EXPERIMENT 10

AIM:

To understand Wireshark tool and explore its features like filters, flow graphs, statistics and protocol hierarchy

THEORY:

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development.It can parse and display the fields, along with their meanings as specified by different networking protocols. Wireshark uses pcap to capture packets, so it can only capture packets on the types of networks that pcap supports. Wireshark can color packets based on rules that match particular fields in packets, to help the user identify the types of traffic at a glance.

Filters : Wireshark share a powerful filter engine that helps remove the noise from a packet trace and lets you see only the packets that interest you. If a packet meets the requirements expressed in your filter, then it is displayed in the list of packets. Display filters let you compare the fields within a protocol against a specific value, compare fields against fields, and check the existence of specified fields or protocols.

Flow Graph : The Flow Graph window shows connections between hosts. It displays the packet time, direction, ports and comments for each captured connection. You can filter all connections by ICMP Flows, ICMPv6 Flows, UIM Flows and TCP Flows. Flow Graph window is used for showing multiple different topics. Each vertical line represents the specific host, which you can see in the top of the window. The numbers in each row at the very left of the window represent the time packet. The numbers at the both ends of each arrow between hosts represent the port numbers.

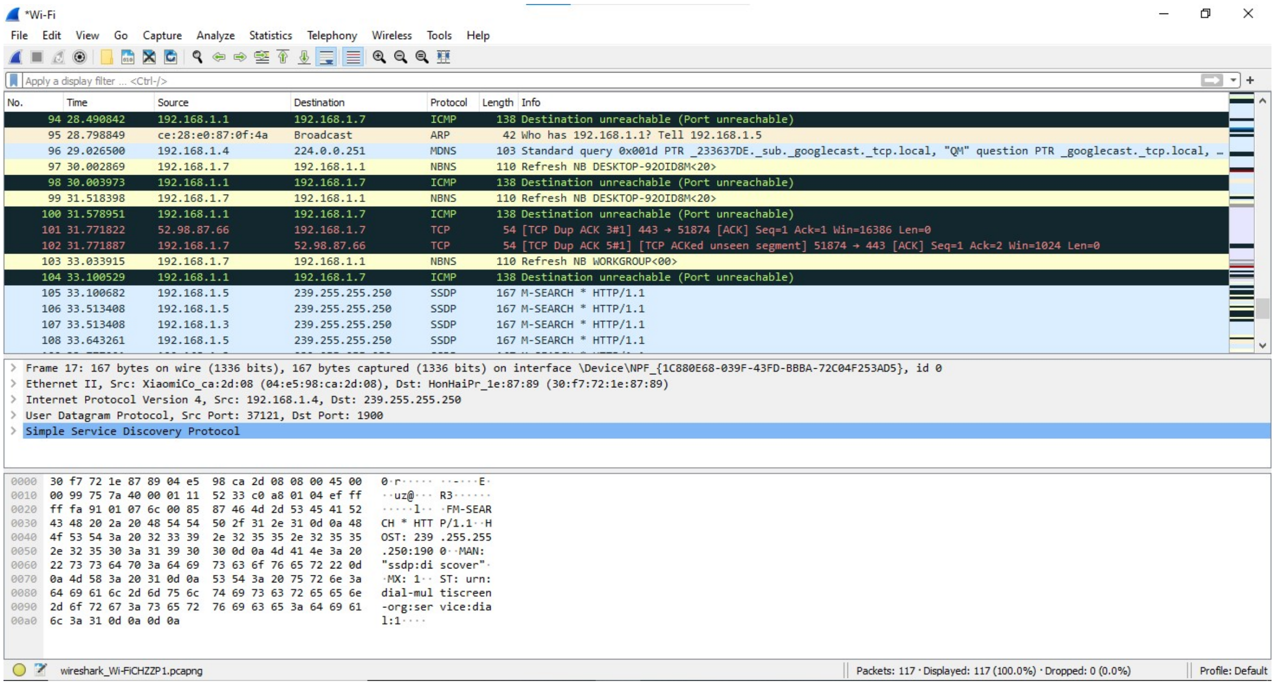
Statistics : Wireshark provides a wide range of network statistics. These statistics range from general information about the loaded capture file (like the number of captured packets), to statistics about specific protocols (e.g. statistics about the number of HTTP requests and responses captured). General statistics involve Summary about the capture file like: packet counts, captured time period, Protocol Hierarchy of the captured packets, Conversations like traffic between specific Ethernet/IP/… addresses etc.

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

RESULT:

The experiment was executed successfully.

OUTPUT:


