Laboratory programs

1. Learn Networking Commands
2. Simulation of error correction code (like CRC).
3. Simulate the transmission of ping messages over a network topology consisting of 6 nodes.
4. Simulate a three nodes point – to – point network with duplex links between them. Set the queue size and vary the bandwidth and find the number of packets dropped.
5. Simulate a three nodes point – to – point network with duplex links between them. Set the queue size and vary the bandwidth and find the number of packets sent with different types of traffic.
6. Implement distance vector algorithm to find the suitable path for transmission between sender and receiver.
7. Simulation of Link State Routing algorithm.
8. Simulation of Routing Information Protocol.
9. Simulate an Ethernet LAN using n nodes, change error rate and data rate and compare throughput.
10. Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

1. Learn networking commands

ping 8.8.8.8
This command sends ICMP Echo Request packets to google's public DNS server (8.8.8.8) to check network connectivity and measure response time.

```
C:\WINDOWS\system32>ping 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=34ms TTL=118
Reply from 8.8.8.8: bytes=32 time=34ms TTL=118
Reply from 8.8.8.8: bytes=32 time=34ms TTL=118
Reply from 8.8.8.8: bytes=32 time=34ms TTL=118

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 34ms, Maximum = 34ms, Average = 34ms
```

Ping -l 1000 8.8.8.8
This command sends an ICMP Echo Request to Google's public DNS server (8.8.8.8) with a packet size of 1000 bytes to test network connectivity and latency

```
C:\WINDOWS\system32>ping -l 1000 8.8.8.8

Pinging 8.8.8.8 with 1000 bytes of data:
Reply from 8.8.8.8: bytes=1000 time=38ms TTL=118
Reply from 8.8.8.8: bytes=1000 time=54ms TTL=118
Reply from 8.8.8.8: bytes=1000 time=80ms TTL=118
Reply from 8.8.8.8: bytes=1000 time=35ms TTL=118

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 35ms, Maximum = 80ms, Average = 51ms
```

ping -n 5 8.8.8.8
This command sends 1000 ICMP Echo Request packets to Google's public DNS server to test network connectivity and response time.

```
C:\WINDOWS\system32>ping -n 5 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=34ms TTL=118
Reply from 8.8.8.8: bytes=32 time=35ms TTL=118
Reply from 8.8.8.8: bytes=32 time=35ms TTL=118
Reply from 8.8.8.8: bytes=32 time=33ms TTL=118
Reply from 8.8.8.8: bytes=32 time=34ms TTL=118

Ping statistics for 8.8.8.8:
    Packets: Sent = 5, Received = 5, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 33ms, Maximum = 35ms, Average = 34ms
```

Ping /a 8.8.8.8
The command ping /a 8.8.8.8 sends ping requests to the IP address **8.8.8.8**. The /a option tries to **resolve the IP address to a hostname** before sending the ping.

```
C:\WINDOWS\system32>ping /a 8.8.8.8

Pinging dns.google [8.8.8.8] with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=35ms TTL=118
Reply from 8.8.8.8: bytes=32 time=34ms TTL=118
Reply from 8.8.8.8: bytes=32 time=36ms TTL=118
Reply from 8.8.8.8: bytes=32 time=36ms TTL=118

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 34ms, Maximum = 36ms, Average = 35ms
```

Ping ipv4.google.com
This command checks connectivity by sending ICMP request to Google's IPv4 server

```
C:\WINDOWS\system32>ping ipv4.google.com

Pinging ipv4.l.google.com [142.250.195.78] with 32 bytes of data:
Reply from 142.250.195.78: bytes=32 time=54ms TTL=117
Reply from 142.250.195.78: bytes=32 time=54ms TTL=117
Reply from 142.250.195.78: bytes=32 time=57ms TTL=117
Reply from 142.250.195.78: bytes=32 time=54ms TTL=117

Ping statistics for 142.250.195.78:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 54ms, Maximum = 57ms, Average = 54ms
```

Ping -i 1 8.8.8.8
This command sets the Time to Live(TTL) value to 1 meaning the packet can only make only one hop before being discard

```
C:\WINDOWS\system32>ping -i 1 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 192.168.0.1: TTL expired in transit.
Reply from 192.168.0.1: TTL expired in transit.
Reply from 192.168.0.1: TTL expired in transit.
Reply from 192.168.0.1: TTL expired in transit.

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

Ping -i 15 8.8.8.8
This command sets the Time to Live(TTL) value to 15, limiting the packet to 15 hops before being discarded

```
C:\WINDOWS\system32>Ping -i 15 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=35ms TTL=118
Reply from 8.8.8.8: bytes=32 time=33ms TTL=118
Reply from 8.8.8.8: bytes=32 time=34ms TTL=118
Reply from 8.8.8.8: bytes=32 time=33ms TTL=118

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 33ms, Maximum = 35ms, Average = 33ms
```

Netstat -a

This command displays all active network connections and listening ports on the system.

