is doing between these requests, or even what it did with the response sent to it. This makes it impossible to determine things like how long a user spends on your site. For example, if an IP address makes a request to your server for your home page, then 15 minutes later makes a request for some other page on your site, can you determine how long the user had been at your site? The answer is of course **No!**. Just because 15 minutes expired between requests, you have no idea what the remote address was doing between those two requests. They could have hit your site, then immediately gone somewhere else on the web, only to come back 15 minutes later to request another page. Some analysis packages will say that the user stayed on your site for at least 15 minutes plus some 'fudge' time for viewing the last page requested (like 5 minutes or so). This is actually just a guess, and nothing more.

You cannot determine individual users

Web servers see requests and send results to IP addresses only. There is no way to determine *what* is at that address, only that some request came from it. It could be a real person, it could be some program running on a machine, or it could be lots of people all using the same IP address (more on that below). Some of you will note that the HTTP protocol *does* provide a mechanism for user authentication, where a username and password are required to gain access to a web site or individual pages. And while that is true, it isn't something that a normal, public web site uses (otherwise it wouldn't be public!). As an example, say that one IP address makes a request to your server, and then a minute later, some other IP address makes a request. Can you say how many people visited your site? Again, the answer is **No!**. One of those requests may have come from a search engine 'spider', a program designed to scour the web looking for links and such. Both requests could have been from the same user, but at different addresses. Some analysis program will try to determine the number of users based on things like IP address plus browser type, but even so, these are nothing more than guesses made on some rather faulty assumptions.

Network topology makes even IP addresses problematic

In the good old days, every machine that wanted to talk on the internet had it's own unique IP address. However, as the internet grew, so did the demand for addresses. As a result, several methods of connecting to the internet were developed to ease the addressing problem. Take, for example, a normal dial-up user sitting at home. They call their service provider, the machines negotiate the connection, and an IP address is assigned from a re-usable 'pool' of IP addresses that have been assigned to the provider. Once the user disconnects, that IP address is made available to other users dialing in. The home user will typically get a different IP address each time they connect, meaning that if for some reason they are disconnected, they will re-connect and get a new IP address. Given this situation, a single user can appear to be at many different IP addresses over a given time. Another typical situation is in a corporate environment, where all the PCs in the organization use private IP addresses to talk on the network, and they connect to the internet through a gateway or firewall machine that translates their private address to the public one the gateway/firewall uses. This can make all the users within the organization appear as if they were all using the same IP address. Proxy servers are similar, where there can be thousands of users, all appearing to come from the same address. Then there are reverse-proxy servers, typical of many large providers such as AOL, that can make a single machine appear to use many different IP addresses while they are connected (the reverse-proxy keeps track of the addresses and translates them back to the user). Given this situation, can you say how many users

visited your site if your logs show 10 requests from the same IP address over an hour? Again, the answer is **No!**. It could have been the same user, or it could have been multiple users sitting behind a firewall. Or how about if your logs show 10 requests from 10 different IP addresses? Think it was from 10 different users? Of course not. It could have been 10 different users, could have been a couple of users sitting behind a reverse proxy, could have been one or more users along with a search engine 'spider', or it could be any combination of them all.

But wait there's more!

Ok, so what have we learned here? Well, in short, you don't know who or what is making requests to your server, and you can't assume that a single IP address is really a single user. Sure, you can make all kinds of assumptions and guesses, but that is all they really are, and you should not consider them at all accurate. Take the following example; IP address A makes a request to your server, 1 minute later, IP address B makes a request, and then 10 minutes later, address A makes another request. What can we determine from that sequence? Well, we can assume that two users visited. But what if address A was that of a firewall? Those two requests from address A could have been two different users. What if the user at address A got disconnected and dialed back in, getting a different address (address B) and someone else dialed in at the same time and got the now free address A? Or maybe the user was sitting behind a reverse-proxy, and all three requests were really from the same user. And can we tell what 'path' or 'trail' these users took while at the web site or how long they remained? Hopefully, you should now see that the answer to all these things is a big resounding **No!** we can't! Without being able to identify individual unique users, there is no way to tell what an individual unique user does. All is not lost however. Over time, people have come up with ways to get around these limitations. Systems have been written to get around the stateless nature of the HTTP protocol. Cookies and other unique identifiers have been used to track individuals, as has various dynamic pages with back end databases. However, these things are all, for the most part, external to the protocol, not logged in a standard web server log, and require specialized tools to analyze. In all other cases, any programs that claim to analyze these types of metrics should just be considered guesses based on certain assumptions. One such example can be found within the Webalizer itself. The concept of a 'visit' is a metric that cannot be accurately reported, yet that is one of the things that the Webalizer does show. It was added because of the huge number of requests received from individuals using the program. It is based on the assumption that a single IP address represents a single user. You have already seen how this assumption falls flat in the real world, and if you read through the documentation provided

with the program, you will see that it clearly says the 'visit' numbers (along with 'entry' and 'exit' pages) are not to be considered accurate, but more of a rough guess. We haven't touched on entry and exit pages yet, but they are based on the concept of a 'visit', which we have already seen isn't accurate. These are supposed to be the first and last page a user sees while at the web site. If a request comes in that is considered a new 'visit', then the URL of that request would be, in theory, the 'Entry' page to the site. Likewise, the last URL requested in a visit would be the 'Exit' page. Similar to user 'paths' or 'trails', and being based on the 'visit' concept, they are to be treated with the same caution. One of the funniest metrics I have seen in one particular analysis program was supposed to tell you where the user was geographically, based on where the domain name of the requesting remote address was registered. Clever idea, but completely worthless. Take for example AOL, which is registered in Virginia. The program considered all AOL users as living in Virginia, which we know is not the case for a provider with access points all over the globe.

