

Laboratory Component:

1. **Implement three nodes point – to – point network with duplex links between them for different topologies. Set the queue size, vary the bandwidth, and find the number of packets dropped for various iterations.**
2. **Implement simple ESS and with transmitting nodes in wireless LAN by simulation and determine the throughput with respect to transmission of packets.**
3. Write a program for error detecting code using CRC-CCITT (16- bits).
4. **Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion in the network.**
5. Write a program to find the shortest path between vertices using bellman-ford algorithm.
6. **Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.**
7. Write a program for congestion control using leaky bucket algorithm.

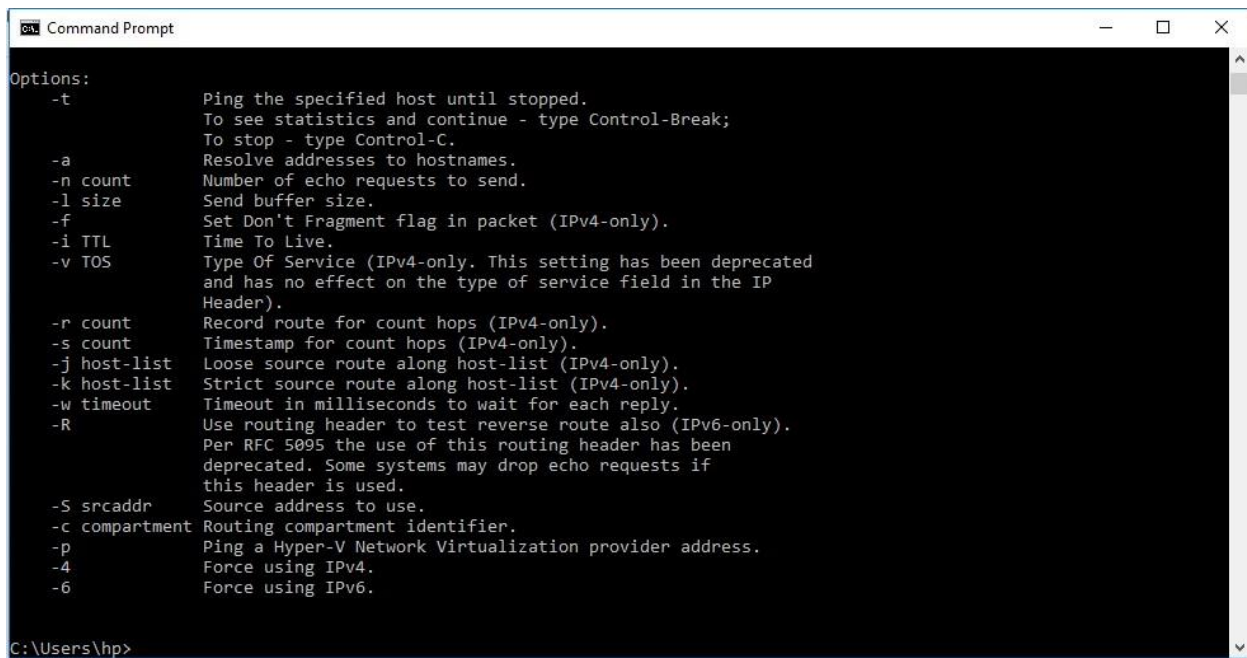
Introduction

Network commands are like a superhero toolkit for anyone who manages computer networks. These commands are like magic spells that help you set up, fix problems, figure out what's wrong, and take care of all the connections in a Linux computer system. Some important network commands have been given below,

- ‡ Ping
- ‡ Tracert
- ‡ Ipconfig
- ‡ Arp
- ‡ Netstat

1. Ping

This command tests the connection between the local machine and the host server. This command sends a small amount of data to the host server, and the host server replies to the computer.



```
Options:
-t          Ping the specified host until stopped.
            To see statistics and continue - type Control-Break;
            To stop - type Control-C.
-a          Resolve addresses to hostnames.
-n count    Number of echo requests to send.
-l size     Send buffer size.
-f          Set Don't Fragment flag in packet (IPv4-only).
-i TTL      Time To Live.
-v TOS      Type Of Service (IPv4-only. This setting has been deprecated
            and has no effect on the type of service field in the IP
            Header).
-r count    Record route for count hops (IPv4-only).
-s count    Timestamp for count hops (IPv4-only).
-j host-list Loose source route along host-list (IPv4-only).
-k host-list Strict source route along host-list (IPv4-only).
-w timeout  Timeout in milliseconds to wait for each reply.
-R          Use routing header to test reverse route also (IPv6-only).
            Per RFC 5095 the use of this routing header has been
            deprecated. Some systems may drop echo requests if
            this header is used.
-S srcaddr  Source address to use.
-c compartment Routing compartment identifier.
-p          Ping a Hyper-V Network Virtualization provider address.
-4          Force using IPv4.
-6          Force using IPv6.

C:\Users\hp>
```

2. Ping -a

The "-a" option resolves the hostname to the respective IP address.

Eg: Ping -a www.google.com

3. Ping -w timeout

The option "-w timeout" sets the timeout, the time after which the data packet will be rejected for each ping. The timeout is in milliseconds. Eg: ping -w 20 www.google.com

```
Command Prompt

C:\Users\hp>ping google.com

Pinging google.com [142.250.193.110] with 32 bytes of data:
Reply from 142.250.193.110: bytes=32 time=12ms TTL=117
Reply from 142.250.193.110: bytes=32 time=11ms TTL=117
Reply from 142.250.193.110: bytes=32 time=11ms TTL=117
Reply from 142.250.193.110: bytes=32 time=11ms TTL=117

Ping statistics for 142.250.193.110:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 11ms, Maximum = 12ms, Average = 11ms

C:\Users\hp>ping -a google.com

Pinging google.com [142.250.193.110] with 32 bytes of data:
Reply from 142.250.193.110: bytes=32 time=11ms TTL=117
Reply from 142.250.193.110: bytes=32 time=11ms TTL=117
Reply from 142.250.193.110: bytes=32 time=22ms TTL=117
Reply from 142.250.193.110: bytes=32 time=11ms TTL=117

Ping statistics for 142.250.193.110:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 11ms, Maximum = 22ms, Average = 13ms

C:\Users\hp>
```