```
C:\WINDOWS\system32>netstat -a

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            Sinchana-ML:0          LISTENING
  TCP    0.0.0.0:445            Sinchana-ML:0          LISTENING
  TCP    0.0.0.0:5040           Sinchana-ML:0          LISTENING
  TCP    0.0.0.0:49664          Sinchana-ML:0          LISTENING
  TCP    0.0.0.0:49665          Sinchana-ML:0          LISTENING
  TCP    0.0.0.0:49666          Sinchana-ML:0          LISTENING
  TCP    0.0.0.0:49667          Sinchana-ML:0          LISTENING
  TCP    0.0.0.0:49670          Sinchana-ML:0          LISTENING
  TCP    0.0.0.0:49674          Sinchana-ML:0          LISTENING
  TCP    127.0.0.1:27017        Sinchana-ML:0          LISTENING
  TCP    192.168.0.150:139      Sinchana-ML:0          LISTENING
  TCP    192.168.0.150:50149    4.213.25.240:https     ESTABLISHED
  TCP    192.168.0.150:50283    20.255.46.225:https    CLOSE_WAIT
  TCP    192.168.0.150:50306    sf-in-f188:5228        ESTABLISHED
  TCP    192.168.0.150:50307    52.187.79.109:https    ESTABLISHED
  TCP    192.168.0.150:50324    whatsapp-chatd-edge-shv-01-pnq1:5222  ESTABLISHED
  TCP    192.168.0.150:50327    52.97.92.114:https     ESTABLISHED
  TCP    192.168.0.150:50379    13.107.226.254:https   CLOSE_WAIT
  TCP    192.168.0.150:50380    13.107.219.254:https   CLOSE_WAIT
  TCP    192.168.0.150:50390    20.69.137.228:https    TIME_WAIT
  TCP    192.168.0.150:50392    199.232.214.172:http   TIME_WAIT
  TCP    192.168.0.150:50393    20.69.137.228:https    ESTABLISHED
  TCP    192.168.0.150:50394    a23-11-215-154:https   ESTABLISHED
  TCP    192.168.0.150:50396    a23-11-215-154:https   ESTABLISHED
  TCP    192.168.0.150:50397    40.104.77.82:https     ESTABLISHED
  TCP    192.168.0.150:50398    104.208.16.90:https    ESTABLISHED
  TCP    192.168.0.150:50399    13.107.213.254:https   ESTABLISHED
  TCP    192.168.0.150:50400    150.171.73.254:https   ESTABLISHED
  TCP    192.168.0.150:50401    13.107.253.37:https    ESTABLISHED
  TCP    192.168.0.150:50402    204.79.197.222:https   ESTABLISHED
  TCP    [::]:135               Sinchana-ML:0          LISTENING
  TCP    [::]:445               Sinchana-ML:0          LISTENING
  TCP    [::]:49664             Sinchana-ML:0          LISTENING
  TCP    [::]:49665             Sinchana-ML:0          LISTENING
  TCP    [::]:49666             Sinchana-ML:0          LISTENING
  TCP    [::]:49667             Sinchana-ML:0          LISTENING
  TCP    [::]:49670             Sinchana-ML:0          LISTENING
  TCP    [::]:49674             Sinchana-ML:0          LISTENING
  TCP    [::1]:49671            Sinchana-ML:0          LISTENING
  UDP    0.0.0.0:5050           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
  UDP    0.0.0.0:5353           *:*
```

Netstat -n

This command displays active network connections showing IP addresses and port numbers in numeric form instead of resolving hostnames.

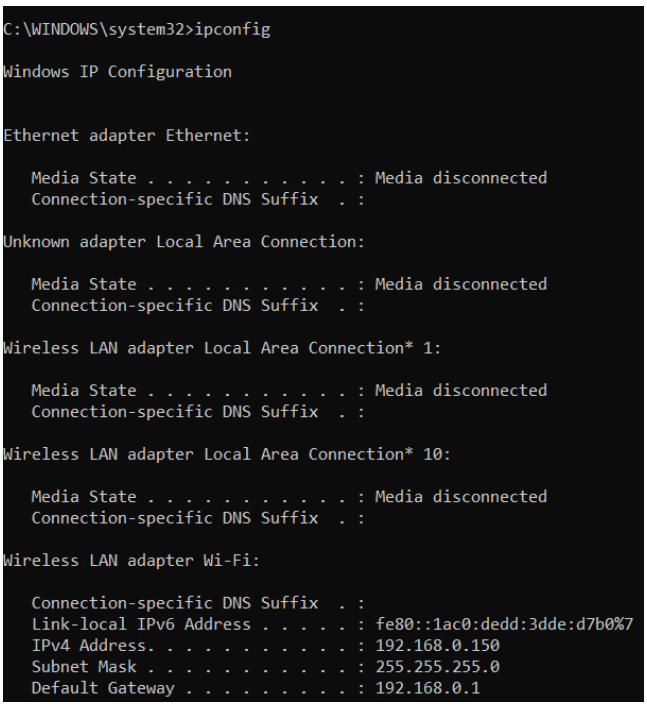
```
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    192.168.0.150:50149    4.213.25.240:443       ESTABLISHED
  TCP    192.168.0.150:50283    20.255.46.225:443      CLOSE_WAIT
  TCP    192.168.0.150:50306    74.125.24.188:5228     ESTABLISHED
  TCP    192.168.0.150:50307    52.187.79.109:443      ESTABLISHED
  TCP    192.168.0.150:50324    157.240.242.61:5222    ESTABLISHED
  TCP    192.168.0.150:50327    52.97.92.114:443       ESTABLISHED
  TCP    192.168.0.150:50379    13.107.226.254:443     CLOSE_WAIT
  TCP    192.168.0.150:50380    13.107.219.254:443     CLOSE_WAIT
  TCP    192.168.0.150:50394    23.11.215.154:443      CLOSE_WAIT
  TCP    192.168.0.150:50396    23.11.215.154:443      CLOSE_WAIT
  TCP    192.168.0.150:50397    40.104.77.82:443       ESTABLISHED
  TCP    192.168.0.150:50399    13.107.213.254:443     CLOSE_WAIT
  TCP    192.168.0.150:50401    13.107.253.37:443      CLOSE_WAIT
  TCP    192.168.0.150:50406    172.64.155.209:443     TIME_WAIT
  TCP    192.168.0.150:50409    104.18.32.47:443       TIME_WAIT
  TCP    192.168.0.150:50415    20.189.173.9:443       TIME_WAIT
  TCP    192.168.0.150:50416    13.68.233.9:443        TIME_WAIT
  TCP    192.168.0.150:50417    52.109.124.29:443      TIME_WAIT
  TCP    192.168.0.150:50418    20.42.65.89:443        ESTABLISHED
  TCP    192.168.0.150:50419    13.68.233.9:443        ESTABLISHED
```

Netstat -r

This command displays all system's routing table, showing network destinations, gateway and interfaces.

```
C:\WINDOWS\system32>Netstat -r
===========================================================================
Interface List
  4...54 ee 75 e0 b4 5b ......Intel(R) Ethernet Connection (4) I219-LM
 14...00 ff d5 1c e4 d7 ......TAP-Windows Adapter V9
 19...bc a8 a6 e9 0c d3 ......Microsoft Wi-Fi Direct Virtual Adapter
 15...be a8 a6 e9 0c d2 ......Microsoft Wi-Fi Direct Virtual Adapter #2
  7...bc a8 a6 e9 0c d2 ......Intel(R) Dual Band Wireless-AC 8265
  1...........................Software Loopback Interface 1
===========================================================================

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      192.168.0.1    192.168.0.150     55
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
      192.168.0.0    255.255.255.0         On-link     192.168.0.150    311
    192.168.0.150  255.255.255.255         On-link     192.168.0.150    311
    192.168.0.255  255.255.255.255         On-link     192.168.0.150    311
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link     192.168.0.150    311
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link     192.168.0.150    311
===========================================================================
Persistent Routes:
  None

IPv6 Route Table
===========================================================================
Active Routes:
 If Metric Network Destination      Gateway
  1    331 ::1/128                  On-link
  7    311 fe80::/64                On-link
  7    311 fe80::1ac0:dedd:3dde:d7b0/128
                                    On-link
  1    331 ff00::/8                 On-link
  7    311 ff00::/8                 On-link
===========================================================================
Persistent Routes:
  None
```

Ipconfig
This command displays the system's IP configuration, including assigned IP address, subnet masks and default gateways for all network interfaces.



Ipconfig/release

This command re leases the current IP address assigned to all network adopters by DHCP server.

```
C:\WINDOWS\system32>Ipconfig/release

Windows IP Configuration

No operation can be performed on Ethernet while it has its media disconnected.
No operation can be performed on Local Area Connection while it has its media disconnected.
No operation can be performed on Local Area Connection* 1 while it has its media disconnected.
No operation can be performed on Local Area Connection* 10 while it has its media disconnected.

Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Unknown adapter Local Area Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 10:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::1ac0:dedd:3dde:d7b0%7
   Default Gateway . . . . . . . . . :
```

Ipconfig/renew
This command requests a new IP address from the DHCP server for all network adopters.

```
C:\WINDOWS\system32>Ipconfig/renew

Windows IP Configuration

No operation can be performed on Ethernet while it has its media disconnected.
No operation can be performed on Local Area Connection while it has its media disconnected.
No operation can be performed on Local Area Connection* 1 while it has its media disconnected.
No operation can be performed on Local Area Connection* 10 while it has its media disconnected.

Ethernet adapter Ethernet:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Unknown adapter Local Area Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 10:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::1ac0:dedd:3dde:d7b0%7
   IPv4 Address. . . . . . . . . . . : 192.168.0.150
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.0.1
```

Tracert -d google.com
This command traces theroute to google.com without resolving IP address to hostnames, making the process faster.

```
C:\WINDOWS\system32>Tracert -d google.com

Tracing route to google.com [142.250.196.174]
over a maximum of 30 hops:

  1     9 ms     6 ms     2 ms  192.168.0.1
  2     8 ms     7 ms     7 ms  172.17.28.1
  3     7 ms     7 ms    10 ms  118.151.209.161
  4     7 ms     8 ms     7 ms  118.151.209.9
  5    35 ms    34 ms    36 ms  118.151.209.126
  6    40 ms    38 ms    34 ms  192.178.110.221
  7    39 ms    35 ms    37 ms  192.178.110.198
  8    58 ms    62 ms    61 ms  72.14.232.35
  9    63 ms    60 ms    70 ms  142.250.62.67
 10    57 ms    59 ms    61 ms  108.170.231.131
 11    65 ms    67 ms   116 ms  142.250.196.174
```

Tracert -h 10 google.com
This command traces the route to google.comlimiting the maximum hops to 10.

```
C:\WINDOWS\system32>Tracert -h 10 google.com

Tracing route to google.com [142.250.192.14]
over a maximum of 10 hops:

  1     1 ms     1 ms     2 ms  192.168.0.1
  2     7 ms     6 ms     6 ms  172.17.28.1
  3     8 ms     7 ms     8 ms  yash-static-161.209.151.118.yashtel.co.in [118.151.209.161]
  4    41 ms    52 ms     8 ms  yash-static-9.209.151.118.yashtel.co.in [118.151.209.9]
  5    35 ms    68 ms    42 ms  yash-static-126.209.151.118.yashtel.co.in [118.151.209.126]
  6    40 ms    36 ms    38 ms  192.178.110.123
  7    35 ms    34 ms    34 ms  142.250.208.223
  8    34 ms    33 ms    33 ms  bom12s14-in-f14.1e100.net [142.250.192.14]

Trace complete.
```

Tracert -w 100 google.com
This command traces the route to google.com, setting the timeout for each reply to 100 milliseconds.

```
C:\WINDOWS\system32>Tracert -w 100 google.com

Tracing route to google.com [142.251.42.46]
over a maximum of 30 hops:

  1     1 ms     2 ms     5 ms  192.168.0.1
  2     8 ms     5 ms     8 ms  172.17.28.1
  3     8 ms     6 ms     6 ms  yash-static-161.209.151.118.yashtel.co.in [118.151.209.161]
  4     9 ms     7 ms     8 ms  yash-static-9.209.151.118.yashtel.co.in [118.151.209.9]
  5    36 ms    35 ms    34 ms  yash-static-126.209.151.118.yashtel.co.in [118.151.209.126]
  6    36 ms    35 ms    35 ms  192.178.110.123
  7    36 ms    33 ms    33 ms  142.251.69.45
  8    33 ms    34 ms    33 ms  bom12s20-in-f14.1e100.net [142.251.42.46]
```

Nslookup -type=mx google.com
This command queries the Mail exchange(MX) records of google.com displaying the mail servers.

Nslookup -domainname.tld

```
C:\WINDOWS\system32>Nslookup -type=mx google.com
Server:  UnKnown
Address:  192.168.0.1

Non-authoritative answer:
google.com      MX preference = 10, mail exchanger = smtp.google.com
```

This queries the DNS to find the IP address of the given domain.

```
C:\WINDOWS\system32>Nslookup -domainname.tld
Default Server:  UnKnown
Address:  192.168.0.1
```

Nslookup yahoo.com
This command attempts to query the DNS for the Ip address of yahoo.com

```
C:\WINDOWS\system32>Nslookup yahoo.com
Server:  UnKnown
Address:  192.168.0.1

DNS request timed out.
    timeout was 2 seconds.
DNS request timed out.
    timeout was 2 seconds.
Non-authoritative answer:
Name:    yahoo.com
Addresses:  2001:4998:44:3507::8000
          2001:4998:24:120d::1:0
          2001:4998:124:1507::f000
          2001:4998:124:1507::f001
          2001:4998:24:120d::1:1
          2001:4998:44:3507::8001
          98.137.11.163
          98.137.11.164
          74.6.231.20
          74.6.143.25
          74.6.231.21
          74.6.143.26
```

Nslookup mcehassan.ac.in
This command queries the DNS server to retrieve the IP address associated with the domain mcehassan.ac.in

```
C:\WINDOWS\system32>Nslookup mcehassan.ac.in
Server:  UnKnown
Address:  192.168.0.1

Non-authoritative answer:
Name:    mcehassan.ac.in
Addresses:  3.6.22.33
          13.127.160.254
```

Route -n

This command displays the routing table with numeric IP addresses, avoidinghostname resolution for faster output

```
C:\WINDOWS\system32>route -n

Manipulates network routing tables.

ROUTE [-f] [-p] [-4|-6] command [destination]
                  [MASK netmask]  [gateway] [METRIC metric]  [IF interface]

  -f           Clears the routing tables of all gateway entries.  If this is
               used in conjunction with one of the commands, the tables are
               cleared prior to running the command.

  -p           When used with the ADD command, makes a route persistent across
               boots of the system. By default, routes are not preserved
               when the system is restarted. Ignored for all other commands,
               which always affect the appropriate persistent routes.

  -4           Force using IPv4.

  -6           Force using IPv6.

  command      One of these:
                 PRINT     Prints  a route
                 ADD       Adds    a route
                 DELETE    Deletes a route
                 CHANGE    Modifies an existing route
  destination  Specifies the host.
  MASK         Specifies that the next parameter is the 'netmask' value.
  netmask      Specifies a subnet mask value for this route entry.
               If not specified, it defaults to 255.255.255.255.
  gateway      Specifies gateway.
  interface    the interface number for the specified route.
  METRIC       specifies the metric, ie. cost for the destination.

All symbolic names used for destination are looked up in the network database
file NETWORKS. The symbolic names for gateway are looked up in the host name
database file HOSTS.