Other metrics you CAN determine

Now that you have seen what is possible, you may be thinking that there are some other things these programs display, and wondering about how accurate they might be. Hopefully, based on what you have already seen thus far, you should be able to figure them out on your own. One such metric is that of a 'page' or 'page view'. As we already know, a web page is made up of an HTML text document and usually other elements such as graphic images, audio or other multimedia objects, style sheets, etc.. One request for a web page might generate dozens of requests for these other elements, but a lot of people just want to know how many web pages were requested without counting all the stuff that makes them up. You can get this number, if you know what type of files you may consider a 'page'. In a normal server, these would be just the URLs that end with a .htm or .html extension. Perhaps you have a dynamic site, and your web pages use an .asp, .pl or .php extension instead. You obviously would not want to count .gif or .jpg images as pages, nor would you want to count style sheets, flash graphic and other elements. You could go through the logs and just count up the requests for whatever URL meets your criteria for a 'page', but most analysis programs (including the Webalizer) allows you to specify what you consider a page and will count them up for you.

Other information

Up to now, we have just discussed the CLF (Common Log Format) log format. There are others. The most common is called 'combined', and takes the basic CLF format and adds two new pieces

of information. Tacked on the end is the 'user agent' and 'referrer'. A user agent is just the name of the browser or program being used to generate the request to the web server. The 'referrer' is supposed to be the page that *referred* the user to your web server. Unfortunately, both of these can be completely misleading. The user agent string can be set to anything in some modern browsers. One common trick for Opera users is to set their user agent string to that of MS Internet Explorer so they can view sites that only allow MSIE visitors. And the referrer string, according to the standards document (RFC) for the HTTP protocol, may or may not be used at the browsers choosing, and if used, does not have to be accurate or even informative. The apache web server (which is the most common on the internet) allows other things to be logged, such as cookie information, length of time to handle the request and lots of other stuff. Unfortunately, the inclusion and placement of this information in the server logs are not standard. Another format, developed by the W3C (world wide web consortium), allows log records to be made up of many different pieces of information, and their location can be anywhere in the log entry with a header record needed to map them. Some analysis programs handle these and other formats better than others.

Analysis techniques

The only true way to get an accurate picture of what your web server is doing is to look at it's logs. This is how most of the analysis packages out there get their information, and is the most accurate. Other methods can be used, with different results. One common method, which was widely popular for a while, was the use of a 'page counter'. Basically, it was a dynamic bit included in a web page that increment a counter and displayed it's value each time the page was requested. Normally, it was included in the page as if it were a standard image file. One problem with this method was that you had to include a different 'image' file for each page you wanted to track. Another problem occurred if the remote user had image display turned off in their browser, or could not display images at all (such as in a text based web browser). You could also easily inflate the number by just hitting the 'reload' button on your browser over and over again. Similar methods were developed using java and javascript, in an attempt to get even more information about the visiting browser, such as screen resolution and operating system type. Of course, these can easily be circumvented as well. Some companies set up systems that claim to track your server usage remotely, by including an image or javascript element on your site which would then contact the companies system each time the image or javascript element was requested. These all have the same problems and limitations. In all of these, you can simply turn off images

and java/javascript and then browse the web site completely uncoded and unseen (except in the web server logs). Beware of these types of counters and remote usage sites, they are not quite as accurate as they may lead you to believe.

Conclusion

It should now be obvious that there are only certain things you can determine from a web server log. There are some completely accurate numbers you can generate without question. And then, there are some wildly inaccurate and misleading numbers you can garner depending on what assumptions you make. Want to know how many requests generated a 404 (not found) result? Go right ahead and count them up and be completely confident with the number you get. Want to know how many 'users' visited your website? Good luck with that one. Unless you go 'outside the logs', it will be a hit or miss stab in the dark. But now you should have a good idea of what is and isn't possible, so when you look at your usage report, you will be able to determine what the numbers mean and how much to trust them. You should also now see that a lot can depend on how the program is configured, and that the wrong configuration can lead to wrong results. Take the example of 'pages'.. if your analysis software thinks that only URLs with a .html or .html extension is a page, and all you have is .php pages on your site, that number will be completely wrong. Not because the program is wrong, but because someone told it the wrong information to base its calculations on. Remember, knowledge is power, so now you have the power to ask the proper questions and get the proper results. The next time you look at a server analysis report, hopefully you will see it in a different light given your new found knowledge.

Webalizer Quick Help

Main Headings

Hits represent the total number of requests made to the server during the given time period (month, day, hour etc..).

Files represent the total number of hits (requests) that actually resulted in something being sent back to the user. Not all hits will send data, such as 404-Not Found requests and requests for pages that are already in the browser's cache.

Tip: By looking at the difference between hits and files, you can get a rough indication of repeat visitors, as the greater the difference between the two, the more people are requesting pages they

have already cached (have viewed already).

Sites is the number of unique IP addresses/hostnames that made requests to the server. Care should be taken when using this metric for anything other than that. Many users can appear to come from a single site, and they can also appear to come from many IP addresses so it should be used simply as a rough gauge as to the number of visitors to your server.