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

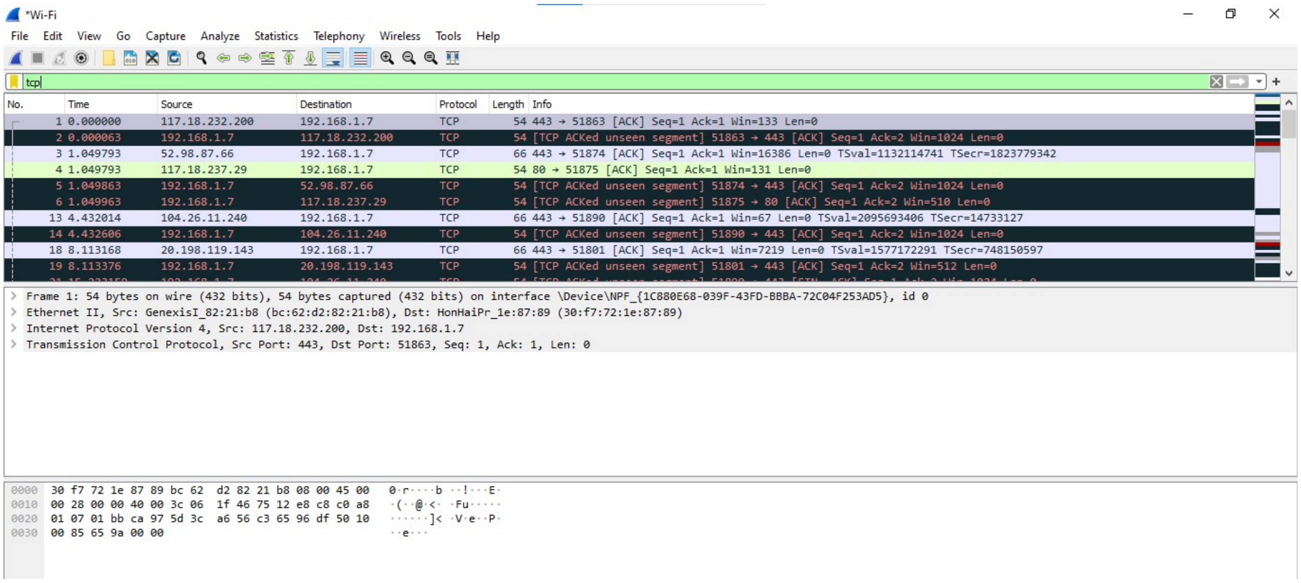
```
No.   Time       Source             Destination       Protocol  Length Info
   94 28.490842  192.168.1.1        192.168.1.7       ICMP      138 Destination unreachable (Port unreachable)
   95 28.798849  ce:28:e0:87:0f:4a  Broadcast         ARP        42 Who has 192.168.1.1? Tell 192.168.1.5
   96 29.026500  192.168.1.4        224.0.0.251       MDNS      103 Standard query 0x001d PTR _233637DE._sub._googlecast._tcp.local, "QM" question PTR _googlecast._tcp.local, …
   97 30.002869  192.168.1.7        192.168.1.1       NBNS      110 Refresh NB DESKTOP-92OID8M<20>
   98 30.003973  192.168.1.1        192.168.1.7       ICMP      138 Destination unreachable (Port unreachable)
   99 31.518398  192.168.1.7        192.168.1.1       NBNS      110 Refresh NB DESKTOP-92OID8M<20>
  100 31.578951  192.168.1.1        192.168.1.7       ICMP      138 Destination unreachable (Port unreachable)
  101 31.771822  52.98.87.66        192.168.1.7       TCP        54 [TCP Dup ACK 3#1] 443 → 51874 [ACK] Seq=1 Ack=1 Win=16386 Len=0
  102 31.771887  192.168.1.7        52.98.87.66       TCP        54 [TCP Dup ACK 5#1] [TCP ACKed unseen segment] 51874 → 443 [ACK] Seq=1 Ack=2 Win=1024 Len=0
  103 33.033915  192.168.1.7        192.168.1.1       NBNS      110 Refresh NB WORKGROUP<00>
  104 33.100529  192.168.1.1        192.168.1.7       ICMP      138 Destination unreachable (Port unreachable)
  105 33.100682  192.168.1.5        239.255.255.250   SSDP      167 M-SEARCH * HTTP/1.1
  106 33.513408  192.168.1.5        239.255.255.250   SSDP      167 M-SEARCH * HTTP/1.1
  107 33.513408  192.168.1.3        239.255.255.250   SSDP      167 M-SEARCH * HTTP/1.1
  108 33.643261  192.168.1.5        239.255.255.250   SSDP      167 M-SEARCH * HTTP/1.1
```

```
> Frame 17: 167 bytes on wire (1336 bits), 167 bytes captured (1336 bits) on interface \Device\NPF_{1C880E68-039F-43FD-BBBA-72C04F253AD5}, id 0
> Ethernet II, Src: XiaomiCo_ca:2d:08 (04:e5:98:ca:2d:08), Dst: HonHaiPr_1e:87:89 (30:f7:72:1e:87:89)
> Internet Protocol Version 4, Src: 192.168.1.4, Dst: 239.255.255.250
> User Datagram Protocol, Src Port: 37121, Dst Port: 1900
> Simple Service Discovery Protocol
```

```
0000  30 f7 72 1e 87 89 04 e5  98 ca 2d 08 08 00 45 00   0·r····· ··-···E·
0010  00 99 75 7a 40 00 01 11  52 33 c0 a8 01 04 ef ff   ··uz@··· R3······
0020  ff fa 91 01 07 6c 00 85  87 46 4d 2d 53 45 41 52   ·····l·· ·FM-SEAR
0030  43 48 20 2a 20 48 54 54  50 2f 31 2e 31 0d 0a 48   CH * HTT P/1.1··H
0040  4f 53 54 3a 20 32 33 39  2e 32 35 35 2e 32 35 35   OST: 239 .255.255
0050  2e 32 35 30 3a 31 39 30  30 0d 0a 4d 41 4e 3a 20   .250:190 0··MAN: 
0060  22 73 73 64 70 3a 64 69  73 63 6f 76 65 72 22 0d   "ssdp:di scover"·
0070  0a 4d 58 3a 20 31 0d 0a  53 54 3a 20 75 72 6e 3a   ·MX: 1·· ST: urn:
0080  64 69 61 6c 2d 6d 75 6c  74 69 73 63 72 65 65 6e   dial-mul tiscreen
0090  2d 6f 72 67 3a 73 65 72  76 69 63 65 3a 64 69 61   -org:ser vice:dia
00a0  6c 3a 31 0d 0a 0d 0a                               l:1····
```

wireshark_Wi-FiCHZZP1.pcapng | Packets: 117 · Displayed: 117 (100.0%) · Dropped: 0 (0.0%) | Profile: Default

Tcp filter


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

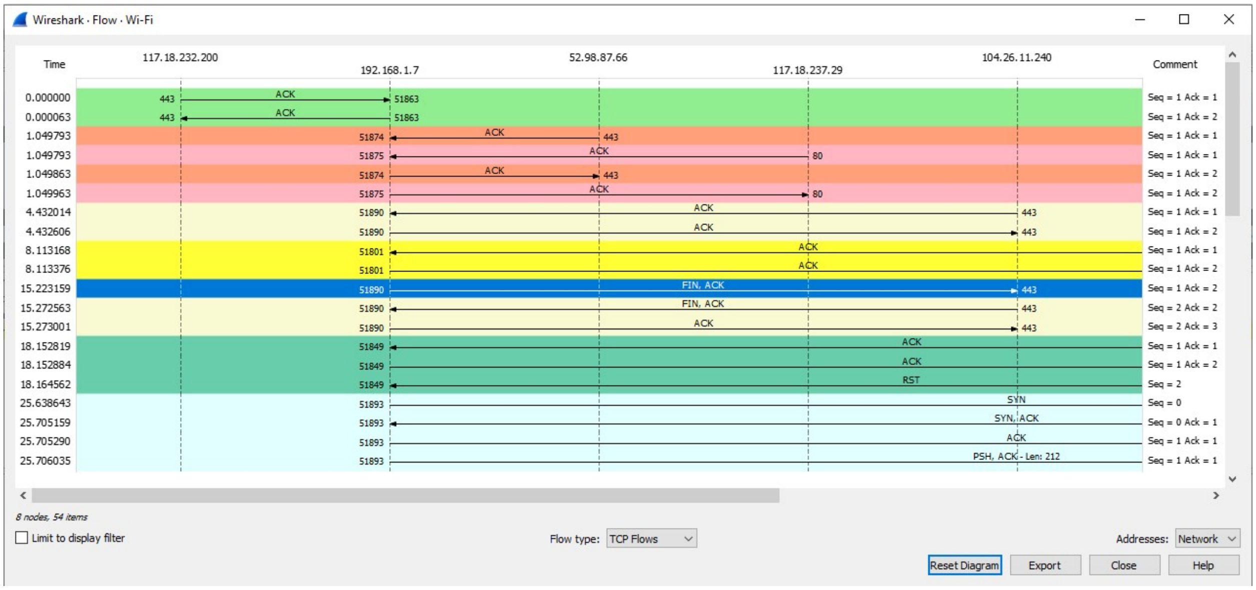
tcp

```
No.   Time       Source             Destination       Protocol  Length Info
    1 0.000000   117.18.232.200     192.168.1.7       TCP        54 443 → 51863 [ACK] Seq=1 Ack=1 Win=133 Len=0
    2 0.000063   192.168.1.7        117.18.232.200    TCP        54 [TCP ACKed unseen segment] 51863 → 443 [ACK] Seq=1 Ack=2 Win=1024 Len=0
    3 1.049793   52.98.87.66        192.168.1.7       TCP        66 443 → 51874 [ACK] Seq=1 Ack=1 Win=16386 Len=0 TSval=1132114741 TSecr=1823779342
    4 1.049793   117.18.237.29      192.168.1.7       TCP        54 80 → 51875 [ACK] Seq=1 Ack=1 Win=131 Len=0
    5 1.049863   192.168.1.7        52.98.87.66       TCP        54 [TCP ACKed unseen segment] 51874 → 443 [ACK] Seq=1 Ack=2 Win=1024 Len=0
    6 1.049963   192.168.1.7        117.18.237.29     TCP        54 [TCP ACKed unseen segment] 51875 → 80 [ACK] Seq=1 Ack=2 Win=510 Len=0
   13 4.432014   104.26.11.240      192.168.1.7       TCP        66 443 → 51890 [ACK] Seq=1 Ack=1 Win=67 Len=0 TSval=2095693406 TSecr=14733127
   14 4.432606   192.168.1.7        104.26.11.240     TCP        54 [TCP ACKed unseen segment] 51890 → 443 [ACK] Seq=1 Ack=2 Win=1024 Len=0
   18 8.113168   20.198.119.143     192.168.1.7       TCP        66 443 → 51801 [ACK] Seq=1 Ack=1 Win=7219 Len=0 TSval=1577172291 TSecr=748150597
   19 8.113376   192.168.1.7        20.198.119.143    TCP        54 [TCP ACKed unseen segment] 51801 → 443 [ACK] Seq=1 Ack=2 Win=512 Len=0
```

```
> Frame 1: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{1C880E68-039F-43FD-BBBA-72C04F253AD5}, id 0
> Ethernet II, Src: GenexisI_82:21:b8 (bc:62:d2:82:21:b8), Dst: HonHaiPr_1e:87:89 (30:f7:72:1e:87:89)
> Internet Protocol Version 4, Src: 117.18.232.200, Dst: 192.168.1.7
> Transmission Control Protocol, Src Port: 443, Dst Port: 51863, Seq: 1, Ack: 1, Len: 0
```

```
0000  30 f7 72 1e 87 89 bc 62  d2 82 21 b8 08 00 45 00   0·r····b ··!···E·
0010  00 28 00 00 40 00 3c 06  1f 46 75 12 e8 c8 c0 a8   ·(··@·<· ·Fu·····
0020  01 07 01 bb ca 97 5d 3c  a6 56 c3 65 96 df 50 10   ······]< ·V·e··P·
0030  00 85 65 9a 00 00                                  ··e···
```