If the command is PRINT or DELETE. Destination or gateway can be a wildcard,
(wildcard is specified as a star '*'), or the gateway argument may be omitted.

If Dest contains a * or ?, it is treated as a shell pattern, and only
matching destination routes are printed. The '*' matches any string,
and '?' matches any one char. Examples: 157.*.1, 157.*, 127.*, *224*.

Pattern match is only allowed in PRINT command.
Diagnostic Notes:
    Invalid MASK generates an error, that is when (DEST & MASK) != DEST.
    Example> route ADD 157.0.0.0 MASK 155.0.0.0 157.55.80.1 IF 1
```

Route -a

```
Manipulates network routing tables.

ROUTE [-f] [-p] [-4|-6] command [destination]
                  [MASK netmask]  [gateway] [METRIC metric]  [IF interface]

  -f           Clears the routing tables of all gateway entries.  If this is
               used in conjunction with one of the commands, the tables are
               cleared prior to running the command.

  -p           When used with the ADD command, makes a route persistent across
               boots of the system. By default, routes are not preserved
               when the system is restarted. Ignored for all other commands,
               which always affect the appropriate persistent routes.

  -4           Force using IPv4.

  -6           Force using IPv6.

  command      One of these:
                 PRINT     Prints  a route
                 ADD       Adds    a route
                 DELETE    Deletes a route
                 CHANGE    Modifies an existing route
  destination  Specifies the host.
  MASK         Specifies that the next parameter is the 'netmask' value.
  netmask      Specifies a subnet mask value for this route entry.
               If not specified, it defaults to 255.255.255.255.
  gateway      Specifies gateway.
  interface    the interface number for the specified route.
  METRIC       specifies the metric, ie. cost for the destination.

All symbolic names used for destination are looked up in the network database
file NETWORKS. The symbolic names for gateway are looked up in the host name
database file HOSTS.

If the command is PRINT or DELETE. Destination or gateway can be a wildcard,
(wildcard is specified as a star '*'), or the gateway argument may be omitted.

If Dest contains a * or ?, it is treated as a shell pattern, and only
matching destination routes are printed. The '*' matches any string,
and '?' matches any one char. Examples: 157.*.1, 157.*, 127.*, *224*.

Pattern match is only allowed in PRINT command.
Diagnostic Notes:
    Invalid MASK generates an error, that is when (DEST & MASK) != DEST.
    Example> route ADD 157.0.0.0 MASK 155.0.0.0 157.55.80.1 IF 1
             The route addition failed: The specified mask parameter is invalid. (Destination & Mask) != Destination.
```

Arp -a

This command displays the ARP showing IP addresses and their corresponding MAC(physical) addresses of devices in the local network

```
C:\WINDOWS\system32>arp -a

Interface: 192.168.0.150 --- 0x7
  Internet Address      Physical Address      Type
  192.168.0.1           00-31-92-b0-c6-2d     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.2             01-00-5e-00-00-02     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

2. Simulation of error correction code (like CRC)

```
#include <stdio.h> // Include standard input-output library
// Declare global variables
char data[20], div[20], temp[4], total[100]; int
i, j, datalen, divlen, len, flag = 1;
void check(); // Function prototype for CRC check int
main()
{
// Ask user for the total number of data bits
printf("Enter the total bit of data:"); scanf("%d",
&datalen);
// Ask user for the total number of divisor bits
printf("\nEnter the total bit of divisor");
scanf("%d", &divlen);
// Calculate the length of total code word (data + remainder length) len
= datalen + divlen - 1;
// Ask user to enter the data bits
printf("\nEnter the data:"); scanf("%s",
&data);
// Ask user to enter the divisor bits
printf("\nEnter the divisor"); scanf("%s",
div);
// Copy the data bits into total and temp arrays for (i
= 0; i < datalen; i++)
{
total[i] = data[i]; // Store data in total array temp[i] =
data[i]; // Store data in temp array
}

  // Append zeroes at the end of total array (equivalent to divisor length - 1) for
(i = datalen; i < len; i++)
total[i] = '0';

// Perform CRC division to get remainder
check();
// Append the CRC remainder to the original data for (i
= 0; i < divlen; i++)
   temp[i + datalen] = data[i];
// Print the transmitted code word (data + CRC remainder)
printf("\ntransmitted Code Word:%s", temp);
// Receive the code word at the receiver side
printf("\n\nEnter the received code word:");
scanf("%s", total);
```

```
    // Perform CRC check at the receiver side
    check();
    // Check if the remainder is all zeroes for (i
    = 0; i < divlen - 1; i++)
        if (data[i] == '1') // If any bit in remainder is '1', there is an error
        {
            flag = 0; break;
        }
    // If remainder is all zeroes, transmission is successful if
    (flag == 1)
        printf("\nsuccessful!!"); else
        printf("\nreceived code word contains errors...\n");
}
// Function to perform CRC division (binary division using XOR) void
check()
{
    // Copy first divlen bits from total into data array for (j
    = 0; j < divlen; j++)
        data[j] = total[j];
    // Perform division using XOR until all bits are processed while
    (j <= len)
    {
        // If the first bit is '1', perform XOR with the divisor if
        (data[0] == '1')
            for (i = 1; i < divlen; i++)
                data[i] = ((data[i] == div[i]) ? '0' : '1'); // XOR operation

        // Left shift the remaining bits to prepare for next XOR for (i
        = 0; i < divlen - 1; i++)
            data[i] = data[i + 1];

        // Bring the next bit from total into the shifted position data[i]
        = total[j++];
    }
}
```

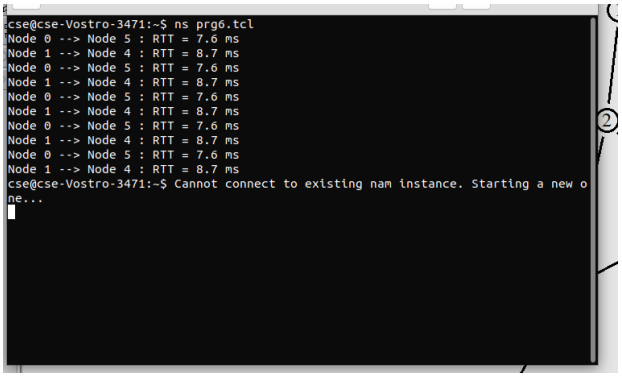
```
Enter the total bit of data:4
Enter the total bit of divisor:3
Enter the data:1011
Enter the divisor:101
transmitted Code Word:101101
Enter the received code word:101101
successful!!
```