4. Tracert

The tracert command traces the route from a computer to a host server. It traces the connection for a fixed maximum number of hops. It is used to diagnose path-related problems. Eg: tracert www.google.com

```
Command Prompt

Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\hp>tracert www.amazon.com

Tracing route to d3ag4hukkh62yn.cloudfront.net [13.33.144.35]
over a maximum of 30 hops:

  0  <1 ms    <1 ms    <1 ms    192.168.1.1
  1  2 ms     2 ms     1 ms     ip-lan1.local [130.1.1.2]
  2  7 ms     8 ms     6 ms     125.21.204.65
  3  *        *        *        Request timed out.
  4  *        *        *        Request timed out.
  5  11 ms    10 ms    11 ms    52.93.19.93
  6  11 ms    16 ms    14 ms    52.93.19.108
  7  17 ms    16 ms    10 ms    150.222.14.59
  8  *        *        *        Request timed out.
  9  *        *        *        Request timed out.
 10  *        *        *        Request timed out.
 11  *        *        *        Request timed out.
 12  *        *        *        Request timed out.
 13  *        *        *        Request timed out.
 14  9 ms     9 ms     9 ms     server-13-33-144-35.maa50.r.cloudfront.net [13.33.144.35]

Trace complete.

C:\Users\hp>
```

5. Tracert/?

Eg: tracert/?

```
Command Prompt
C:\Users\hp>tracert/?

Usage: tracert [-d] [-h maximum_hops] [-j host-list] [-w timeout]
              [-R] [-S srcaddr] [-4] [-6] target_name

Options:
  -d                Do not resolve addresses to hostnames.
  -h maximum_hops  Maximum number of hops to search for target.
  -j host-list      Loose source route along host-list (IPv4-only).
  -w timeout        Wait timeout milliseconds for each reply.
  -R                Trace round-trip path (IPv6-only).
  -S srcaddr        Source address to use (IPv6-only).
  -4                Force using IPv4.
  -6                Force using IPv6.

C:\Users\hp>
```

6. -d

The "-d" option with the tracert command instructs TRACERT not to perform a DNS lookup on each IP address.

Eg: tracert -d www.google.com

7. -h maximum hops

The "-h maximum hops" option sets the maximum number of hops for which the tracert command will trace the connection. Eg: tracert -h 20 www.cambridge.edu.in

```
Command Prompt
C:\Users\hp>tracert -h 20 www.cambridge.edu.in

Tracing route to www.cambridge.edu.in [15.207.194.161]
over a maximum of 20 hops:

  1  <1 ms    <1 ms    <1 ms    192.168.1.1
  2  1 ms     2 ms     <1 ms    ip-lan1.local [130.1.1.2]
  3  44 ms    3 ms     4 ms    125.21.204.65
  4  *         *         *        Request timed out.
  5  *         *         *        Request timed out.
  6  *         *         *        Request timed out.
  7  32 ms    31 ms    30 ms    52.95.64.170
  8  32 ms    32 ms    35 ms    52.95.64.169
  9  37 ms    31 ms    31 ms    99.83.76.129
 10  35 ms    31 ms    32 ms    99.83.76.140
 11  *         *         *        Request timed out.
 12  *         *         *        Request timed out.
 13  *         *         *        Request timed out.
 14  *         *         *        Request timed out.
 15  *         *         *        Request timed out.
 16  *         *         *        Request timed out.
 17  *         *         *        Request timed out.
 18  *         *         *        Request timed out.
 19  *         *         *        Request timed out.
 20  *         *         *        Request timed out.

Trace complete.
```

8. Ipconfig

As the command name suggests, it gives information about the IP address. It not only gives the IP address of the computer it is executed on but also much more information as DNS addresses are stored in the cache.

Eg: ipconfig

```
Command Prompt

C:\Users\hp>ipconfig

Windows IP Configuration

Wireless LAN adapter Wi-Fi:

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

Ethernet adapter Ethernet:

    Connection-specific DNS Suffix  . :
    Link-local IPv6 Address . . . . . : fe80::dd2:2a9e:2bf9:c904%7
    IPv4 Address. . . . . : 192.168.1.193
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.1

Tunnel adapter Local Area Connection* 11:

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

Tunnel adapter Reusable ISATAP Interface {7C1206EF-5642-4281-BDCB-89589678501A}:

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

9. Ipconfig/? Help to know attributes Eg: ipconfig/?

```
Select Command Prompt

C:\Users\hp>ipconfig /registerdns
The requested operation requires elevation.

C:\Users\hp>ipconfig/?

USAGE:
    ipconfig [/allcompartments] [/? | /all |
        /renew [adapter] | /release [adapter] |
        /renew6 [adapter] | /release6 [adapter] |
        /flushdns | /displaydns | /registerdns |
        /showclassid adapter |
        /setclassid adapter [classid] |
        /showclassid6 adapter |
        /setclassid6 adapter [classid] ]

where
    adapter
        Connection name
        (wildcard characters * and ? allowed, see examples)

Options:
    /?          Display this help message
    /all        Display full configuration information.
    /release    Release the IPv4 address for the specified adapter.
    /release6   Release the IPv6 address for the specified adapter.
    /renew      Renew the IPv4 address for the specified adapter.
    /renew6     Renew the IPv6 address for the specified adapter.
    /flushdns   Purges the DNS Resolver cache.
    /registerdns Refreshes all DHCP leases and re-registers DNS names
    /displaydns Display the contents of the DNS Resolver Cache.
    /showclassid Displays all the dhcp class IDs allowed for adapter.
    /setclassid Modifies the dhcp class id.
    /showclassid6 Displays all the IPv6 DHCP class IDs allowed for adapter.
    /setclassid6 Modifies the IPv6 DHCP class id.

The default is to display only the IP address, subnet mask and
Default gateway for each adapter bound to TCP/IP.

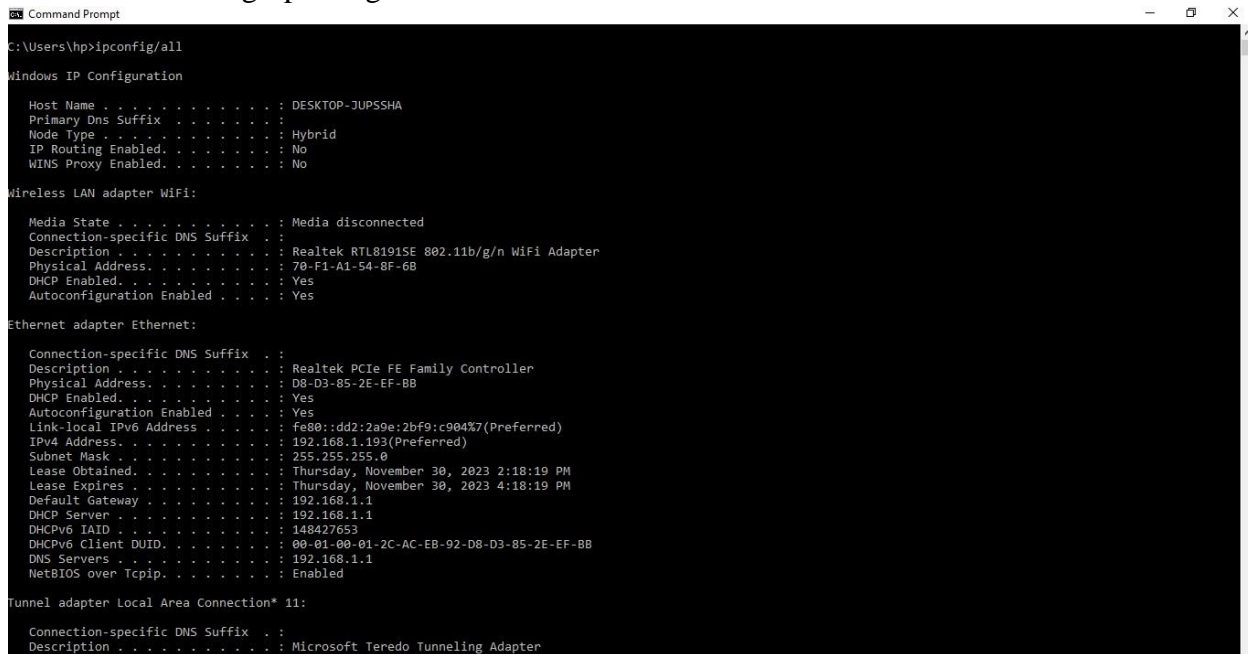
For Release and Renew, if no adapter name is specified, then the IP address
leases for all adapters bound to TCP/IP will be released or renewed.