Flow graphs



Wireshark · Flow · Wi-Fi

Time	117.18.232.200	192.168.1.7	52.98.87.66	117.18.237.29	104.26.11.240		Comment
0.000000	443	51863				ACK	Seq = 1 Ack = 1
0.000063	443	51863				ACK	Seq = 1 Ack = 2
1.049793		51874	443			ACK	Seq = 1 Ack = 1
1.049793		51875		80		ACK	Seq = 1 Ack = 1
1.049863		51874	443			ACK	Seq = 1 Ack = 2
1.049963		51875		80		ACK	Seq = 1 Ack = 2
4.432014		51890			443	ACK	Seq = 1 Ack = 1
4.432606		51890			443	ACK	Seq = 1 Ack = 2
8.113168		51801				ACK	Seq = 1 Ack = 1
8.113376		51801				ACK	Seq = 1 Ack = 2
15.223159		51890			443	FIN, ACK	Seq = 1 Ack = 2
15.272563		51890			443	FIN, ACK	Seq = 2 Ack = 2
15.273001		51890			443	ACK	Seq = 2 Ack = 3
18.152819		51849				ACK	Seq = 1 Ack = 1
18.152884		51849				ACK	Seq = 1 Ack = 2
18.164562		51849				RST	Seq = 2
25.638643		51893				SYN	Seq = 0
25.705159		51893				SYN, ACK	Seq = 0 Ack = 1
25.705290		51893				ACK	Seq = 1 Ack = 1
25.706035		51893				PSH, ACK - Len: 212	Seq = 1 Ack = 1