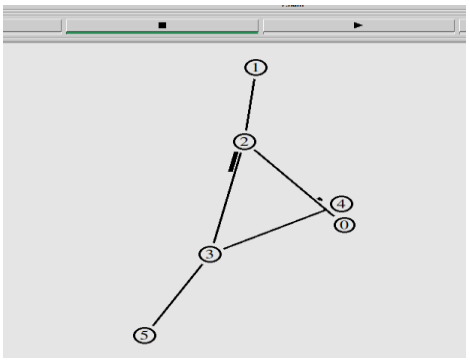
```
Enter the total bit of data:4
Enter the total bit of divisor:3
Enter the data:1011
Enter the divisor:110
transmitted Code Word:101110
Enter the received code word:111110
received code word contains errors...
```

3. Simulate the transmission of ping messages over a network topology consisting of 6 nodes.

```
# Create a simulator object set ns
[new Simulator]
# Define colors for flows
$ns color 1 Red
$ns color 2 Green #
Create trace file
        set nt [open "7.tr" w]
        $ns trace-all $nt
        # Create NAM file
                set na [open "7.nam" w]
        $ns namtrace-all
        $na # Create nodes
set n0 [$ns node] set n1
[$ns node] set n2 [$ns
node] set n3 [$ns node]
set n4 [$ns node] set n5
[$ns node] # Create
links
$ns duplex-link $n0 $n2 10Mb 1ms DropTail
$ns duplex-link $n1 $n2 10Mb 1ms DropTail
$ns duplex-link $n2 $n3 1Mb 1ms DropTail
$ns duplex-link $n3 $n4 1Mb 1ms DropTail
$ns duplex-link $n3 $n5 2Mb 1ms DropTail # Set
queue limits
$ns queue-limit $n2 $n3 3
$ns queue-limit $n3 $n2 3 #
Create Ping agents
set Ping1 [new Agent/Ping] set
Ping2 [new Agent/Ping] set Ping3
[new Agent/Ping] set Ping4 [new
Agent/Ping] # Attach agents to
nodes
$ns attach-agent $n0 $Ping1
$ns attach-agent $n1 $Ping2
$ns attach-agent $n4 $Ping3
$ns attach-agent $n5 $Ping4 #
Connect agents
$ns connect $Ping1 $Ping4
$ns connect $Ping2 $Ping3 # Set
```

```
class for colors
$Ping1 set class_ 1
$Ping2 set class_ 2
# Define recv procedure for Ping agent Agent/Ping
instproc recv {from rtt} {
   $self instvar node_
   puts "Node [$node_ id] --> Node $from : RTT = $rtt ms"
}
# Define end procedure proc end
{} {
   global ns nt na
   $ns flush-trace close
   $nt
   close $na
   exec nam 7.nam & exit 0
}
# Schedule Ping sends
for {set t 0} {$t < 5.0} {set t [expr $t + 1.0]} {
   $ns at $t "$Ping1 send"
   $ns at $t "$Ping2 send"
}
# Schedule end
$ns at 6.0 "end" # Run
simulation
$ns run
```





4. Simulate a three nodes point – to – point network with duplex links between them. Set the queue size and vary the bandwidth and find the number of packets dropped.

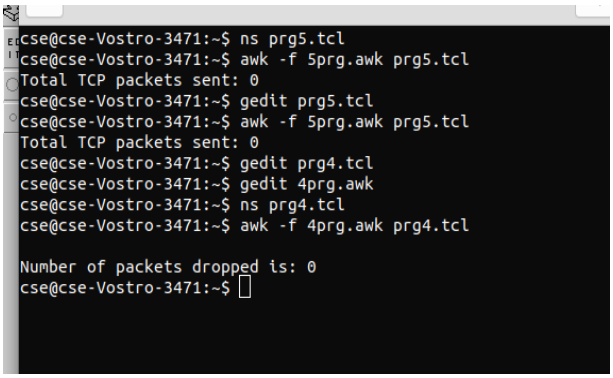
```
#Create a simulator object set ns
[new Simulator]
#Define different colors for data flows
$ns color 1 Purple
$ns color 2 MAgenta #create a
trace file
set mytrace [open prog1.tr w]
$ns trace-all $mytrace #Open
the nam trace file set nf [open
out.nam w]
$ns namtrace-all $nf #Define a
'finish' procedure proc finish {} {
global ns nf
$ns flush-trace #Close the
trace file close $nf
#Execute nam on the trace file exec
nam out.nam &
exit 0
}
#Create four nodes set n0
[$ns node] set n1 [$ns
node] set n2 [$ns node]
set n3 [$ns node] set n4
[$ns node]
# Specify color and shape for nodes
$n0 color Purple
$n3 color Purple
$n1 color MAgenta
$n1 shape box
$n2 color MAgenta
$n2 shape box
#Create links between the nodes
$ns duplex-link $n0 $n4 1Mb 10ms DropTail
$ns duplex-link $n1 $n4 1Mb 10ms DropTail
$ns duplex-link $n2 $n4 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 2Mb 10ms DropTail
$ns duplex-link-op $n0 $n4 orient right-down
```

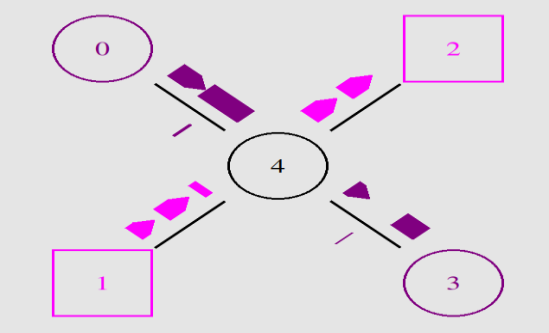
```
$ns duplex-link-op $n1 $n4 orient right-up
$ns duplex-link-op $n2 $n4 orient left-down
$ns duplex-link-op $n3 $n4 orient left-up
#Monitor the queue for the link between node 2 and node 3
$ns duplex-link-op $n4 $n3 queuePos 0.5
$ns duplex-link-op $n4 $n2 queuePos 0.5 #Create
a TCP agent and attach it to node n0 set tcp0 [new
Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n0 $tcp0
# Create a CBR traffic source and attach it to udp0 set
cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $tcp0
#Create a UDP agent and attach it to node n1 set
udp0 [new Agent/UDP]
$udp0 set class_ 2
$ns attach-agent $n1 $udp0
# Create a CBR traffic source and attach it to udp1 set
cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp0
#Create a Null agent (a traffic sink) and attach it to node n3 set
tsink0 [new Agent/TCPSink]
$ns attach-agent $n3 $tsink0 set
null1 [new Agent/Null]
$ns attach-agent $n2 $null1
#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $tsink0
$ns connect $udp0 $null1 #$ns at
0.0 "$n0 TCPSource" #$ns at 0.0
"$n1 UDPSource" #$ns at 0.0 "$n4
CenterNode" #$ns at 0.0 "$n2
UDPNull" #$ns at 0.0 "$n3
TCPSink"
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish" #Run
the simulation
```

$ns run

Awk file:

```
BEGIN{Count=0;}
{
if($1=="d")
Count++;
} END{
printf("\nNumber of packets dropped is: %d\n",Count);
}
```