Visits occur when some remote site makes a request for a *page* on your server for the first time. As long as the same site keeps making requests within a given timeout period, they will all be considered part of the same **Visit**. If the site makes a request to your server, and the length of time since the last request is greater than the specified timeout period (*default is 30 minutes*), a new **Visit** is started and counted, and the sequence repeats. Since only *pages* will trigger a visit, remote sites that link to graphic and other non- page URLs will not be counted in the visit totals, reducing the number of *false visits*.

Pages are those URLs that would be considered the actual page being requested, and not all of the individual items that make it up (such as graphics and audio clips). Some people call this metric *pageviews* or *page impressions*, and defaults to any URL that has an extension of **.htm**, **.html** or **.cgi**.

A **KByte** (KB) is 1024 bytes (1 Kilobyte). Used to show the amount of data that was transferred between the server and the remote machine, based on the data found in the server log.

Common Definitions

A **Site** is a remote machine that makes requests to your server, and is based on the remote machines IP Address/Hostname.

URL - Uniform Resource Locator. All requests made to a web server need to request *something*. A URL is that *something*, and represents an object somewhere on your server, that is accessible to the remote user, or results in an error (ie: 404 - Not found). URLs can be of any type (HTML, Audio, Graphics, etc...).

Referrers are those URLs that lead a user to your site or caused the browser to request something from your server. The vast majority of requests are made from your own URLs, since most HTML pages contain links to other objects such as graphics files. If one of your HTML pages contains links to 10 graphic images, then each request for the HTML page will produce 10 more

hits with the referrer specified as the URL of your own HTML page.

Search Strings are obtained from examining the referrer string and looking for known patterns from various search engines. The search engines and the patterns to look for can be specified by the user within a configuration file. The default will catch most of the major ones.

Note: Only available if that information is contained in the server logs.

User Agents are a fancy name for *browsers*. Netscape, Opera, Konqueror, etc.. are all **User Agents**, and each reports itself in a unique way to your server. Keep in mind however, that many *browsers allow* the user to change its reported name, so you might see some obvious fake names in the listing.

Note: Only available if that information is contained in the server logs.

Entry/Exit pages are those pages that were the first requested in a visit (**Entry**), and the last requested (**Exit**). These pages are calculated using the **Visits** logic above. When a visit is first triggered, the requested page is counted as an **Entry** page, and whatever the last requested URL was, is counted as an **Exit** page.

Countries are determined based on the *top level domain* of the requesting site. This is somewhat questionable however, as there is no longer strong enforcement of domains as there was in the past. A .COM domain may reside in the US, or somewhere else. An .IL domain may actually be in Israel, however it may also be located in the US or elsewhere. The most common domains seen are .COM (US Commercial), .NET (Network), .ORG (Non-profit Organization) and .EDU (Educational). A large percentage may also be shown as *Unresolved/Unknown*, as a fairly large percentage of dialup and other customer access points do not resolve to a name and are left as an IP address.

The Webalizer GUI is a graphical user interface for the web server log file analysis program **Webalizer**. By analyzing the web server log files this program produces yearly, monthly, daily and hourly usage statistics of a website, along with the ability to display usage by site, URL, referrer, user agent (browser), username, search strings, entry/exit pages, and country. but as Webalizer has to be configured by a configuration file and run from a command line prompt, this graphical user interface for Windows and Linux was created. The Webalizer GUI allows the easy

configuration of most of the Webalizer options by standard graphical user elements. Afterwards the GUI executes the Webalizer, which produces the usage statistics in HTML format for viewing with a browser.

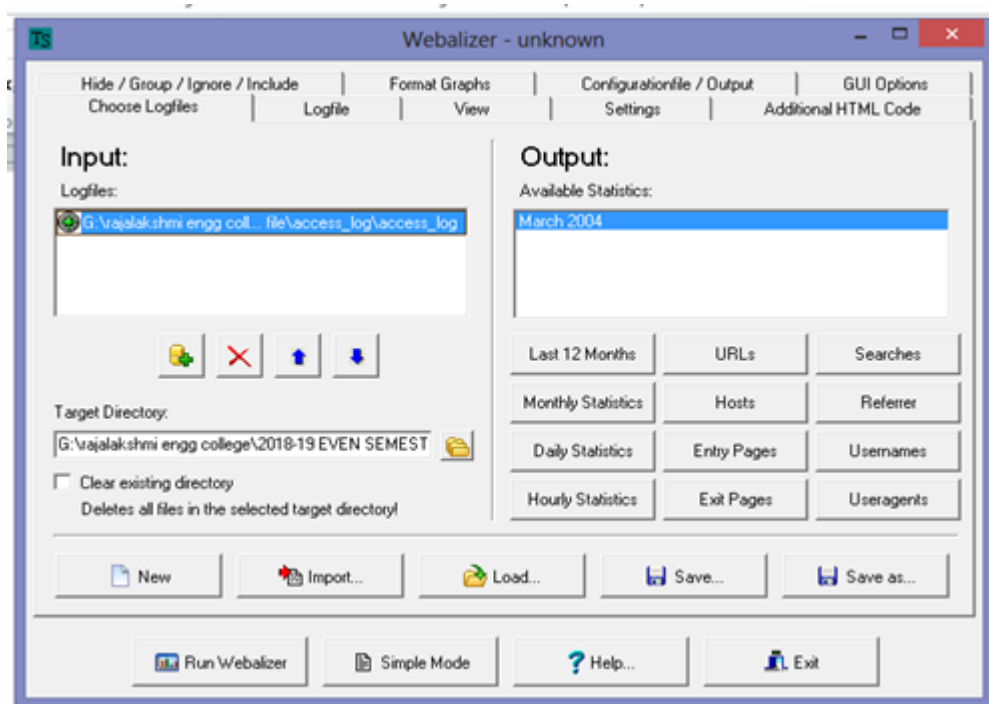
The 'plus'-download packages of the GUI already contain the Stone Steps Version of the Webalizer and are therefore recommended for people who have never worked with the Webalizer before. But you can also use the GUI with another version of the Webalizer and even import an existing Webalizer configuration file.