For Setclassid and Setclassid6, if no ClassId is specified, then the ClassId is removed.

Examples:
    > ipconfig           ... Show information
    > ipconfig /all       ... Show detailed information
    > ipconfig /renew     ... renew all adapters
    > ipconfig /renew EL* ... renew any connection that has its
                        ... name starting with EL
    > ipconfig /release *Con* ... release all matching connections,
                        ... eg. "Wired Ethernet Connection 1" or
                        ... "Wired Ethernet Connection 2"
    > ipconfig /allcompartments ... Show information about all
                        ... compartments
    > ipconfig /allcompartments /all ... Show detailed information about all
                        ... compartments
```

10. /all

The "/all" option of the ipconfig command displays the full configuration information. Eg: ipconfig/all



```
Command Prompt
C:\Users\hp>ipconfig/all

Windows IP Configuration

Host Name . . . . . : DESKTOP-JUPSSHA
Primary Dns Suffix . . . . . :
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No

Wireless LAN adapter WiFi:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :
Description . . . . . : Realtek RTL8191SE 802.11b/g/n WiFi Adapter
Physical Address. . . . . : 70-F1-A1-54-8F-6B
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes

Ethernet adapter Ethernet:

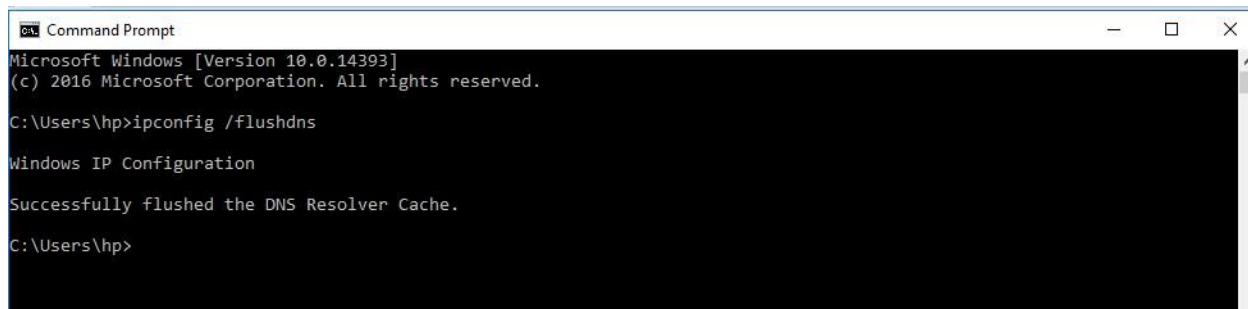
Connection-specific DNS Suffix . :
Description . . . . . : Realtek PCIe FE Family Controller
Physical Address. . . . . : D8-D3-85-2E-EF-BB
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes
Link-Local IPv6 Address . . . . . : fe80::dd2:2a9e:2bf9:c904%7(Preferred)
IPv4 Address. . . . . : 192.168.1.193(Preferred)
Subnet Mask . . . . . : 255.255.255.0
Lease Obtained. . . . . : Thursday, November 30, 2023 2:18:19 PM
Lease Expires . . . . . : Thursday, November 30, 2023 4:18:19 PM
Default Gateway . . . . . : 192.168.1.1
DHCP Server . . . . . : 192.168.1.1
DHCPv6 IAID . . . . . : 148427653
DHCPv6 Client DUID. . . . . : 00-01-00-01-2C-AC-EB-92-D8-D3-85-2E-EF-BB
DNS Servers . . . . . : 192.168.1.1
NetBIOS over Tcpip. . . . . : Enabled

Tunnel adapter Local Area Connection* 11:

Connection-specific DNS Suffix . :
Description . . . . . : Microsoft Teredo Tunneling Adapter
```

11. /flushdns

The "/flushdns" option clears the DNS table stored in the cache of the local machine Eg: ipconfig /flushdns



```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\hp>ipconfig /flushdns

Windows IP Configuration

Successfully flushed the DNS Resolver Cache.

C:\Users\hp>
```

12. /registerdns

The "/registerdns" option refreshes all DHCP leases and re-registers the DNS names in the local machine's cache. Eg: ipconfig /registerdns

13. Arp

The arp command is a short form for Address Resolution Protocol. This command is used to display and modify the IP to the physical address translation table used by the address resolution protocol.

Eg: arp

14. Arp/? Help to get arp attributes Eg: arp/?

15. Arp -a

The "-a" command in arp displays current ARP entries by interrogating the current protocol data. If inet_addr is specified, the IP and physical addresses for only the specified computer are displayed. Eg: arp -a

```
Command Prompt
C:\Users\hp>arp -a

Interface: 192.168.1.193 --- 0x7
Internet Address      Physical Address      Type
192.168.1.1           28-87-ba-01-c6-23     dynamic
192.168.1.145         dc-e9-94-32-a1-81     dynamic
192.168.1.255         ff-ff-ff-ff-ff-ff     static
224.0.0.2             01-00-5e-00-00-02     static
224.0.0.22            01-00-5e-00-00-16     static
224.0.0.251           01-00-5e-00-00-fb     static
224.0.0.252           01-00-5e-00-00-fc     static
224.0.0.253           01-00-5e-00-00-fd     static
239.255.255.250       01-00-5e-7f-ff-fa     static
255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