5. Simulate a three nodes point – to – point network with duplex links between them. Set the queue size and vary the bandwidth and find the number of packets sent with different types of traffic

```
#Create a simulator object set
ns [new Simulator]
#Define different colors for data flows
$ns color 1 Purple
$ns color 2 MAgenta
#create a trace file
set mytrace [open prog1.tr w]
$ns trace-all $mytrace
#Open the nam trace file set
nf [open out.nam w]
$ns namtrace-all $nf
#Define a 'finish' procedure
proc finish {} {
global ns nf
$ns flush-trace #Close
the trace file close $nf
#Execute nam on the trace file
exec nam out.nam &
exit 0
}
#Create four nodes set
n0 [$ns node] set n1
[$ns node] set n2 [$ns
node] set n3 [$ns
node] set n4 [$ns
node]
# Specify color and shape for nodes
$n0 color Purple
$n3 color Purple
$n1 color MAgenta
$n1 shape box
$n2 color MAgenta
$n2 shape box
#$ns at 0.0 "$n0 TCPSource"
$ns at 0.0 "$n0 label TCPSource"
$ns at 0.0 "$n1 label UDPSource"
$ns at 0.0 "$n4 label CenterNode"
$ns at 0.0 "$n2 label UDPNull"
$ns at 0.0 "$n3 label TCPSink"
#$ns at 0.0 "$n1 UDPSource"
```

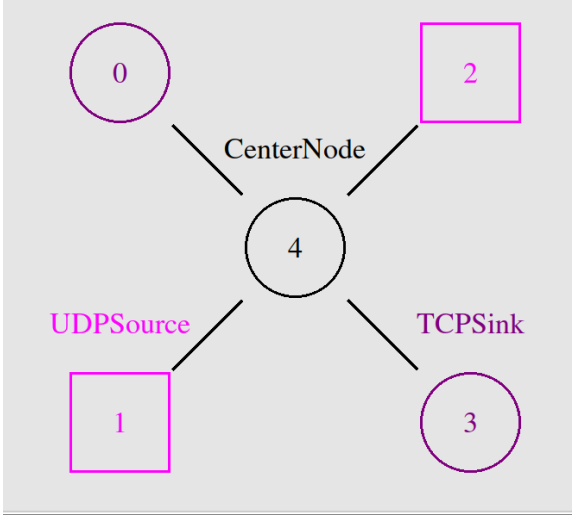
```
#$ns at 0.0 "$n4 CenterNode"
#$ns at 0.0 "$n2 UDPNull"
#$ns at 0.0 "$n3 TCPSink"
#Create links between the nodes
$ns duplex-link $n0 $n4 1Mb 10ms DropTail
$ns duplex-link $n1 $n4 1Mb 10ms DropTail
$ns duplex-link $n2 $n4 0.5Mb 10ms DropTail
$ns queue-limit $n2 $n4 3
$ns duplex-link $n3 $n4 0.5Mb 10ms DropTail
$ns queue-limit $n3 $n4 3
$ns duplex-link-op $n0 $n4 orient right-down
$ns duplex-link-op $n1 $n4 orient right-up
$ns duplex-link-op $n2 $n4 orient left-down
$ns duplex-link-op $n3 $n4 orient left-up
#Monitor the queue for the link between node 2 and node 3
$ns duplex-link-op $n4 $n3 queuePos 0.5
$ns duplex-link-op $n4 $n2 queuePos 0.5
#Create a TCP agent and attach it to node n0 set
tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n0 $tcp0
# Create a FTP source and attach it to tcp0 set
ftp0 [new Application/FTP]
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.005
$ftp0 attach-agent $tcp0
#Create a UDP agent and attach it to node n1 set
udp0 [new Agent/UDP]
$udp0 set class_ 2
$ns attach-agent $n1 $udp0
# Create a CBR traffic source and attach it to udp1 set
cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp0
#Create a Null agent (a traffic sink) and attach it to node n3 set
tsink0 [new Agent/TCPSink]
$ns attach-agent $n3 $tsink0
set null1 [new Agent/Null]
$ns attach-agent $n2 $null1
#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $tsink0
$ns connect $udp0 $null1
#Schedule events for the CBR agents
```

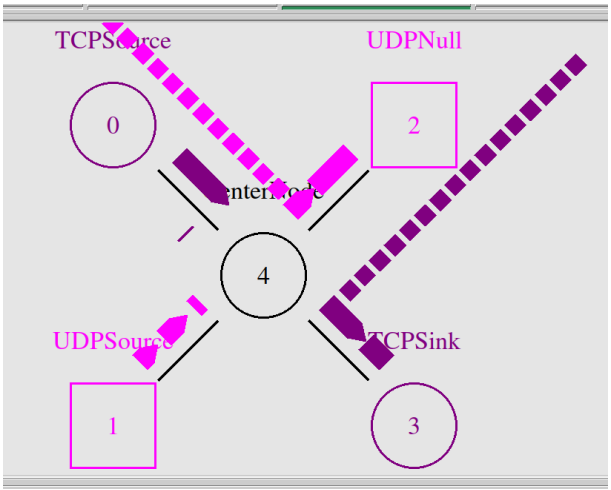
```
$ns at 0.5 "$ftp0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$ftp0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

Awk file

```
BEGIN {
dcount = 0;
scount = 0;
}
{
  event = $1; if(event == "d")
{
dcount++;
}
if(event == "+")
{
scount++;
}
}
END {
printf("The no.of packets dropped : %d\n ",dcount);
printf("The no.of packets sent : %d\n ",scount);
}
```

```
cse@cse-Vostro-3471:~$ ns 5prg.tcl
cse@cse-Vostro-3471:~$ awk -f prg5.awk 5prg.tcl
The no.of packets dropped : 0
 The no.of packets sent : 0
cse@cse-Vostro-3471:~$
```





6. Implement distance vector algorithm to find the suitable path for transmission between sender and receiver.

```
#include <stdio.h>
struct node
{
   int dist[20]; int
   from[20];
} route[10];
int main()
{
   int dm[20][20], n;
   printf("Enter the number of nodes: \n");
   scanf("%d", &n);
   printf("Enter the distance matrix: \n");
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         scanf("%d", &dm[i][j]);
         dm[i][i] = 0; route[i].dist[j] =
         dm[i][j]; route[i].from[j] = j;
      }
   }
   int flag; do
   {
      flag = 0;
      for (int i = 0; i < n; i++)
      {
         for (int j = 0; j < n; j++)
         {
            for (int k = 0; k < n; k++)
            {
               if ((route[i].dist[j]) > (route[i].dist[k] + route[k].dist[j]))
               {
                  route[i].dist[j] = route[i].dist[k] + route[k].dist[j];
                  route[i].from[j] = k;
                  flag = 1;
               }
            }
         }
      }
   } while (flag);
   for (int i = 0; i < n; i++)
   {  printf("\nRouter info for router: %d\n", i + 1); printf("Dest.\tNext Hop\tDist.\n");
```