Running steps

Step1: Run webalizer windows version

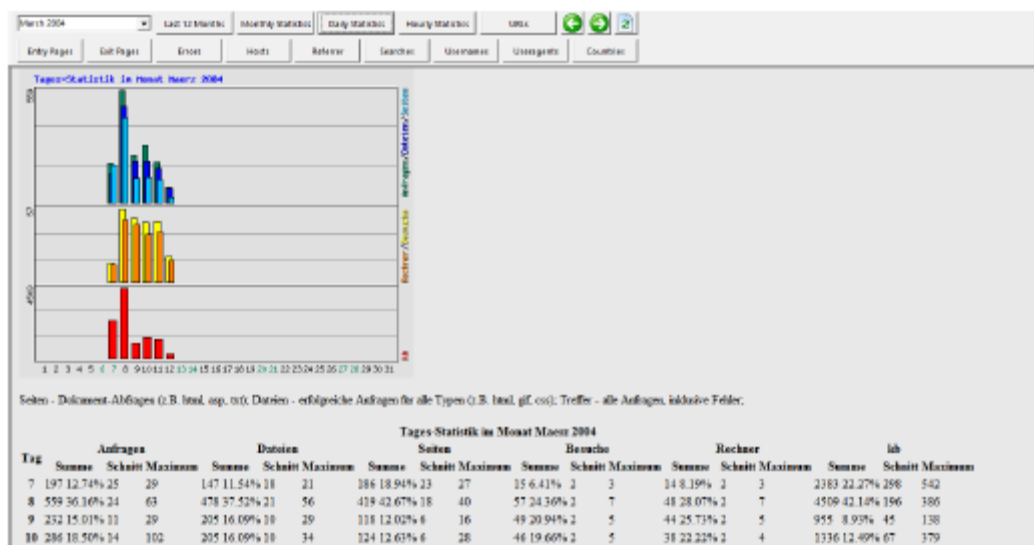
Step2. Input web log file (download from web)

Step3: Press Run webalizer



Output:

Monthly statistics



Hosts

March 2004 Last 12 Months Monthly Statistics Daily Statistics Hourly Statistics URLs

Entry Pages Exit Pages Errors Hosts Referrer Searcher Usernames Useragents Countries

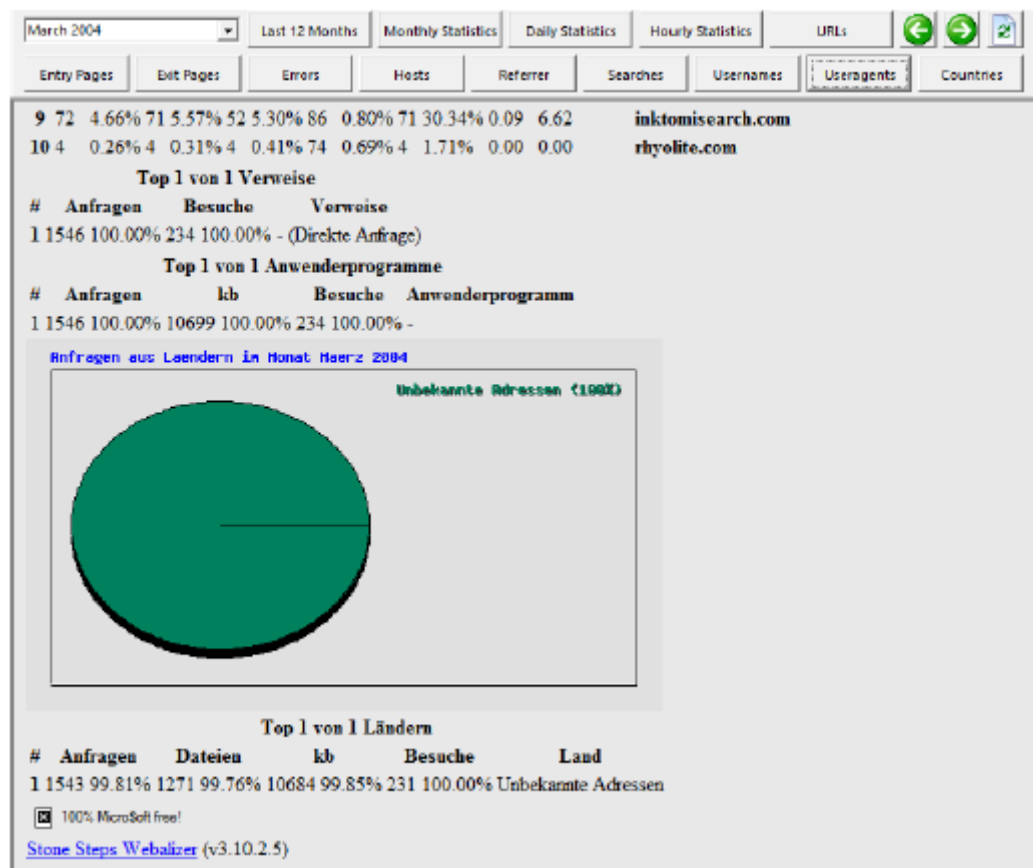
Top 20 von 171 Rechnern (IP-Adressen)