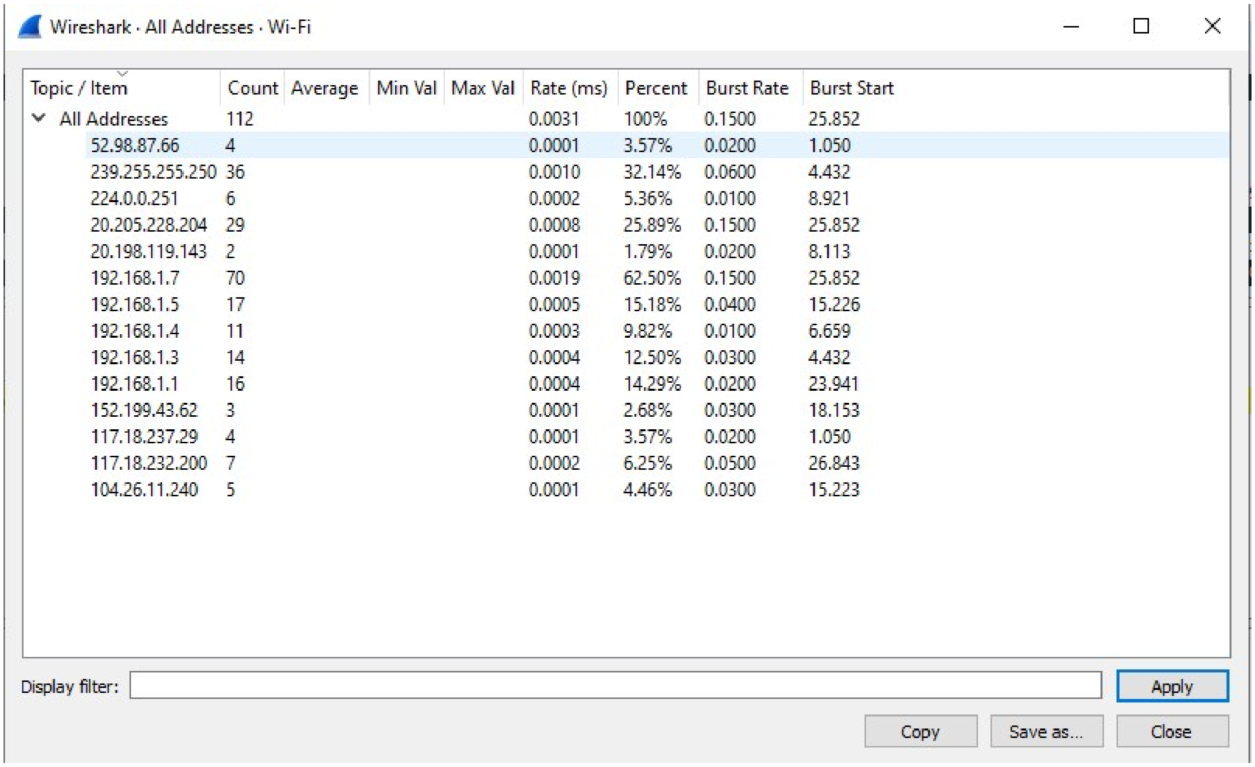
8 nodes, 54 items

☐ Limit to display filter

Flow type: TCP Flows

Addresses: Network

Statistics

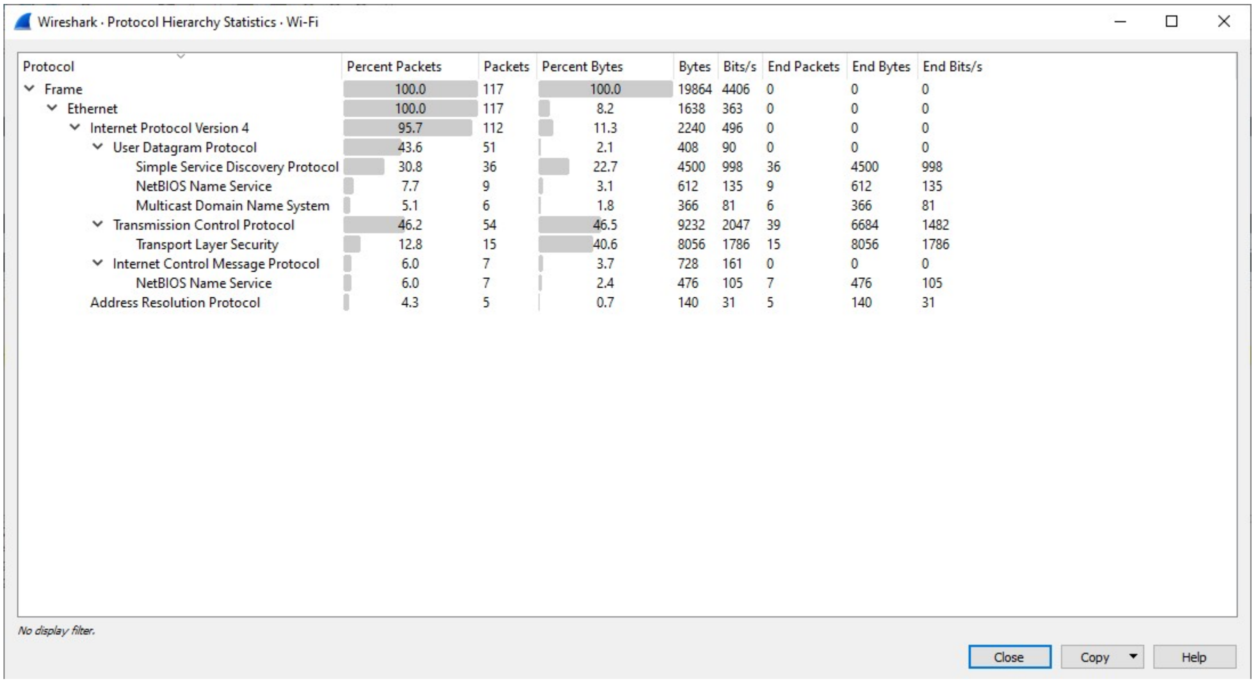


Wireshark · All Addresses · Wi-Fi

Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
All Addresses	112				0.0031	100%	0.1500	25.852
52.98.87.66	4				0.0001	3.57%	0.0200	1.050
239.255.255.250	36				0.0010	32.14%	0.0600	4.432
224.0.0.251	6				0.0002	5.36%	0.0100	8.921
20.205.228.204	29				0.0008	25.89%	0.1500	25.852
20.198.119.143	2				0.0001	1.79%	0.0200	8.113
192.168.1.7	70				0.0019	62.50%	0.1500	25.852
192.168.1.5	17				0.0005	15.18%	0.0400	15.226
192.168.1.4	11				0.0003	9.82%	0.0100	6.659
192.168.1.3	14				0.0004	12.50%	0.0300	4.432
192.168.1.1	16				0.0004	14.29%	0.0200	23.941
152.199.43.62	3				0.0001	2.68%	0.0300	18.153
117.18.237.29	4				0.0001	3.57%	0.0200	1.050
117.18.232.200	7				0.0002	6.25%	0.0500	26.843
104.26.11.240	5				0.0001	4.46%	0.0300	15.223

Display filter:

Protocol Hierarchy



Wireshark · Protocol Hierarchy Statistics · Wi-Fi

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	117	100.0	19864	4406	0	0	0
Ethernet	100.0	117	8.2	1638	363	0	0	0
Internet Protocol Version 4	95.7	112	11.3	2240	496	0	0	0
User Datagram Protocol	43.6	51	2.1	408	90	0	0	0
Simple Service Discovery Protocol	30.8	36	22.7	4500	998	36	4500	998
NetBIOS Name Service	7.7	9	3.1	612	135	9	612	135
Multicast Domain Name System	5.1	6	1.8	366	81	6	366	81
Transmission Control Protocol	46.2	54	46.5	9232	2047	39	6684	1482
Transport Layer Security	12.8	15	40.6	8056	1786	15	8056	1786
Internet Control Message Protocol	6.0	7	3.7	728	161	0	0	0
NetBIOS Name Service	6.0	7	2.4	476	105	7	476	105
Address Resolution Protocol	4.3	5	0.7	140	31	5	140	31

No display filter.

EXPERIMENT 11

AIM:

Study of Cisco Packet Tracer and configure FTP server, DHCP server and DNS server in a wired network using required network devices

THEORY:

Packet Tracer is a cross-platform visual simulation tool designed by Cisco Systems that allows users to create network topologies and imitate modern computer networks. The software allows users to simulate the configuration of Cisco routers and switches using a simulated command line interface. Packet Tracer supports an array of simulated Application Layer protocols, as well as basic routing with RIP, OSPF, EIGRP and BGP.

DNS : The Domain Name System (DNS) is the hierarchical and decentralized naming system used to identify computers reachable through the Internet or other Internet Protocol (IP) networks. The resource records contained in the DNS associate domain names with other forms of information. These are most commonly used to map human-friendly domain names to the numerical IP addresses computers need to locate services and devices using the underlying network protocols.

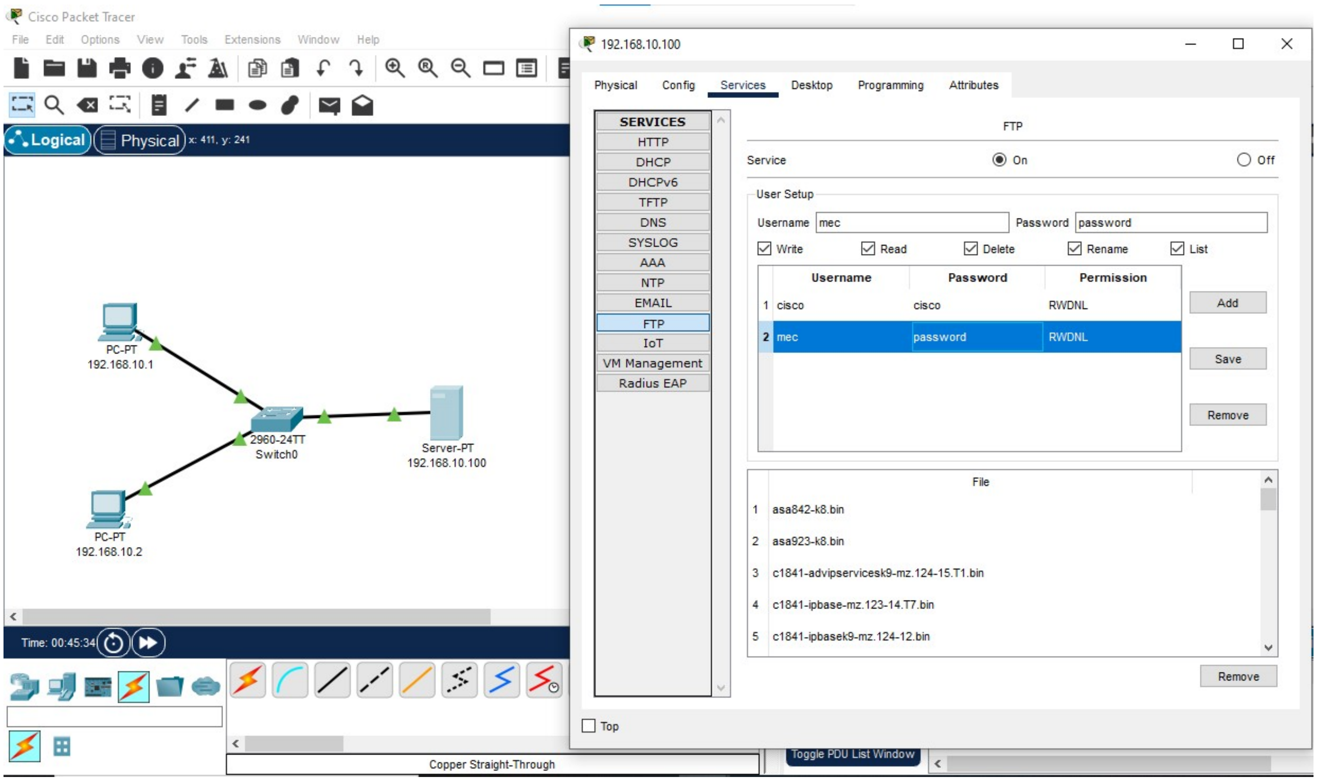
DHCP: The Dynamic Host Configuration Protocol (DHCP) is a network management protocol used on Internet Protocol networks for automatically assigning IP addresses and other communication parameters to devices connected to the network using a client–server architecture. It employs a connectionless service model, using the User Datagram Protocol (UDP). It is implemented with two UDP port numbers, 67 is the destination port of a server, and 68 is used by the client.