```
    for (int j = 0; j < n; j++)
       printf("%d\t%d\t\t%d\n", j + 1, route[i].from[j] + 1, route[i].dist[j]);
       }
       return  0;}
```

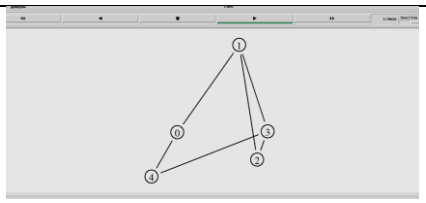
```
Enter the number of nodes:
5
Enter the distance matrix:
0 3 5 99 99
3 0 99 2 1
5 99 0 1 99
99 2 1 0 8
99 1 99 8 0

Router info for router: 1
Dest.    Next Hop     Dist.
1                1            0
2                2            3
3                3            5
4                2            5
5                2            4

Router info for router: 2
Dest.    Next Hop     Dist.
1                1            3
2                2            0
3                4            3
4                4            2
5                5            1

Router info for router: 3
Dest.    Next Hop     Dist.
1                1            5
2                4            3
3                3            0
4                4            1
5                2            4

Router info for router: 4
Dest.    Next Hop     Dist.
1                2            5
2                2            2
3                3            1
4                4            0
5                2            3

Router info for router: 5
Dest.    Next Hop     Dist.
1                2            4
2                2            1
3                2            4
4                2            3
5                5            0
```

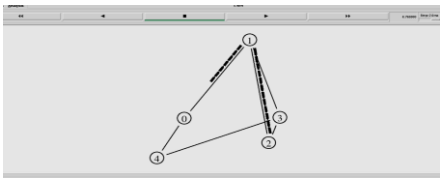
7. Simulation of Link State Routing algorithm.

```
# Define the simulation environment set ns
[new Simulator]
#create a trace file set
nt [open 5.tr w]
$ns trace-all $nt
#Open the nam trace file set
na [open 5.nam w]
$ns namtrace-all $na
# Create nodes
set  node0  [$ns  node]
set  node1  [$ns  node]
set  node2  [$ns  node]
set  node3  [$ns  node]
set node4 [$ns node]
# Create links between nodes with different delays and bandwidths
$ns duplex-link $node0 $node1 10Mb 10ms DropTail
$ns duplex-link $node1 $node2 10Mb 10ms DropTail
$ns duplex-link $node2 $node3 10Mb 10ms DropTail
$ns duplex-link $node3 $node4 10Mb 10ms DropTail
$ns duplex-link $node0 $node4 5Mb 50ms DropTail
$ns duplex-link $node1 $node3 5Mb 30ms DropTail #Implement
Link State Updates:
#To simulate a simple link state routing mechanism, we can manually configure the routes
according to the precomputed shortest path.
# Manual routing to simulate link state updates
$ns rtproto LS
# Simulate Link State Routing protocol
# Assume node0 knows the full topology and updates its routes
$node0 add-route $node1 1
$node0 add-route $node2 2
$node0 add-route $node3 2
$node0 add-route $node4 1
#Chooses direct link as shortest path despite higher delay # Assume
node1 updates its routes
$node1 add-route $node0 1
$node1 add-route $node2 1
$node1 add-route $node3 1
$node1 add-route $node4 2 #
Through node2
# Similar updates for other nodes based on full topology knowledge
$node2 add-route $node0 2
$node2 add-route $node1 1
$node2 add-route $node3 1
```

```
$node2 add-route $node4 2 #
Through node3
$node3 add-route $node0 2
$node3 add-route $node1 1
$node3 add-route $node2 1
$node3 add-route $node4 1 #
Through node4
$node4 add-route $node0 1
$node4 add-route $node1 2
$node4 add-route $node2 2
$node4 add-route $node3 1
#Simulate Traffic to Test the Routing Protocol:
# Attach TCP agents and applications to test routing set tcp1
[new Agent/TCP]
$ns attach-agent $node0 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $node2 $sink1
$ns connect $tcp1 $sink1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
# Start the FTP application
$ns at 1.0 "$ftp1 start"
# Add another traffic source to test multi-path routing set udp1
[new Agent/UDP]
$ns attach-agent $node1 $udp1
set sink2 [new Agent/Null]
$ns attach-agent $node4 $sink2
$ns connect $udp1 $sink2
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packetSize_ 512
$cbr1 set rate_ 1Mb
# Start the CBR application
$ns at 2.0 "$cbr1 start"
$ns at 8.0 "$cbr1 stop"
# Define the finish procedure proc
finish { } {
global ns nt na
$ns flush-trace
close $na
close $nt
exec nam 5.nam & exit
0
}
# Schedule the simulation end
```

```
$ns at 10.0 "finish"
# Run the simulation
$ns run
```

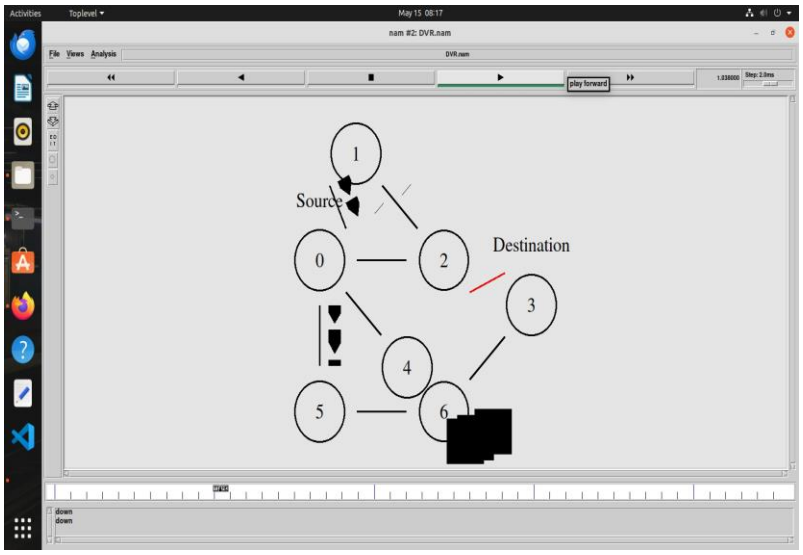




8. Simulation of Routing Information Protocol.