#	Anfragen	Dateien	Seiten	lb	Besuche	Dauer	Land	Rechnername
1	100	6.47%	83	6.51%	46	4.68%	472	4.41%
2	72	4.66%	71	5.57%	52	5.30%	86	0.80%
3	47	3.04%	41	3.22%	43	4.38%	613	5.73%
4	44	2.85%	42	3.30%	23	2.34%	244	2.28%
5	35	2.26%	29	2.28%	14	1.43%	218	2.03%
6	29	1.88%	28	2.20%	14	1.43%	97	0.91%
7	23	1.49%	14	1.10%	10	1.02%	135	1.26%
8	22	1.42%	22	1.73%	8	0.81%	67	0.63%
9	19	1.23%	19	1.49%	7	0.71%	61	0.57%
10	19	1.23%	11	0.86%	10	1.02%	51	0.48%
11	15	0.97%	14	1.10%	13	1.32%	130	1.22%
12	13	0.84%	13	1.02%	13	1.32%	120	1.12%
13	13	0.84%	13	1.02%	1	0.10%	28	0.26%
14	13	0.84%	13	1.02%	2	0.20%	30	0.28%
15	12	0.78%	11	0.86%	9	0.92%	68	0.63%
16	12	0.78%	12	0.94%	1	0.10%	27	0.25%
17	12	0.78%	12	0.94%	1	0.10%	27	0.25%
18	11	0.71%	11	0.86%	7	0.71%	41	0.38%
19	10	0.65%	9	0.71%	7	0.71%	41	0.38%
20	10	0.65%	10	0.78%	2	0.20%	61	0.57%

Top 10 von 171 Rechnern (IP-Adressen) sortiert nach lb

#	Anfragen	Dateien	Seiten	lb	Besuche	Dauer	Land	Rechnername
1	47	3.04%	41	3.22%	43	4.38%	613	5.73%
2	100	6.47%	83	6.51%	46	4.68%	472	4.41%
3	44	2.85%	42	3.30%	23	2.34%	244	2.28%

User-agents



RESULT:-

Thus the analyze web logs using webalizer is executed successfully

Ex No: 10**STUDY THE DIFFERENT KINDS OF CABLES****Date****AIM:**

To study the different kinds of cables.

DESCRIPTION:

Students are instructed to do the following,

Study the following cables and list their specifications based on the questions followed.

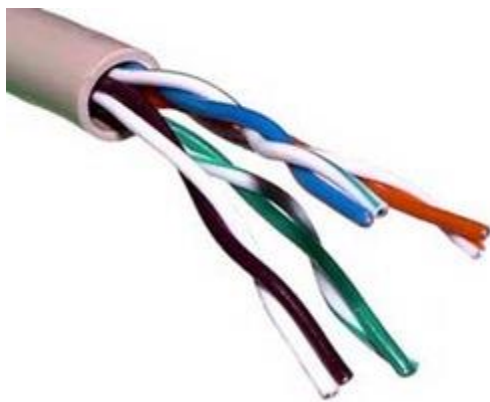
1. Unshielded Twisted Pair (UTP) Cable
2. Shielded Twisted Pair (STP) Cable
3. Coaxial Cable
4. Fiber Optic Cable
5. Wireless LANs
6. USB Cables
7. Crossover Cables

1. UNSHIELDED TWISTED PAIR (UTP) CABLE

UTP stands for Unshielded Twisted Pair cable. UTP cable is a 100 ohm copper cable that consists of 2 to 1800 unshielded twisted pairs surrounded by an outer jacket. They have no metallic shield. This makes the cable small in diameter but unprotected against electrical interference. The twist helps to improve its immunity to electrical noise and EMI.

Local Area Networks (LAN) also make use of twisted pair cables. **They can be used for both analog and digital transmission**

UTP cables are mostly used for LAN networks. They can be used for voice, low-speed data, high-speed data, audio and paging systems, and building automation and control systems. UTP cable can be used in both the horizontal and backbone cabling subsystems.



Inside a UTP cable is up to four twisted pairs of copper wires, enclosed in a protective plastic cover, with the greater number of pairs corresponding to more bandwidth. The two individual

wires in a single pair are twisted around each other, and then the pairs are twisted around each other, as well.

Advantages of the UTP:

It is a less costly and less expensive unshielded wire from another network medium. It is designed to reduce crosstalk, RFI, and EMI. Its size is small, and hence the installation of the UTP is easier. It is mostly useful for short-distance network connections like home and small organizations.

Disadvantage of the UTP:

It can only be used in length segment up to 100 meters. It has limited bandwidth for transmitting the data. It does not provide a secure connection for data transmitting over the network.

2. SHIELDED TWISTED PAIR (STP)

Shielded twisted pair (STP) cable was originally designed by IBM for token ring networks that include two individual wires covered with a foil shielding, which prevents electromagnetic interference, thereby transporting data faster.

STP is similar to unshielded twisted pair (UTP); however, it contains an extra foil wrapping or copper braid jacket to help shield the cable signals from interference. STP cables are costlier when compared to UTP, but has the advantage of being capable of supporting higher transmission rates across longer distances.

UTP cables are popular all over the world due to their low cost, ease of installation and flexibility. They also support high data transfer speeds of up to 1 gigabit per second (Gbps) for transmission distances of up to 100 meters (m).

They can be used for both analog and digital transmission.