RESULT:

The experiment was executed successfully

OUTPUT:

FTP



192.168.10.100

Physical | Config | Services | Desktop | Programming | Attributes

SERVICES: HTTP, DHCP, DHCPv6, TFTP, DNS, SYSLOG, AAA, NTP, EMAIL, FTP, IoT, VM Management, Radius EAP

FTP

Service ● On ○ Off

User Setup

Username mec Password password

☑ Write ☑ Read ☑ Delete ☑ Rename ☑ List

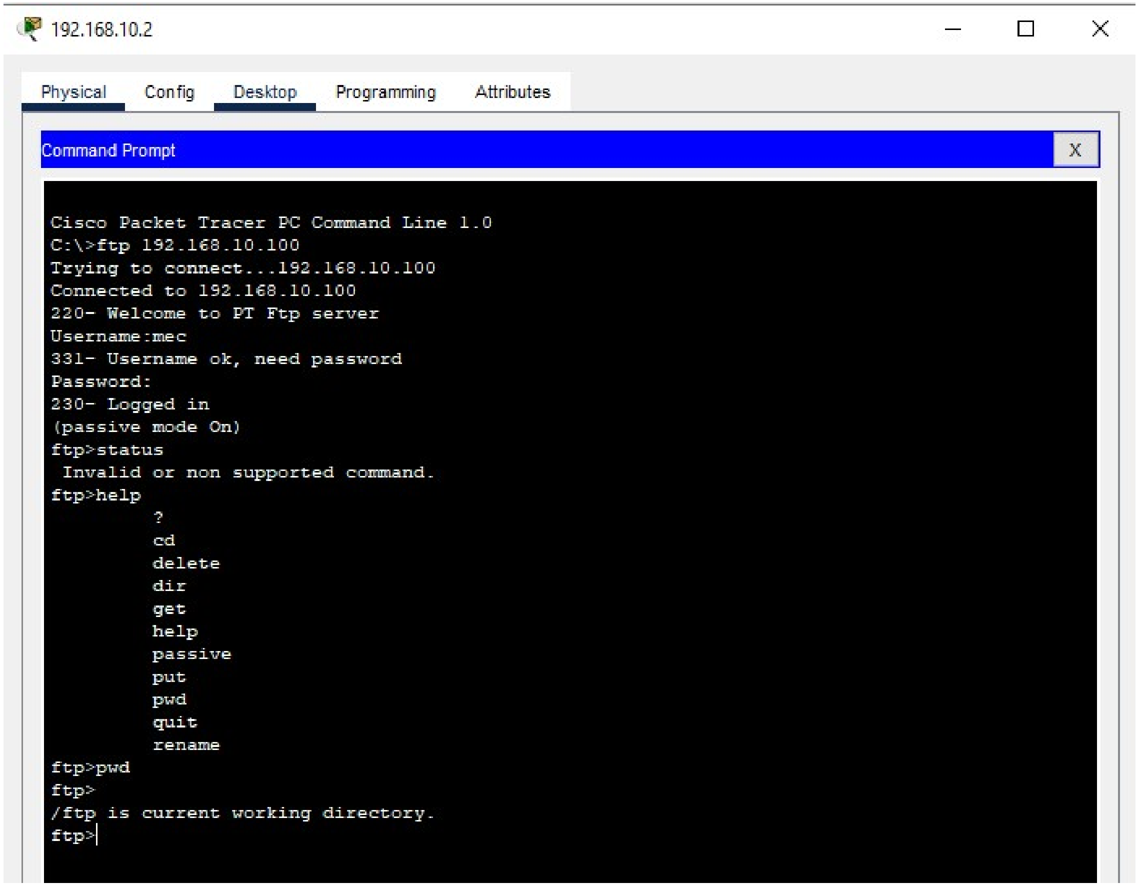
	Username	Password	Permission
1	cisco	cisco	RWDNL
2	mec	password	RWDNL

Add Save Remove

	File
1	asa842-k8.bin
2	asa923-k8.bin
3	c1841-advipservicesk9-mz.124-15.T1.bin
4	c1841-ipbase-mz.123-14.T7.bin
5	c1841-ipbasek9-mz.124-12.bin