16. Netstat

This command displays the connections active on the computer and the ports the computer is listening to. The command displays the four parameters: proto, local address, foreign address, and state. Eg: netstat

```
Command Prompt
C:\Users\hp>netstat

Active Connections

Proto Local Address           Foreign Address         State
TCP   192.168.1.193:50135      sb-in-f188:5228        ESTABLISHED
TCP   192.168.1.193:50252      whatsapp-cdn-shv-01-maa2:https ESTABLISHED
TCP   192.168.1.193:50419      134:https               ESTABLISHED
TCP   192.168.1.193:50429      a23-46-207-114:https     ESTABLISHED
TCP   192.168.1.193:50543      a23-46-207-136:https     CLOSE_WAIT
TCP   192.168.1.193:50984      162.247.243.29:https     ESTABLISHED
TCP   192.168.1.193:51029      server-52-84-12-106:https ESTABLISHED
TCP   192.168.1.193:51030      maa05s21-in-f14:https   TIME_WAIT
TCP   192.168.1.193:51041      maa05s22-in-f14:https   ESTABLISHED
TCP   192.168.1.193:51042      sg-in-f155:https        ESTABLISHED
TCP   192.168.1.193:51043      maa05s22-in-f4:https    ESTABLISHED
TCP   192.168.1.193:51044      bom05s11-in-f3:https    ESTABLISHED
TCP   192.168.1.193:51045      20.198.118.190:https     ESTABLISHED
TCP   192.168.1.193:51047      20.198.118.190:https     ESTABLISHED
TCP   192.168.1.193:51048      20.189.173.6:https       TIME_WAIT
TCP   192.168.1.193:51051      20.198.118.190:https     ESTABLISHED
TCP   192.168.1.193:51052      20.198.118.190:https     ESTABLISHED
TCP   192.168.1.193:51055      server-108-159-15-74:https ESTABLISHED
TCP   192.168.1.193:51059      maa05s18-in-f3:https    ESTABLISHED
TCP   192.168.1.193:51060      ec2-52-2-3-136:https     ESTABLISHED

C:\Users\hp>
```

17. Netstat/? Help to get the attributes

```
Command Prompt
C:\Users\hp>netstat/?
Displays protocol statistics and current TCP/IP network connections.

NETSTAT [-a] [-b] [-e] [-f] [-n] [-o] [-p proto] [-r] [-s] [-x] [-t] [interval]

-a      Displays all connections and listening ports.
-b      Displays the executable involved in creating each connection or
        listening port. In some cases well-known executables host
        multiple independent components, and in these cases the
        sequence of components involved in creating the connection
        or listening port is displayed. In this case the executable
        name is in [] at the bottom, on top is the component it called,
        and so forth until TCP/IP was reached. Note that this option
        can be time-consuming and will fail unless you have sufficient
        permissions.
-e      Displays Ethernet statistics. This may be combined with the -s
        option.
-f      Displays Fully Qualified Domain Names (FQDN) for foreign
        addresses.
-n      Displays addresses and port numbers in numerical form.
-o      Displays the owning process ID associated with each connection.
-p proto Shows connections for the protocol specified by proto; proto
        may be any of: TCP, UDP, TCPv6, or UDPv6. If used with the -s
        option to display per-protocol statistics, proto may be any of:
        IP, IPv6, ICMP, ICMPv6, TCP, TCPv6, UDP, or UDPv6.
-q      Displays all connections, listening ports, and bound
        nonlistening TCP ports. Bound nonlistening ports may or may not
        be associated with an active connection.
-r      Displays the routing table.
-s      Displays per-protocol statistics. By default, statistics are
        shown for IP, IPv6, ICMP, ICMPv6, TCP, TCPv6, UDP, and UDPv6;
        the -p option may be used to specify a subset of the default.
-t      Displays the current connection offload state.
-x      Displays NetworkDirect connections, listeners, and shared
        endpoints.
-y      Displays the TCP connection template for all connections.
        Cannot be combined with the other options.
interval Redisplay selected statistics, pausing interval seconds
        between each display. Press CTRL+C to stop redisplaying
        statistics. If omitted, netstat will print the current
        configuration information once.
```

18. -a

The "-a" option of the netstat command displays all connections and listening ports.

```
Command Prompt - netstat -a
C:\Users\hp>netstat -a
Active Connections

Proto Local Address           Foreign Address         State
TCP    0.0.0.0:135              DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:445              DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:554              DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:2869             DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:5357             DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:10243            DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:49664            DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:49665            DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:49666            DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:49667            DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:49670            DESKTOP-JUPSSHA:0      LISTENING
TCP    0.0.0.0:49678            DESKTOP-JUPSSHA:0      LISTENING
TCP    192.168.1.193:139        DESKTOP-JUPSSHA:0      LISTENING
TCP    192.168.1.193:50135      sb-in-f188:5228        ESTABLISHED
TCP    192.168.1.193:50252      whatsapp-cdn-shv-01-maa2:https ESTABLISHED
TCP    192.168.1.193:50419      134:https               ESTABLISHED
TCP    192.168.1.193:50429      a23-46-207-114:https     ESTABLISHED
TCP    192.168.1.193:50543      a23-46-207-136:https     CLOSE_WAIT
TCP    192.168.1.193:50984      162.247.243.29:https     ESTABLISHED
TCP    192.168.1.193:51029      server-52-84-12-106:https ESTABLISHED
TCP    192.168.1.193:51041      maa05s22-in-f14:https    ESTABLISHED
TCP    192.168.1.193:51042      sg-in-f155:https         ESTABLISHED
TCP    192.168.1.193:51043      maa05s22-in-f4:https     ESTABLISHED
TCP    192.168.1.193:51044      bom05s11-in-f3:https     ESTABLISHED
```


Network simulation

Network simulation is a technique used to model the behavior of computer networks and simulate their performance under various conditions. It involves creating a virtual representation of a network using software to analyze and predict its functionality. Here's a summary of key points related to network simulation:

Purpose: Network simulation is employed for various purposes, including testing, analysis, design, and optimization of computer networks. It allows researchers, engineers, and network administrators to evaluate network protocols, configurations, and performance without the need for physical implementation.

Simulation Tools: Numerous simulation tools are available, ranging from open-source options like NS-2 (Network Simulator 2), NS-3, GNS3 (Graphical Network Simulator), and Mininet, to commercial tools like Cisco Packet Tracer and OPNET. These tools provide a platform to create, configure, and analyze network scenarios.

Components of a Network Simulation:

Nodes: Represent devices such as computers, routers, and switches.

Links: Simulate the connections between nodes, representing communication channels.

Protocols: Network simulation involves modeling various networking protocols and their interactions.

Types of Simulations: **Packet-Level Simulations:** Model individual data packets and their movement through the network.

Event-Driven Simulations: Triggered by specific events, such as a change in network topology or a failure.

Time-Driven Simulations: Progress based on predefined time intervals.

Applications:

Protocol Testing: Evaluate the performance and reliability of network protocols.

Network Design and Optimization: Test different network configurations to identify the most efficient design.

Education and Training: Provide a practical learning environment for networking students.

Troubleshooting: Simulate network issues to develop and test troubleshooting skills.

Network simulators

Network simulators are software tools that replicate the behavior of computer networks, allowing users to model, analyze, and test various network scenarios in a virtual environment. These simulators provide a platform for researchers, engineers, and network administrators to experiment with different network configurations, protocols, and scenarios without the need for physical hardware.