Shielded Twisted Pair (STP)



Advantages and Disadvantages of Twisted Pair Cable

Advantages

- Inexpensive and readily available
- Flexible and light weight
- Easy to set up and install

Disadvantages

- Susceptible to interference and noise
 - Noise is an electrical disturbance that can degrade communications
- Low bandwidth

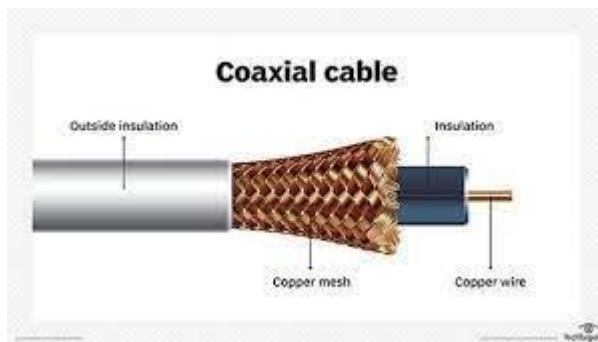
3.COAXIAL CABLE

Coaxial cable is a type of copper cable specially built with a metal shield and other components engineered to block signal interference. It is primarily used by cable TV companies to connect their satellite antenna facilities to customer homes and businesses.

Coaxial cable is a type of transmission line, used to carry high-frequency electrical signals with low losses. It is used in such applications as telephone trunk lines, broadband internet networking cables, high-speed computer data busses, cable television signals, and connecting radio transmitters and receivers to their antennas. It differs from other shielded cables because the dimensions of the cable and connectors are controlled to give a precise, constant conductor spacing, which is needed for it to function efficiently as a transmission line. It is commonly used by cable operators, telephone companies, and internet providers worldwide to convey data, video, and voice communications to customers. It has also been

used extensively within homes.

Broadband coaxial cable supports the frequency range above 4kHz and are used for analog signals



Advantages & Disadvantage of Coaxial Cable

Advantages:

- High transmission rate
- Higher noise immunity

Disadvantages:

- comparatively costly
- security problem

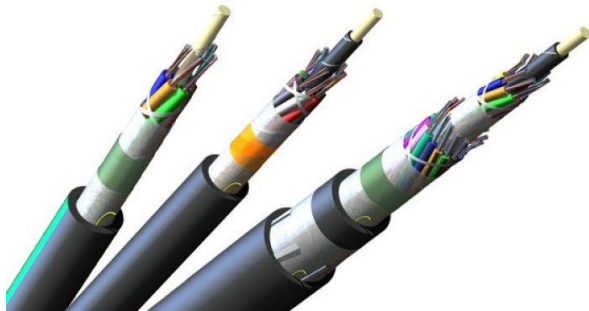
4.FIBER OPTIC CABLE

A fiber-optic cable, also known as an optical-fiber cable, is an assembly similar to an electrical cable, but containing one or more optical fibers that are used to carry light. The optical fiber elements are typically individually coated with plastic layers and contained in a protective tube suitable for the environment where the cable is used. Different types of cable^[1] are used for different applications, for example, long distance telecommunication, or providing a high-speed data connection between different parts of a building.

For indoor applications, the jacketed fiber is generally enclosed, together with a bundle of flexible fibrous polymer *strength members* like aramid (e.g. Twaron or Kevlar), in a lightweight plastic cover to form a simple cable. Each end of the cable may be terminated with a specialized optical fiber connector to allow it to be easily connected and disconnected from

transmitting and receiving equipment.

Since fiber optic data transmissions in networking use square waves, it is a **digital signal**.



Advantages and Disadvantages of fibre-Optic Cables	
Advantages	Disadvantages
<ul style="list-style-type: none"> • Able to carry significantly more signals than wire • Faster data transmission • Less susceptible to noise from other devices • Better security for signals during transmission • Smaller physical size 	<ul style="list-style-type: none"> • Costs more than twisted pair and coaxial cable • Can be difficult to install and modify • More expensive over shorter distances

5.WIRELESS LAN (WLAN)

A **wireless LAN (WLAN)** is a wireless computer network that links two or more devices using wireless communication to form a local area network (LAN) within a limited area such as a home, school, computer laboratory, campus, or office building. This gives users the ability to move around within the area and remain connected to the network. Through a gateway, a WLAN can also provide a connection to the wider Internet.

Wireless LANs based on the IEEE 802.11 standards are the most widely used computer networks in the world. These are commonly called Wi-Fi, which is a trademark belonging to the Wi-Fi Alliance. They are used for home and small office networks that link together laptop computers, printers, smartphones, Web TVs and gaming devices with a wireless router, which links them to the internet. Hotspots provided by routers at restaurants, coffee shops, hotels, libraries, and airports allow consumers to access the internet with portable wireless devices.



WLAN ADVANTAGES AND DISADVANTAGES

Advantages	Disadvantages
The wireless network does not require any cable or wires, and hence communication is possible even when the user is moving.	The range of a wireless network is minimal, and it causes problems for many users.
Wireless networks can be easily extended to places where wires and cables are not accessible.	People who are inexperienced in the computer field may face trouble installing a wireless network.
Installing a wireless network is easier and faster.	The wireless network is very prone to interference, and hence, fog and radiation can cause it to malfunction.
Wireless networks require a one-time investment and, hence, are cheaper.	The cost of installing a wireless network is prohibitive.
Improved and better communication is available if one is using the wireless network.	The wireless network has minimal bandwidth.