Remove

□ Top



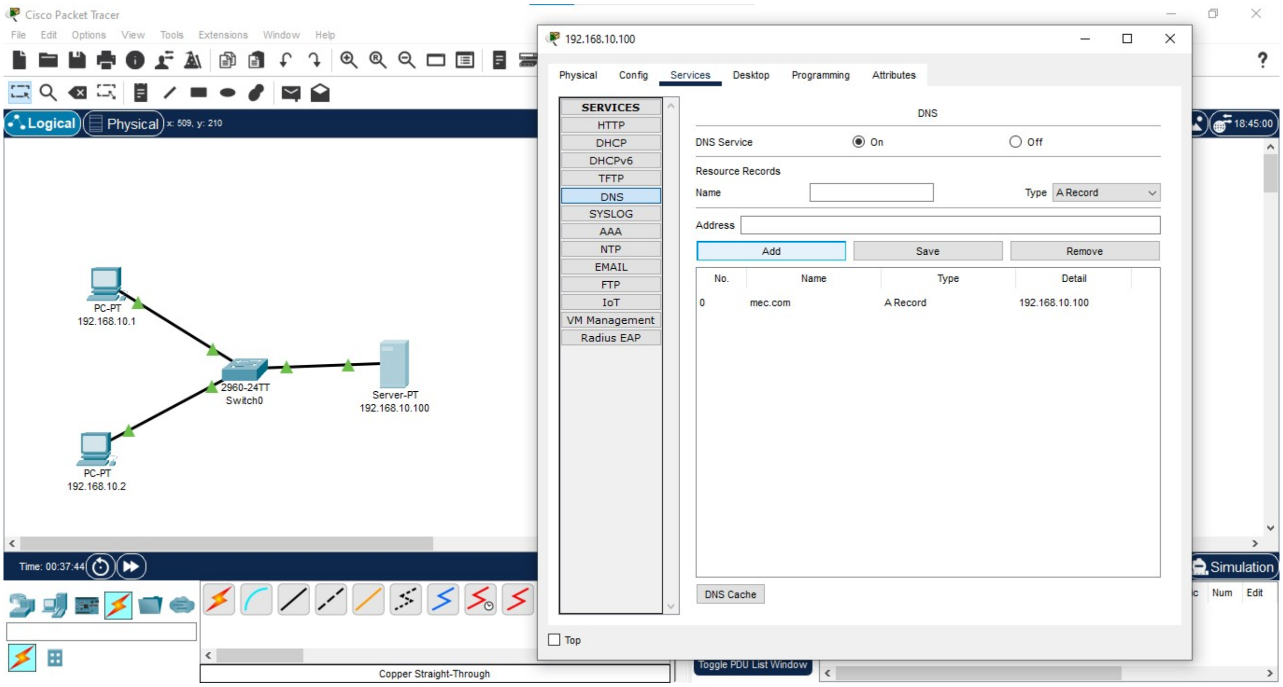
192.168.10.2

Physical | Config | Desktop | Programming | Attributes

Command Prompt X

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ftp 192.168.10.100
Trying to connect...192.168.10.100
Connected to 192.168.10.100
220- Welcome to PT Ftp server
Username:mec
331- Username ok, need password
Password:
230- Logged in
(passive mode On)
ftp>status
 Invalid or non supported command.
ftp>help
         ?
         cd
         delete
         dir
         get
         help
         passive
         put
         pwd
         quit
         rename
ftp>pwd
ftp>
/ftp is current working directory.
ftp>
```

DNS



192.168.10.100

DNS

DNS Service ● On ○ Off

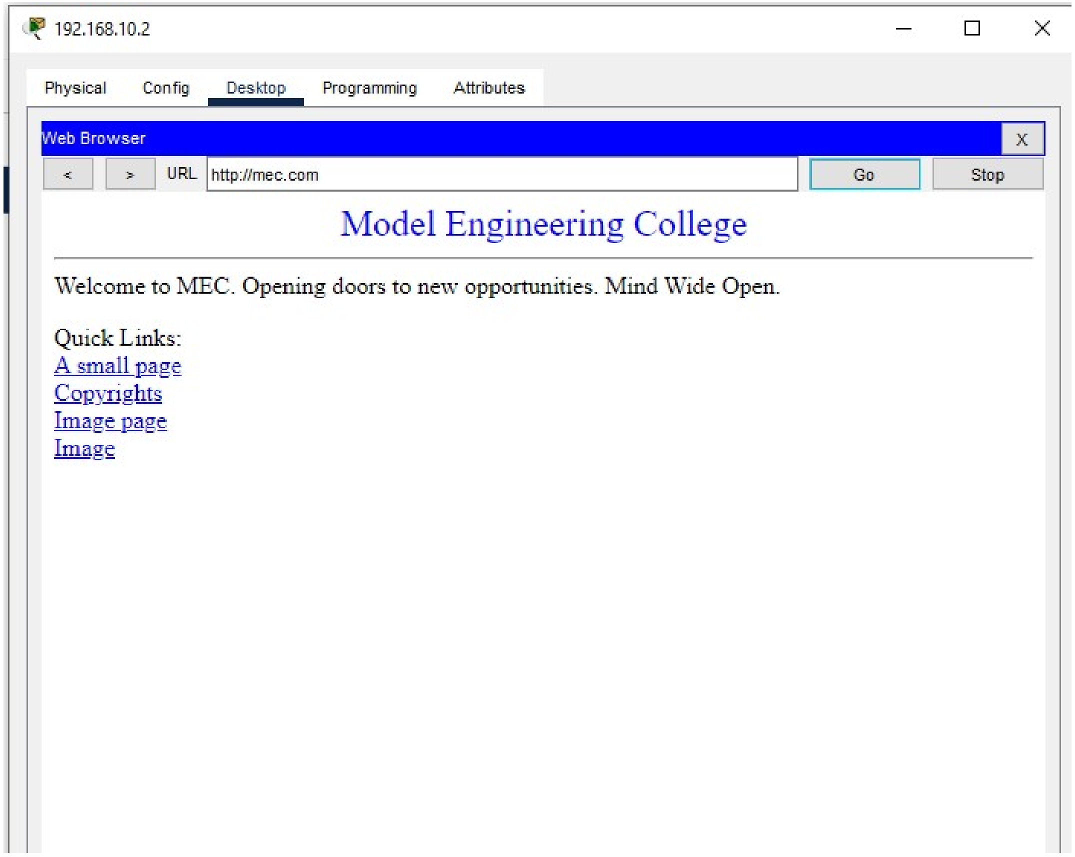
Resource Records

Name

Type A Record

Address

No.	Name	Type	Detail
0	mec.com	A Record	192.168.10.100



192.168.10.2

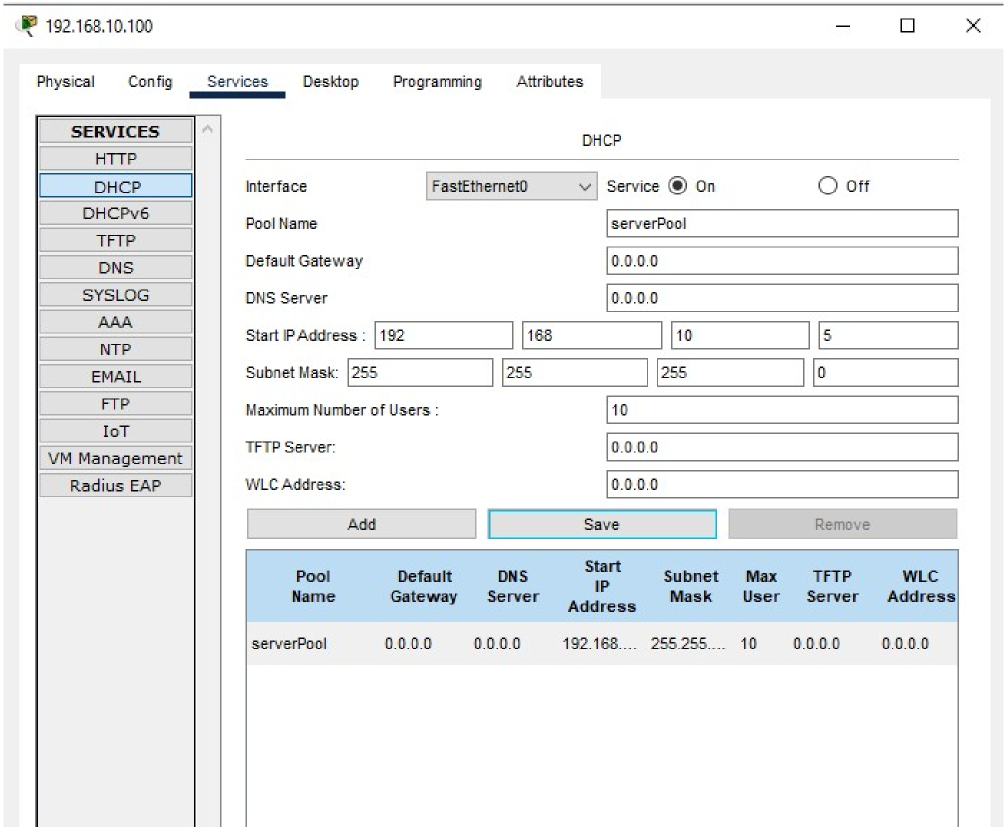
URL http://mec.com

Model Engineering College

Welcome to MEC. Opening doors to new opportunities. Mind Wide Open.

Quick Links:
A small page
Copyrights
Image page
Image

DHCP

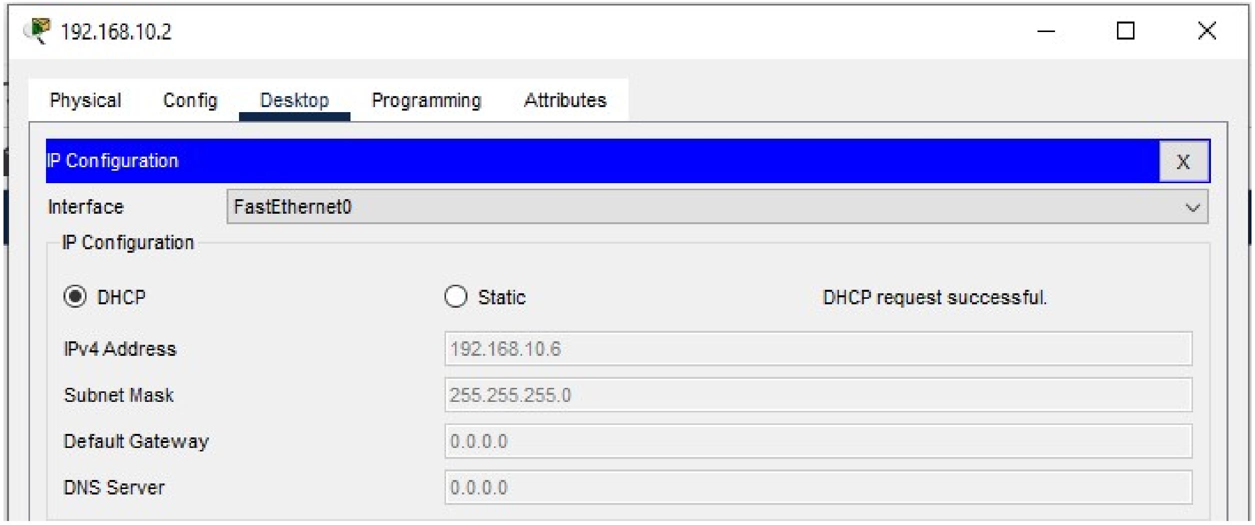


DHCP

Interface: FastEthernet0   Service ● On ○ Off

Pool Name	serverPool
Default Gateway	0.0.0.0
DNS Server	0.0.0.0
Start IP Address :	192 . 168 . 10 . 5
Subnet Mask:	255 . 255 . 255 . 0
Maximum Number of Users :	10
TFTP Server:	0.0.0.0
WLC Address:	0.0.0.0

Pool Name	Default Gateway	DNS Server	Start IP Address	Subnet Mask	Max User	TFTP Server	WLC Address
serverPool	0.0.0.0	0.0.0.0	192.168....	255.255....	10	0.0.0.0	0.0.0.0



Interface: FastEthernet0

IP Configuration

● DHCP ○ Static DHCP request successful.

IPv4 Address	192.168.10.6
Subnet Mask	255.255.255.0
Default Gateway	0.0.0.0
DNS Server	0.0.0.0

EXPERIMENT 12

AIM:

Study of NS2 and simulate Link State Protocol and Distance Vector Routing protocol in it

THEORY:

NS (Network Simulator) is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. It is an object-oriented, discrete event-driven simulator written in C++ and Otcl/Tcl. NS-2 can be used to implement network protocols such as TCP and UDP, traffic source behavior such as FTP, Telnet, Web, CBR, and VBR, router queues management mechanism such as Drop Tail, RED, and CBQ and routing algorithms.Install NS-2 using this command :