Here are key characteristics and functionalities of network simulators:

Virtual Representation: Network simulators create a virtual representation of a computer network, including nodes (devices like computers, routers, and switches), links (communication channels between nodes), and protocols governing communication.

Topology Design: Users can design and configure network topologies by defining the arrangement and connectivity of nodes and links. This flexibility is crucial for testing different network architectures.

Protocol Simulation: Simulators model various networking protocols, allowing users to observe how these protocols interact and perform under different conditions. This is especially useful for testing and refining network protocols.

Traffic Generation: Users can simulate different types and volumes of network traffic to assess how the network handles varying loads. This helps in evaluating performance, identifying bottlenecks, and optimizing configurations.

Scalability: Network simulators support the scalability of experiments. Users can easily scale the size and complexity of the simulated network to assess performance under various conditions.

Visualization Tools: Many network simulators come with visualization tools that help users observe and analyze the behavior of the simulated network. This visual representation aids in identifying issues and understanding the impact of changes.

Educational Use: Network simulators are widely used in educational settings to provide students with hands-on experience in networking concepts and protocols. They offer a safe environment for learning without affecting live networks.

Popular Network Simulators:

NS-2 (Network Simulator 2): NS-3 (Network Simulator 3): An open-source network simulator widely used for research and educational purposes. GNS3 (Graphical Network Simulator): Primarily used for emulating and simulating network devices for Cisco's networking certification exams. Cisco Packet Tracer: Cisco's simulation tool designed for learning and practicing networking concepts. Mininet: An open-source network emulator used for creating virtual networks for SDN (Software-Defined Networking) testing.

Software required

The software required for network simulators depends on the specific simulator you choose.
Network Simulator Software:

NS-2 (Network Simulator 2): NS2 is written in C++ and requires a compatible C++ compiler (**GNU Compiler Collection (GCC)**) or java programming. It supports Linux Operating Systems and is typically run from the command line.

Tcl Programming/commands

In network simulation, specifically in the context of network simulators like NS-2 (Network Simulator 2) or NS-3 (Network Simulator 3), Tcl (Tool Command Language) is often used for defining and configuring network scenarios. Tcl scripts are commonly used to set up simulation parameters, create network topologies, and define various simulation events. Below are some basic Tcl commands and examples for network simulation:

To start with type **tclsh** in the windows start bar. You will get like windows **cmd** prompt with **%** only.

Few basic commands

- ❖ **cls** – clear screen
- ❖ **exit** – exit from the tcl shell
- ❖ **set** - Read and write variables
- ❖ **puts** – Write to a channel
- ❖ **proc** - Create a Tcl procedure
- ❖ **expr** - Evaluate an expression

puts command

- ❖ you can print something to the console using the puts/echo command:

1	%	puts Cambridge
2		Cambridge
3	%	echo Cambridge
4		Cambridge
5	%	puts "Hello, World!"
6		Hello, World!
7	%	puts "Hello, World!"; puts "Welcome to Cambridge"
8		Hello, World! Welcome to Cambridge

set command

❖ For numbers representation:

1	%	set x 1.2 (floating point)
2		1.2
3	%	set y 100 (integer)
4		100
5	%	set num1 50
6		50
7	%	set num2 100
8		100

Set and puts command for variable assign

❖ you can set the variable:

1	%	set variable_name “Raja, King”
2		Raja, King
3	%	puts \$variable_name
4		Raja, King
5	%	set cit “Cambrian Bangalore”
6		Cambrian Bangalore
7	%	puts \$cit
8		Cambrian Bangalore

Math Operations:

Perform basic math operations using expressions – with **expr**

For example, to add/sub/multiply/divide two numbers:

1	%	expr 25 + 25
2		50
3	%	expr 25 – 25
4		0
5	%	expr 25 / 25
6		1
7	%	expr 25 * 25
8		125

Sample Coding

Creating Nodes and Links:

```
# Creating
nodes set node0
[$ns node] set
node1 [$ns
node]
```

```
# Creating a link between nodes
set link0 [$ns duplex-link $node0 $node1 10Mb 5ms DropTail]
```

Defining Traffic

```
Sources: # Creating a
traffic source set udp0
[new Agent/UDP]
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$ns attach-agent $node0 $udp0
```

```
# Setting traffic parameters
```

```
$cbr0 set rate 1Mbps
$cbr0 set packetSize 100
```

Configuring Routing:

```
# Creating a routing agent (e.g.,
AODV) set agent0 [new
Agent/AODV] $ns attach-agent
$node0 $agent0
```

```
# Creating a routing agent (e.g.,
AODV) set agent1 [new
Agent/AODV] $ns attach-agent
$node1 $agent1
```

```
# Creating a routing object
set router0 [new AODV-LL $node0 $agent0]
```

Defining Simulation Time and Events:

Setting simulation time

```
$ns at 5.0 "$cbr0 start"
```

```
# Ending simulation at 10 seconds
$ns at 10.0 "$ns halt"
```

Running NS-2

Running NS-2 (Network Simulator 2) involves several steps, including creating a simulation script in Tcl, compiling it, and executing the simulation. Here's a general guide on how to run NS-2:

Step 1: Install NS-2

Ensure that NS-2 is installed on your system. You can follow the installation instructions provided by the NS-2 documentation or use package managers like apt (for Linux) or Homebrew (for macOS).

Step 2: Create a Tcl Script

Write a Tcl script that describes your network topology, nodes, links, traffic sources, and simulation events. Save the script with a .tcl extension. Here's a simple example:

```
# Sample NS-2 Tcl script
set ns [new Simulator]
```

```
# Create nodes
set node0 [$ns
```



```
node] set node1
[$ns node]

# Create a link between nodes
set link0 [$ns duplex-link $node0 $node1 10Mb 5ms DropTail]

# Set simulation time
$ns at 5.0 "$node0 start-udp"

# End simulation at 10 seconds
$ns at 10.0 "$ns halt"

# Run simulation
$ns run
```

Step 3: Compile the Tcl Script

To compile the Tcl script, use the following command in the terminal:
ns your_script_name.tcl

Step 4: Analyze Results

Once the simulation completes, NS-2 generates trace files and outputs relevant information. You can analyze these files using tools like nam (Network Animator) and **tracegraph** to visualize the simulation results. `nam your_script_name.nam`
`tracegraph your_script_name.tr`

Viewing Animation (Optional): If you want to visualize the network animation during simulation, you can use **nam**. Open a new terminal window and run:
`nam your_script_name.nam`

Adjust Simulation Parameters: Modify the Tcl script based on your specific simulation requirements, such as the network topology, traffic, and simulation duration.

Explore NS-2 Documentation: NS-2 has extensive documentation that provides details on various components, commands, and options. Refer to the documentation for advanced features and customization.