6.USB

Universal Serial Bus (USB) is an industry standard that establishes specifications for cables, connectors and protocols for connection, communication and power supply (interfacing) between computers, peripherals and other computers.^[2] A broad variety of USB hardware exists, including 14 different connector types, of which USB-C is the most recent and the only one not currently deprecated.

First released in 1996, the USB standards are maintained by the USB Implementers Forum (USB-IF). The four generations of USB are: USB 1.x, USB 2.0, USB 3.x, and USB4.^[3]



Benefits and Advantages of USB Flash Drives in Points

- *Flash drives or USB drives are solid-state drives that are less susceptible and vulnerable to damage from drop or shock compared to traditional hard disk drives.*
- *Flash drives are smaller and compact than other types of data storage devices.*
- *Modern flash drives carries huge storage space up to 128 GB.*
- *They are used in back up of hard disk.*
- *USB 3.0 devices can transfer data and information at the rate up to 4.8 GB per second which is relatively faster than other devices.*
- *Even faster hard drives reach maximum speeds of 150 MB.*
- *USB flash drives can be easily connected to computer through USB ports.*
- *In many laptops, notebooks, and desktops, there are USB ports available additionally to connect more devices.*

Drawbacks and Disadvantages of USB Flash Drives

- *Since USB drives are used in different computers, if one of the computers becomes infected, it could be transferred the infection to other computers. The files on the drives would become unreadable; the entire drive would become useless.*
- *USB drives do not have an unlimited lifespan as there are limits to the number of reads and write cycles. It is about 100000 cycles after that, the USB drive will fail.*
- *The flash manufacturer has updated software since some USB drives have no encryption. This type of vulnerability can compromise your personal data.*
- *Since the flash drives are small, there is risk of being theft.*
- *Some of the USB Drives are manufactured with low quality and cheaper components which results in low quality USB drives.*
- *USB drives are less weak when it comes to mechanical damages. It is completely free of it; the USB plug that can be worn and tear or even bend is completely useless if it is bending the drive.*
- *USB flash drives are small in size and can be easily misplaced.*
- *They can easily spread viruses, malwares, from one computer to other or in networks.*
- *Before using USB drives it is almost mandatory to scan them prior use to eliminate the chances of spreading infection.*

7.CROSSOVER CABLES

An **Ethernet crossover cable** is a crossover cable for Ethernet used to connect computing devices together directly. It is most often used to connect two devices of the same type, e.g. two computers (via their network interface controllers) or two switches to each other. By contrast, *straight through* patch cables are used to connect devices of *different types*, such as a computer to a network switch.

Intentionally crossed wiring in the crossover cable connects the transmit signals at one end to the receive signals at the other end.

Many network devices today support auto MDI-X (aka "auto crossover") capability, wherein a patch cable can be used in place of a crossover cable, or vice versa, and the receive and transmit signals are reconfigured automatically within the device to yield a working connection.



Advantages and Disadvantages

○ Advantages:

- its ability to run on many different media types
- its efficient use of bandwidth
- its stable behavior during high load times
- its deterministic nature
- its priority scheme, which enables nodes with high-priority data to reserve the network

○ Disadvantages:

- need special recovery procedures when network fails
- difficulty in configuring new hosts to an established LAN
- in priority scheduling, the susceptibility of low priority nodes to increased delays in accessing the network.

RESULT

Thus the study of different cable is executed successfully executed

Ex No: 11 SETING UP A LOCAL AREA NETWORK USING A SWITCH**Date****AIM:**

To study and perform the setting up of a LAN using SWITCH.

DESCRIPTION:**SETTING UP A LAN:**

Creating and configuring a LAN consists of these steps:

1. **Setting up LAN hardware** — This entails choosing a network topology, purchasing the equipment you need, and installing it (adding cards and connecting wires or using wireless antennas).
2. **Configuring TCP/IP** — To use most of the networking applications and tools that come with Linux, you must have TCP/IP configured. TCP/IP lets you communicate not only with computers on your LAN, but to any computers that you can reach on your LAN, modem, or other network connection (particularly to the Internet).

Setting up LAN hardware

To set up a simple LAN, the following decisions need to be done

- Decisions about network topology (that is, how computers are connected) should be done
- Decisions about network equipment (network interface cards, wires, hubs, and so on).

LAN topologies:

Most small office and home LANs connect computers together in one of the following topologies:

Star topology — the star topology is by far the most popular LAN topology. In this arrangement, each computer contains a Network Interface Card (NIC) that connects with a cable to a central hub.

Bus topology — Instead of using hubs, the bus topology connects computers in a chain from one to the next. The cabling usually used is referred to as coaxial, or Thin Ethernet cable. A "T" connector attaches to each computer's NIC, then to two adjacent computers in the chain. At the two ends of the chain, the T connectors are terminated.

Ring topology — this is a less popular topology than star and bus topologies. In a ring topology, computers connect to a ring of wires on which tokens are taken and passed by computers that want to send information on the network.

You can configure a wireless Ethernet LAN in several different topologies, depending on how you want to use the LAN. With a wireless LAN, each computer broadcasts in the air rather than across wires. Here are some examples of wireless topologies:

Wireless peer-to-peer — in this topology, frames of data are broadcast to all nodes within range, but are consumed only by the computers for which they are intended. This arrangement is useful if you are sharing file and print services among a group of client computers.

Wireless access point — a wireless interface can act as an access point for one or more wireless clients. Clients can be configured to communicate directly with the access point, instead of with every client that is within range. This arrangement is useful for point-to-point connections between two buildings, where the access point is acting as a gateway to the Internet or, for example, a campus intranet.