```
sudo apt-get install ns2
```

Nam (Network Animator) is an animation tool to graphically represent the network and packet traces. Use this command :

```
sudo apt-get install nam
```

Basic Commands :

```
set a 8
set b [expr $a/8]
```

In the first line, the variable **a** is assigned the value 8. In the second line, the result of the command [expr $a/8], which equals 1, is then used as an argument to another command, which in turn assigns a value to the variable **b**. The “$” sign is used to obtain a value contained in a variable and square brackets are an indication of command substitution.

Define new procedures with *proc* command :

```
proc factorial fact {

   if {$fact <= 1} {

         return 1
   }
expr $fact * [factorial [expr $fact-1]]
}
```

To open a file for reading :

```
set testfile [open hello.data r]
```

Similarly, put command is used to write data into the file.

```
set testfile [open hello.data w]

puts $testfile “hello1”
```

To call sub processes within another process, exec is used, which creates a subprocess and waits for it to complete.

```
exec rm $testfile
```

To be able to run a simulation scenario, a network topology must first be created. In ns2, the topology consists of a collection of nodes and links.

```
set ns [new Simulator]
```

The simulator object has member functions that enable the creation of the nodes and define the links between them. The class simulator contains all the basic functions. Since ns was defined to handle the Simulator object, the command $ns is used for using the functions belonging to the simulator class.

In the network topology nodes can be added in the following manner :

```
set n0 [$ns node]

set n1 [$ns node]
```

Traffic agents (TCP, UDP, etc.) and traffic sources (FTP, CBR, etc.) must be set up if the node is not a router. It enables the creation of CBR traffic source using UDP as transport protocol or an FTP traffic source using TCP as a transport protocol.

CBR traffic source using UDP :

```
set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packet_size_ 512
```

PROGRAM:

ls.tcl

```
set ns [new Simulator]
$ns rtproto LS
set nf [open ls1.tr w]
$ns trace-all $nf
set nr [open ls2.nam w]
$ns namtrace-all $nr

proc finish {} {
global ns nf nr
$ns flush-trace
close $nf
close $nr
exec nam ls2.nam
exit 0
}
for {set i 0} {$i<12} {incr i} {
set n$i [$ns node]
}

$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns duplex-link $n5 $n6 1Mb 10ms DropTail
$ns duplex-link $n6 $n7 1Mb 10ms DropTail
$ns duplex-link $n7 $n8 1Mb 10ms DropTail
$ns duplex-link $n8 $n0 1Mb 10ms DropTail
$ns duplex-link $n0 $n9 1Mb 10ms DropTail
$ns duplex-link $n1 $n10 1Mb 10ms DropTail
$ns duplex-link $n9 $n11 1Mb 10ms DropTail
$ns duplex-link $n10 $n11 1Mb 10ms DropTail
$ns duplex-link $n11 $n5 1Mb 10ms DropTail

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
```

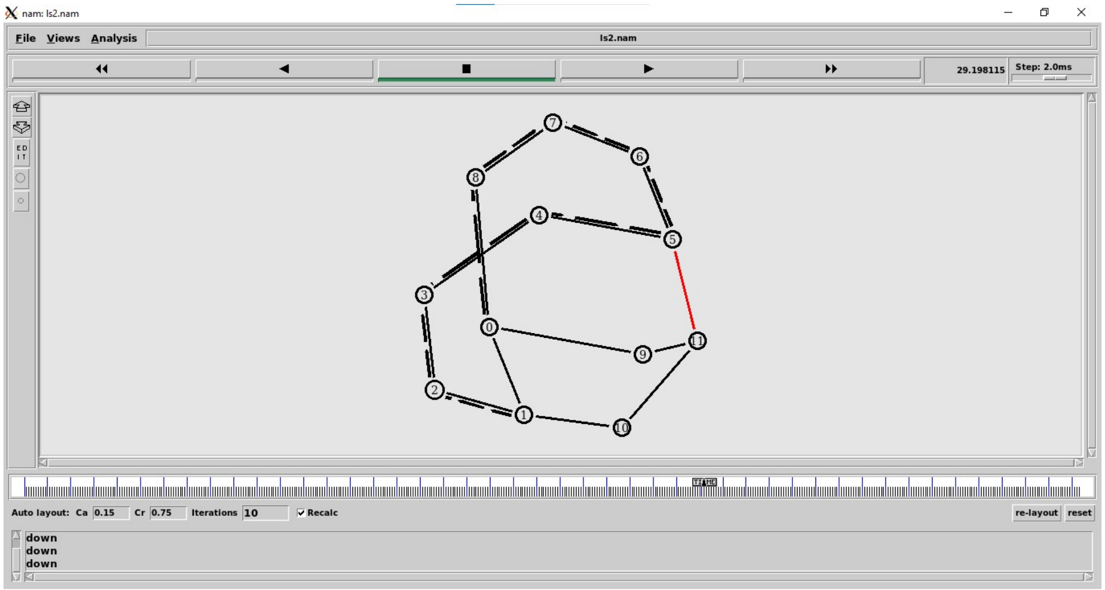
```
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
set null0 [new Agent/Null]
$ns attach-agent $n5 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005