Ex No.1: Implement three nodes point – to – point network with duplex links between them for different topologies. Set the queue size, vary the bandwidth, and find the number of packets dropped for various iterations.

Aim: To implement three nodes point – to – point network with duplex links between them for different topologies. Set the queue size, vary the bandwidth, and find the number of packets dropped for various iterations.

Algorithm for NS-2 with Tcl scripting

Step 1. Initialize Simulation:

- Create a new NS-2 simulator instance
- Set simulation parameters (e.g., simulation time, node positions)

Step 2. Create Nodes and Links:

- Create three nodes: Node A, Node B, Node C
- Create duplex links between nodes (e.g., A to B, B to C)

Step 3. Set Queue Size and Bandwidth:

- Loop through various iterations:
- Vary the queue size and bandwidth parameters
- Set queue size for each link
- Set bandwidth for each link

Step 4. Traffic Generation:

- Create traffic sources (e.g., TCP or UDP applications)
- Attach traffic sources to nodes
- Set traffic parameters (e.g., packet size, traffic rate)

Step 5. Run Simulation:

- Run the NS-2 simulation

Step 6. Record Results:

- Capture and analyze simulation results, such as trace files
- Extract the number of packets dropped for each iteration

Step 7. Display or Save Results:

- Display or save the results, including the number of packets dropped for each iteration
- Optionally, visualize the network behavior using tools like nam (Network Animator)

Step 8. Repeat for Different Topologies:

- Optionally, repeat the entire process for different network topologies by modifying link connections or positions

Step 9. End Simulation:

- End the NS-2 simulation

Step 10. Analyze Overall Performance:

- Analyze the overall performance based on the collected results
- Evaluate the impact of varying queue size and bandwidth on packet drops

Step 11. End

Sample Coding

TCL:

```
#Create a simulator object
set ns [ new Simulator ]
```

```
#Open the nam trace
file set tf [ open lab1.tr
w ]
$ns trace-all $tf
```

```
#Open the nam trace
file set nf [ open
lab1.nam w ]
$ns namtrace-all $nf
```

```
#Define a 'finish'
procedure proc finish
{ } { global ns nf tf
$ns flush-trace exec
nam lab1.nam & close
$tf close $nf exit 0 }
```

```
#Creating
nodes set n0
[$ns node] set
n1 [$ns node]
set n2 [$ns
node] set n3
[$ns node]
```

```
#Define different colors and labels for data flows
$ns color 1 "red"
$ns color 2 "blue"
```

```
$n0 label "Source/udp0"
$n1 label "Source/udp1"
$n2 label "Router"
$n3 label "Destination/Null"

#Create link between nodes
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n3 1Mb 300ms DropTail

#Set queue size of links
$ns set queue-limit $n0 $n2 50
$ns set queue-limit $n1 $n2 50
$ns set queue-limit $n2 $n3 5

#Setup a UDP connection
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 UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$udp1 set class_ 2
$ns attach-agent $n1 $udp1

# 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 $udp1

#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 sources with the traffic sink
$ns connect $udp0 $null0
$ns connect $udp1 $null0
```

```
#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:
```

```
BEGIN{
```

```
count=0;
```

```
}
```

```
{
```

```
if($1=="d
```

```
")
```

```
count++
```

```
}
```

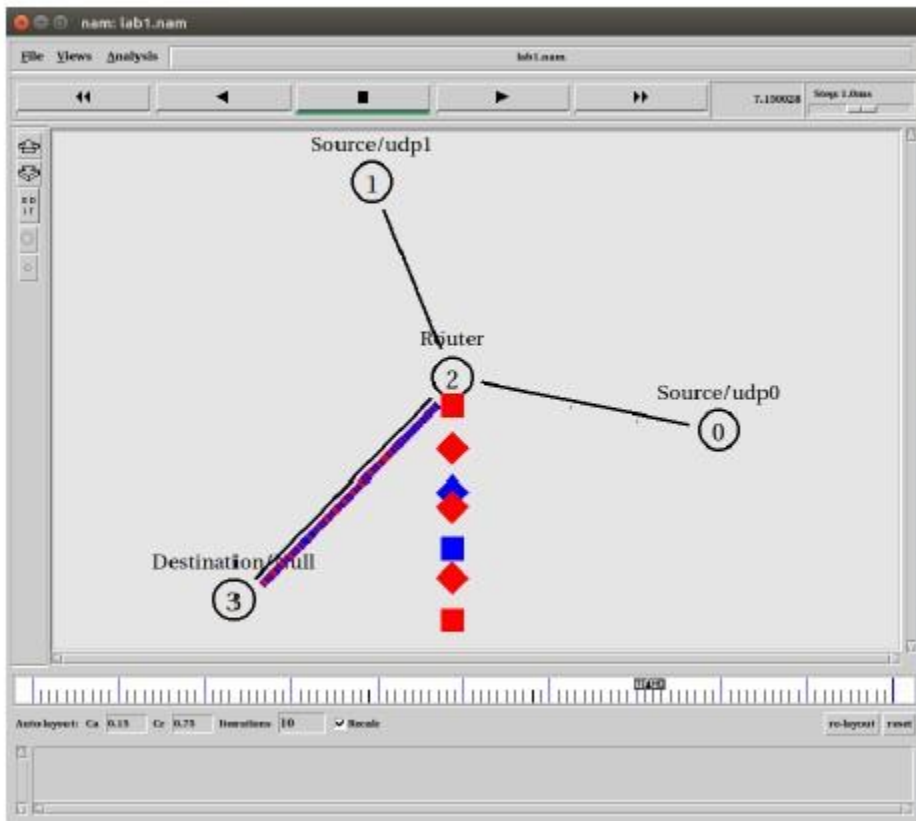
```
END{
```

```
printf("The Total no of Packets Drop is :%d\n\n",
```

```
count) }
```

Sample Output

Simulated Network:



Packet Dropped:

```
krishna@ubuntu:~$ awk -f lab1.awk lab1.tr
The Total no of Packets Drop is :708
krishna@ubuntu:~$
```

OR

Using grep command: `cat lab1.tr | grep ^d | wc -l`

Result:

Step by Step Instruction Explanation

Ex No.1: Creating a three-node point-to-point network with duplex links, setting the queue size, varying bandwidth, and analyzing packet drops using NS2 involves several steps.

NS2 (Network Simulator 2) is a popular discrete event simulator for computer networks.

Step 1: Create a topology file

Open a text editor (e.g., gedit, vim, or notepad) and create a file named `my_topology.tcl`.