LAN equipment:

The equipment that you need to connect your LAN can include some or all of the following:

Network Interface Card (NIC) — typically, one of these cards goes into a slot in each computer. For wired Ethernet networks, the cards can transmit data at 10 Mbps or 100 Mbps. Gigabit (1000 Mbps) NICs are also now available, but are quite a bit more expensive. An 802.11B wireless NIC card can operate at speeds of up to 11 Mbps, but are more expensive than a wired NIC card.

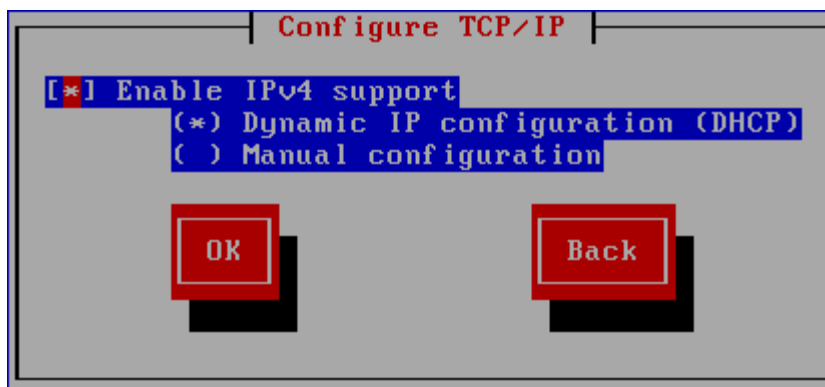
Cables — for star topologies, cables are referred to as twisted-pair. Category 5e wiring, which contains four twisted-pair sets per wire, is the most common type of wiring used for LANs today. A connector at each end of the cable is an RJ-45 plug, similar to those used on telephone cables. Ethernet interfaces are either 10Base-T (10 Mbps speeds) or 100Base-TX (100 Mbps speeds). These cables plug into the computer's NIC at one end and the hub at the other.

Hubs — with the star topology, a hub is typically used to connect the computers. Sometimes hubs are also referred to as repeaters or concentrators because they receive signals from the nodes connected to them and send the signals on to other nodes.

Switches — a switch can be used instead of a hub. It lets you divide a LAN that is getting too large into segments that are more manageable. A switch can reduce network traffic by directing messages intended for a specific computer directly to that computer. This is as opposed to a hub, which broadcasts all data to all nodes. Because switches have come down so much in price, in most cases you should just pay a few extra dollars and get a switch instead of a hub.

Configuring TCP/IP for your LAN(during Installation of OS):

During OS Installation, in the part of network installation, the **Configure TCP/IP** dialog appears.



This dialog asks for your IP and other network addresses.

1. You can choose to configure the IP address and Netmask of the device via DHCP or manually.
2. By default, the installation program uses DHCP to automatically provide network settings.
3. If you use a cable or DSL modem, router, firewall, or other network hardware to communicate with the Internet, DHCP is a suitable option.
4. If your network has no DHCP server, clear the check box labelled **Use dynamic IP configuration (DHCP)**. Enter the IP address you are using during installation.

Network Configuration

Fedora, the Linux Operating System, contains support for both IPv4 and IPv6. However, by default, the installation program configures network interfaces on your computer for IPv4.

1. During Network Configuration, a Setup prompts you to supply a host name and domain name for the computer, in the format **hostname. domainname**.
2. Many networks have a DHCP (Dynamic Host Configuration Protocol) service that automatically supplies connected systems with a domain name, leaving the user to enter a hostname.
3. Unless you have a specific need to customize the host name and domain name, the default setting **localhost. localdomain** is a good choice for most users.
4. To set up a network that is behind an Internet firewall or router, you may want to use hostname. localdomain for your Fedora system.
5. If you have more than one computer on this network, you should give each one a separate host name in this domain.
6. In some networks, the DHCP provider also provides the name of the computer, or hostname. The complete hostname includes both the name of the machine and the name of the domain of which it is a member, such as machine1.example.com. The machine name (or "short hostname") is machine1, and the domain name is example.com.





Please name this computer. The hostname identifies the computer on a network.

Hostname:

← BackNext →

Configuring the network interface:

Enable network interface

This requires that you have an active network connection during the installation process. Please configure a network interface.

Interface: eth0 - Advanced Micro Devices [AMD] 79c970 [PCnet32 LANCE] ▼

☒ Use dynamic IP configuration (DHCP)

☒ Enable IPv4 support

IPv4 Address: /

Gateway:

Nameserver:

If your network does not have DHCP enabled, or if you need to override the DHCP settings, select the network interface that you plan to use from the **Interfaces** menu. Clear the checkbox for **Use dynamic IP configuration (DHCP)**. You can now enter an IPv4 address and netmask for this system in the form **address / netmask**, along with the gateway address and nameserver address for your network.

LAN setup:

With an Ethernet NIC, appropriate cables, and a switch, you are ready to set up your wired Ethernet LAN.

Steps for setting up an Ethernet LAN are:

- Power down each computer and physically install the NIC card (following the manufacturer's instructions).
- Using cables appropriate for your NIC cards and switch, connect each NIC to the switch.
- Power up each computer.
- If Linux is not installed yet, install the software and reboot (as instructed).
- If Linux is already installed, when the system comes up, your Ethernet card and interface (eth0) should be ready to use.

RESULT:

Thus the setting up the local network using switch is executed successfully