```
#Create a simulator object set ns
[new Simulator]
#Use distance vector routing
$ns rtproto DV
#Open the nam trace file set nf
[open DVR.nam w]
$ns namtrace-all $nf
# Open tracefile
set tr [open DVR.tr w]
$ns trace-all $tr
#Define 'finish' procedure proc
finish {} {
global ns nf tr
$ns flush-trace #Close the
trace file close $nf
#Execute nam on the trace file exec nam -a
DVR.nam &
exit 0
}
# Create 7 nodes set n0
[$ns node] set n1 [$ns
node]   set   n2   [$ns
node]   set   n3   [$ns
node]   set   n4   [$ns
node]   set   n5   [$ns
node]   set   n6   [$ns
node]
# Specify link characterestics
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n0 $n4 1Mb 10ms DropTail
$ns duplex-link $n0 $n5 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n6 1Mb 10ms DropTail
$ns duplex-link $n5 $n6 1Mb 10ms DropTail
# specify layout as a indirected graph
$ns duplex-link-op $n0 $n1 orient right-up
$ns duplex-link-op $n0 $n2 orient right
$ns duplex-link-op $n0 $n4 orient right-down
$ns duplex-link-op $n0 $n5 orient down
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right-down
```

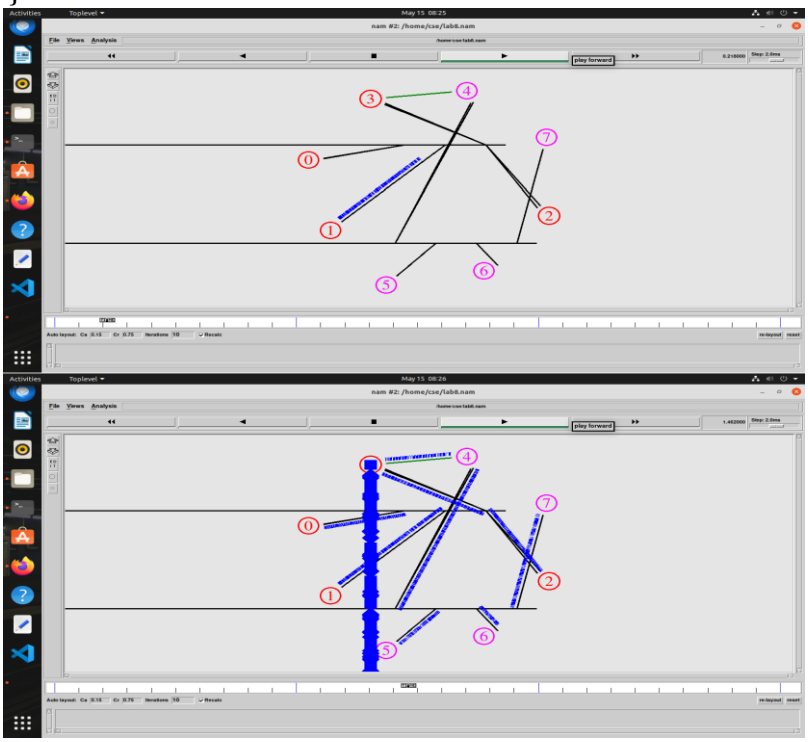
```
$ns duplex-link-op $n5 $n6 orient right
$ns duplex-link-op $n6 $n3 orient right-up
#Create a UDP agent and attach it to node n0 set udp0 [new
Agent/UDP]
$ns attach-agent $n0 $udp0
#Create a CBR traffic source and attach it to udp0 set cbr0 [new
Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
#Create a Null agent (a traffic sink) and attach it to node n3 set null0 [new
Agent/Null]
$ns attach-agent $n3 $null0
#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0
#Schedule events for the CBR agent and the network dynamics
$ns at 0.0 "$n0 label Source"
$ns at 0.0 "$n3 label Destination"
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n2 $n3
$ns rtmodel-at 2.0 up $n2 $n3
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```



9. Simulate an Ethernet LAN using n nodes, change error rate and data rate and compare throughput.

```
set ns [new Simulator] set tf
[open lab8.tr w]
$ns trace-all $tf
set nf [open lab8.nam w]
$ns namtrace-all $nf
$ns color 0 blue
set n0 [$ns node]
$n0 color "red" set n1
[$ns node]
$n1 color "red" set n2
[$ns node]
$n2 color "red" set n3
[$ns node]
$n3 color "red" set n4
[$ns node]
$n4 color "magenta" set n5
[$ns node]
$n5 color "magenta" set n6
[$ns node]
$n6 color "magenta" set n7
[$ns node]
$n7 color "magenta"
$ns make-lan "$n0 $n1 $n2 $n3" 100Mb 300ms LL Queue/DropTail Mac/802_3
$ns make-lan "$n4 $n5 $n6 $n7" 100Mb 300ms LL Queue/DropTail Mac/802_3
$ns duplex-link $n3 $n4 100Mb 300ms DropTail
$ns duplex-link-op $n3 $n4 color "green"
set err [new ErrorModel]
$ns lossmodel $err $n3 $n4
$err set rate_ 0.1
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set fid_ 0
$cbr set packetSize_ 1000
$cbr set interval_ 0.0001 set null
[new Agent/Null]
$ns attach-agent $n7 $null
$ns connect $udp $null proc
finish {} {
```

```
global ns nf tf
$ns flush-trace
close $nf  close $tf
exec nam lab8.nam & exit 0
}
$ns at 0.1 "$cbr start"
$ns at 3.0 "finish"
$ns run
Awk File BEGIN{
pkt=0; time=0;
}
{ if($1=="r"&& $3=="9"&& $4=="7"){
pkt=pkt+$6;
time=$2;
}
}
END{
printf("Throughput=%fMbps",((pkt/time)*(8/1000000)));
}
```



10. Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

```
set ns [new Simulator] set tf
[open p9.tr w]
$ns trace-all $tf
set nf [open p9.nam w]
$ns namtrace-all $nf set
n0 [$ns node]
$n0 color "magenta"
$n0 label "src1" set
n1 [$ns node]
$n1 color "red" set
n2 [$ns node]
$n2 color "magenta"
$n2 label "src2" set
n3 [$ns node]
$n3 color "blue"
$n3 label "dest2" set
n4 [$ns node]
$n4 shape square set
n5 [$ns node]
$n5 color "blue"
$n5 label "dest1"
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 50Mb 100ms LL Queue/DropTail Mac/802_3
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
$ns duplex-link-op $n4 $n5 orient right
$ns duplex-link-op $n4 $n5 color green set tcp0
[new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
set sink0 [new Agent/TCPSink]
$ns attach-agent $n5 $sink0
$ns connect $tcp0 $sink0 set
tcp1 [new Agent/TCP]
$ns attach-agent $n2 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set packetSize_ 600
$ftp1 set interval_ 0.001
set sink1 [new Agent/TCPSink]
```

```
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1 set
file1 [open file1.tr w]
$tcp0 attach $file1
set file2 [open file2.tr w]
$tcp1 attach $file2
$tcp0 trace cwnd_
$tcp1 trace cwnd_
proc finish {} {
global ns nf tf
$ns flush-trace
close $tf close
$nf
  exec nam p9.nam & exit 0
}
$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp1 start"
$ns at 8 "$ftp1 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp1 start"
$ns at 15 "$ftp1 stop"

$ns at 16 "finish"
$ns run
```

Awk file:

```
BEGIN {
}
{ if($6=="cwnd_")
printf("%f\t%f\t\n",$1,$7);
}  END
{
}
```