Copy and paste the following

```
code: # Create a simulator
instance
set ns [new Simulator]
```

```
# Create three
nodes set n0
[$ns node] set
n1 [$ns node]
set n2 [$ns
node]
```

```
# Create duplex links between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 20ms DropTail
```

```
# Set queue size
$ns queue-limit $n0 $n1 5
$ns queue-limit $n1 $n2 10
```

```
# Set up traffic between
nodes set tcp [new
Agent/TCP] $ns attach-
agent $n0 $tcp
```

```
set sink [new Agent/TCPSink]
$ns attach-agent $n2 $sink
```

```
$ns connect $tcp $sink
```

```
# Create a traffic source and attach it to node n0
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.1 "$ftp start"
```

```
# Define simulation time
$ns at 5.0 "$ns stop"
```

```
# Run the simulation
$ns run
```

Step 2: Save the file and close the text editor.

Step 3: Run the simulation:

Open a terminal and navigate to the directory where **my_topology.tcl** is located. Type the following command to run the simulation: ns my_topology.tcl

Step 4: Analyze the results:

After the simulation completes, you will see output files such as **my_topology.nam** and **my_topology.tr**.

To visualize the network topology, use Nam:
nam my_topology.nam

To analyze the packet drop, you can check the **my_topology.tr** file.

Step 5: Modify and Iterate:

You can modify the topology, bandwidth, queue size, and other parameters in the my_topology.tcl file to perform various iterations and experiments.

Ex No.2: Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion

Aim: To Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion

Algorithm for NS-2 with Tcl scripting

Step 1: Install NS-2

Step 2: Create NS-2 Simulation Script (ping_trace_simulation.tcl)

Step 3: Create AWK Script (analyze.awk)

Step 4: Run the Simulation and Analyze Dropped Packets

Step 5: Analysis the network transmission using awk

Sample Coding

```
set ns [new simulator]
set tf [open lab2.tr w]
$ns trace-all $tf set nf
[open lab2.nam w] $ns
namtrace-all $nf set n0
[$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]
$n0 label "Ping0"
$n4 label "Ping4"
$n5 label "Ping5"
$n6 label "Ping6"
$n2 label "Router"

$ns color 1 "red"
$ns color 2 "green"

$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 1Mb 300ms DropTail
$ns duplex-link $n3 $n2 1Mb 300ms DropTail

$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4 1Mb 300ms DropTail
```

```
$ns duplex-link $n2 $n6 1Mb 300ms DropTail
```

```
$ns queue-limit $n0 $n2 5
```

```
$ns queue-limit $n2 $n4 3
```

```
$ns queue-limit $n2 $n6 2
```

```
$ns queue-limit $n5 $n2 5
```

```
set ping0 [new Agent/Ping]
```

```
$ns attach-agent $n0 $ping0
```

```
set ping4 [new Agent/Ping]
```

```
$ns attach-agent $n4 $ping4
```

```
set ping5 [new Agent/Ping]
```

```
$ns attach-agent $n5 $ping5
```

```
set ping6 [new Agent/Ping]
```

```
$ns attach-agent $n6 $ping6
```

```
$ping0 set packetSize_ 50000
```

```
$ping0 set interval_ 0.0001
```

```
$ping5 set packetSize_ 60000
```

```
$ping5 set interval_ 0.00001
```

```
$ping0 set class_ 1
```

```
$ping5 set class_ 2
```

```
$ns connect $ping0 $ping4
```

```
$ns connect $ping5 $ping6
```

```
#Define a 'recv' function for the class 'Agent/Ping'
```

```
Agent/Ping instproc recv {from rtt} {
```

```
  $self instvar node_
```

```
  puts " The node [$node_ id] received an reply from $from with round trip time of  
  $rtt" } proc finish {}
```

```
  { global ns nf
```

```
  tf
```

```
  exec nam lab2.nam &
```

```
  $ns flush-trace close
```

```
  $tf
```

```
  close
```

```
  $nf
```

```
exit  
0  
}
```

```
#Schedule events
```

```
$ns at 0.1 "$ping0 send"  
$ns at 0.2 "$ping0 send"  
$ns at 0.3 "$ping0 send"  
$ns at 0.4 "$ping0 send"  
$ns at 0.5 "$ping0 send"  
$ns at 0.6 "$ping0 send"  
$ns at 0.7 "$ping0 send"  
$ns at 0.8 "$ping0 send"  
$ns at 0.9 "$ping0 send"
```

```
$ns at 1.0 "$ping0 send"  
$ns at 1.1 "$ping0 send"  
$ns at 1.2 "$ping0 send"  
$ns at 1.3 "$ping0 send"  
$ns at 1.4 "$ping0 send"  
$ns at 1.5 "$ping0 send"  
$ns at 1.6 "$ping0 send"  
$ns at 1.7 "$ping0 send"  
$ns at 1.8 "$ping0 send"
```

```
$ns at 0.1 "$ping5 send"  
$ns at 0.2 "$ping5 send"  
$ns at 0.3 "$ping5 send"  
$ns at 0.4 "$ping5 send"  
$ns at 0.5 "$ping5 send"  
$ns at 0.6 "$ping5 send"  
$ns at 0.7 "$ping5 send"  
$ns at 0.8 "$ping5 send"  
$ns at 0.9 "$ping5 send"  
$ns at 1.0 "$ping5 send"  
$ns at 1.1 "$ping5 send"  
$ns at 1.2 "$ping5 send"  
$ns at 1.3 "$ping5 send"  
$ns at 1.4 "$ping5 send"  
$ns at 1.5 "$ping5 send"  
$ns at 1.6 "$ping5 send"  
$ns at 1.7 "$ping5 send"  
$ns at 1.8 "$ping5 send"
```


\$ns at 5.0 "finish"

\$ns run

AWK:

```
BEGIN{  
count=0;  
}  
{  
if($1=="d")  
count++; } END{  
printf("The Total no of Packets Drop is :%d\n\n", count); }
```

Sample Output:

Result:

Step by Step Instruction – Explanation

Ex No.2: Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion

NS2 Initialization:

- Creates a new NS2 simulator object (ns).
- Opens trace and nam files (lab2.tr and lab2.nam) for writing.
- Enables tracing for all events in the simulation.

```
set ns [new Simulator]
set tf [open lab2.tr w]
$ns trace-all $tf set nf
[open lab2.nam w]
$ns namtrace-all $nf
```

Node and Link Configuration:

- Creates nodes and sets labels for them.
- Sets colors for nodes.
- Defines duplex links with specified characteristics.
- Sets queue limits for specific links.

```
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]
```

```
# Setting labels for nodes
```

```
$n0 label "Ping0"
```

```
$n4 label "Ping4"
```

```
$n5 label "Ping5"
```

```
$n6 label "Ping6"
```

```
$n2 label "Router"
```

```
# Setting colors for nodes
```

```
$ns color 1 "red"
```

```
$ns color 2 "green"
```

```
# Creating duplex links with specified bandwidth, delay, and queue type
```

```
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
```

```
$ns duplex-link $n1 $n2 1Mb 300ms DropTail
```

```
$ns duplex-link $n3 $n2 1Mb 300ms DropTail
```

```
$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4 1Mb 300ms DropTail
$ns duplex-link $n2 $n6 1Mb 300ms DropTail
```