set null1 [new Agent/Null]
$ns attach-agent $n5 $null1

$ns connect $udp0 $null0
$ns connect $udp1 $null1

$ns rtmodel-at 10.0 down $n11 $n5
$ns rtmodel-at 30.0 up $n11 $n5
$ns rtmodel-at 15.0 down $n7 $n6
$ns rtmodel-at 20.0 up $n7 $n6

$ns at .1 "$cbr1 start"
$ns at .2 "$cbr0 start"
$ns at 45.0 "$cbr1 stop"
$ns at 45.1 "$cbr0 stop"

$ns at 50.0 "finish"
$ns run
```

AWK Program for LS

```
BEGIN {
print " performance evaluation"
send=0
recv=0
dropped=0
rout=0
}
{
if($1=="+" && $3=="0"||"1" && $5=="cbr")
  {
```

```
 send++
   }
if($1=="r" && $4=="5" && $5=="cbr")
  {
 recv++
   }
if($1=="d")
  {
 dropped++
   }
if($1=="r" && $5=="rtProtoLS")
  {
 rout++
   }
}
END {
print "No of packets Send :" send
print "No of packets Received :" recv
print "No of packets dropped :" dropped
print "No of routing packets :" rout
NOH=rout/recv
PDR=recv/send
print "Normalised overhead :" NOH
print "Packet delivery ratio :" PDR
  }
```

OUTPUT:



Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ☑ Recalc

```
down
down
down
```

dv.tcl

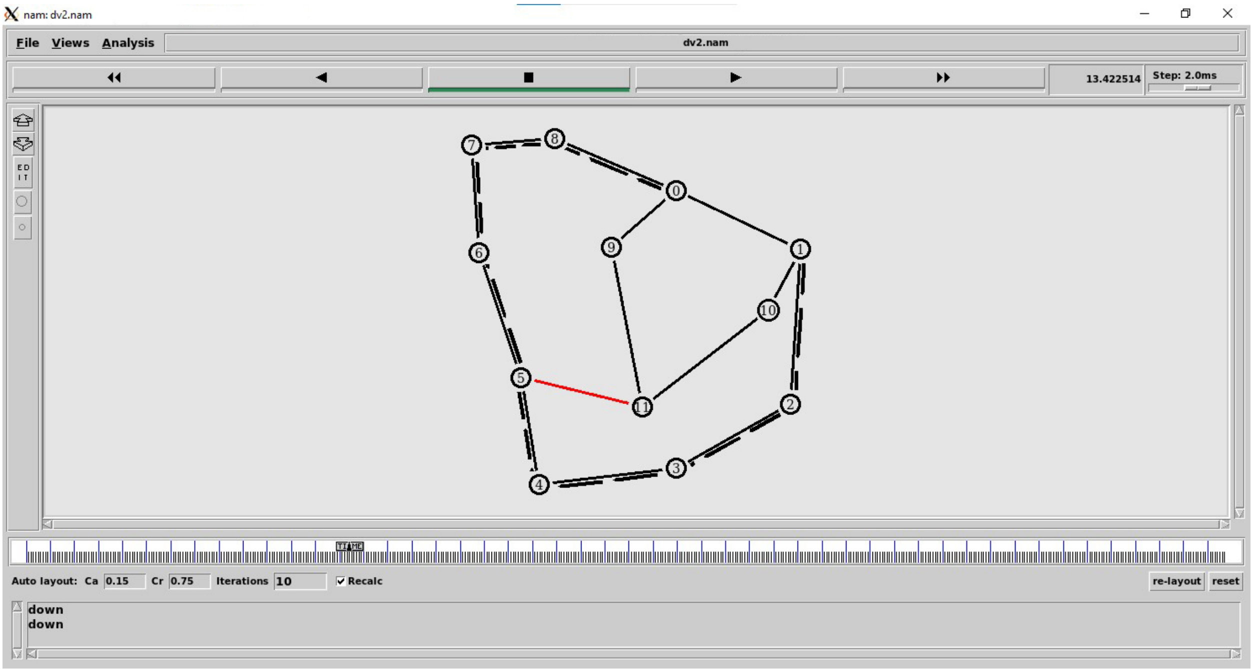
```
set ns [new Simulator]
$ns rtproto DV
set nf [open dv1.tr w]
$ns trace-all $nf
set nr [open dv2.nam w]
$ns namtrace-all $nr

proc finish {} {
global ns nf nr
$ns flush-trace
close $nf
close $nr
#exec nam dv2.nam
exit 0
}
for {set i 0} {$i<12} {incr i} {
set n$i [$ns node]
}
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns duplex-link $n5 $n6 1Mb 10ms DropTail
$ns duplex-link $n6 $n7 1Mb 10ms DropTail
$ns duplex-link $n7 $n8 1Mb 10ms DropTail
$ns duplex-link $n8 $n0 1Mb 10ms DropTail
$ns duplex-link $n0 $n9 1Mb 10ms DropTail
$ns duplex-link $n1 $n10 1Mb 10ms DropTail
$ns duplex-link $n9 $n11 1Mb 10ms DropTail
$ns duplex-link $n10 $n11 1Mb 10ms DropTail
$ns duplex-link $n11 $n5 1Mb 10ms DropTail

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
```

```
set null0 [new Agent/Null]
$ns attach-agent $n5 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005

set null1 [new Agent/Null]
$ns attach-agent $n5 $null1

$ns connect $udp0 $null0
$ns connect $udp1 $null1

$ns rtmodel-at 10.0 down $n11 $n5
$ns rtmodel-at 30.0 up $n11 $n5
$ns rtmodel-at 15.0 down $n7 $n6
$ns rtmodel-at 20.0 up $n7 $n6

$ns at .1 "$cbr1 start"
$ns at .2 "$cbr0 start"
$ns at 45.0 "$cbr1 stop"
$ns at 45.1 "$cbr0 stop"

$ns at 50.0 "finish"
$ns run
```

AWK Program for DV

```
BEGIN {
print " performance evaluation"
send=0
recv=0
dropped=0
rout=0
}
{
if($1=="+" && $3=="0"||"1" && $5=="cbr")
  {
 send++
   }
if($1=="r" && $4=="5" && $5=="cbr")
  {
 recv++
   }
if($1=="d")
  {
 dropped++
   }
if($1=="r" && $5=="rtProtoDV")
  {
 rout++
   }
}
END {
print "No of packets Send :" send
print "No of packets Received :" recv
print "No of packets dropped :" dropped
print "No of routing packets :" rout
NOH=rout/recv
PDR=recv/send
print "Normalised overhead :" NOH
print "Packet delivery ratio :" PDR
  }
```

OUTPUT:



13.422514 Step: 2.0ms

Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ☑ Recalc

```
down
down
```

RESULT:

The experiment was executed successfully.