Setting queue limits for specific links

```
$ns queue-limit $n0 $n2 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n2 $n6 2
$ns queue-limit $n5 $n2 5
```

Agent Configuration:

- Configures ping agents for each node.
- Sets parameters such as packet size and interval for ping agents.
- Assigns classes to ping agents and establishes connections between them.

```
set ping0 [new Agent/Ping] $ns
attach-agent $n0 $ping0
```

```
set ping4 [new Agent/Ping] $ns
attach-agent $n4 $ping4
```

```
set ping5 [new Agent/Ping] $ns
attach-agent $n5 $ping5
```

```
set ping6 [new Agent/Ping] $ns
attach-agent $n6 $ping6
```

```
$ping0 set packetSize_ 50000
$ping0 set interval_ 0.0001
```

```
$ping5 set packetSize_ 60000
$ping5 set interval_ 0.00001
```

```
$ping0 set class_ 1
$ping5 set class_ 2
```

```
$ns connect $ping0 $ping4
$ns connect $ping5 $ping6
```

Event Scheduling:

- Schedules multiple events for sending ping packets at different times.
- Schedules the finish event at time 5.0.

- Runs the NS2 simulation.

```
# Schedule events for ping0
```

```
$ns at 0.1 "$ping0 send"
```

```
# ... Repeat for other times
```

```
# Schedule events for ping5
```

```
$ns at 0.1 "$ping5 send"
```

```
# ... Repeat for other times
```

```
# Schedule finish event
```

```
$ns at 5.0 "finish"
```

```
# Run the simulation
```

```
$ns run
```

Finish Procedure and AWK Script:

- Defines the finish procedure, which is called at the end of the simulation.
- Opens the nam visualization tool for viewing the simulation.
- Flushes and closes the trace and nam files.
- Exits the simulation.

```
proc finish {} {
```

```
global ns nf tf exec
```

```
nam lab2.nam & $ns
```

```
flush-trace close $tf
```

```
close $nf exit 0
```

```
}
```

AWK Script:

- Begins the AWK script with an initialization block.

- Increments a counter for each line in the trace file where the first field is "d" (indicating a dropped packet).
- Prints the total number of dropped packets at the end.

```
BEGIN {  
    count = 0;  
}  
  
{  
    if ($1 == "d") count++;  
}  
  
END {  
    printf("The Total no of Packets Drop is: %d\n\n", count); }
```

Note 1: To use this script, save it with a **".tcl" extension (e.g., "lab2.tcl")** and run it using the NS2 simulator:

```
ns lab2.tcl
```

After the simulation finishes, you can use the AWK script to analyze the trace file:

```
awk -f awk_script.awk lab2.tr
```

Note 2: Replace **"awk_script.awk"** with the actual name of your AWK script file.

The AWK script calculates and prints the total number of dropped packets based on the trace file generated during the simulation.

Ex No. 3: Implement simple ESS (Extended Service Set) and with transmitting nodes in wire-less LAN by simulation and determine the throughput with respect to transmission of packets.

Aim: To implement simple ESS (Extended Service Set) and with transmitting nodes in wire-less LAN by simulation and determine the throughput with respect to transmission of packets and find the Role in simulating the wireless network and measuring throughput.

Algorithm:

- Step 1: Install NS-2
- Step 2: Create NS-2 Simulation Script (ess_simulation.tcl)
- Step 3: Create AWK Script (analyze.awk)
- Step 4: Run the Simulation
- Step 5: Evaluate the throughput of WLAN.

Sample Coding:

```
# Create a NS simulator object set
ns [new Simulator]
#setup trace support by opening file lab3.tr and call the procedure trace-all
set tf [open lab3.tr w] $ns trace-all $tf
#create a topology object that keeps track of movements of mobile nodes #within the topological
boundary. set topo [new Topography] $topo load_flatgrid 1000 1000 set nf [open lab3.nam w]
$ns namtrace-all-wireless $nf 1000 1000
# creating a wireless node you MUST first select (configure) the node #configuration parameters
to "become" a wireless node.
$ns node-config -adhocRouting DSDV \
  -llType LL \
  -macType Mac/802_11 \
  -ifqType Queue/DropTail \
  -ifqLen 50 \
  -phyType Phy/WirelessPhy \
  -channelType Channel/WirelessChannel \
  -propType Propagation/TwoRayGround \
  -antType Antenna/OmniAntenna \
  -topoInstance $topo \
  -agentTrace ON \
  -routerTrace ON
# Create god object
create-god 3 set n0
[$ns node] set n1
[$ns node] set n2
[$ns node] $n0
label "tcp0"
```



```

$ns1 label "sink1/tcp1"
$ns2 label "sink2"
$ns0 set X_ 50
$ns0 set Y_ 50
$ns0 set Z_ 0
$ns1 set X_ 100
$ns1 set Y_ 100
$ns1 set Z_ 0
$ns2 set X_ 600
$ns2 set Y_ 600
$ns2 set Z_ 0
$ns at 0.1 "$ns0 setdest 50 50 15"
$ns at 0.1 "$ns1 setdest 100 100 25"
$ns at 0.1 "$ns2 setdest 600 600 25"
set tcp0 [new Agent/TCP] $ns
attach-agent $ns0 $tcp0 set ftp0
[new Application/FTP] $ftp0
attach-agent $tcp0 set sink1 [new
Agent/TCPSink] $ns attach-agent
$ns1 $sink1 $ns connect $tcp0
$sink1 set tcp1 [new Agent/TCP]
$ns attach-agent $ns0 $tcp1 set ftp1
[new Application/FTP] $ftp0
attach-agent $tcp1 set sink2 [new
Agent/TCPSink] $ns attach-agent
$ns2 $sink2
$ns connect $tcp0 $sink2
$ns at 5 "$ftp0 start"
$ns at 5 "$ftp1 start"
$ns at 100 "$ns1 setdest 550 550 15"
$ns at 190 "$ns1 setdest 70 70 15" proc finish { } {
    global ns nf tf
    $ns flush-trace
    exec
    nam lab3.nam &
    close $tf
    exit 0
}
$ns at 250 "finish"
$ns run

```

AWK

```

BEGIN{
count1=0
count2=0
pack1=0

```

```
pack2=0
time1=0 time2=0
}
{
if($1 == "r" && $3 == "_1_" && $4 == "AGT")
{ count1++
pack1=pack1+$8
time1=$2
}
if($1 == "r" && $3 == "_2_" && $4 == "AGT")
{ count2++
pack2=pack2+$8
time2=$2
}
}
END{
printf("The Throughput from n0 to n1: %f Mbps \n", ((count1*pack1*8)/(time1*1000000)));
printf("The Throughput from n1 to n2: %f Mbps \n", ((count2*pack2*8)/(time2*1000000))); }
```

Sample Output:

Result:

Step By Step Instruction Explanation

Ex No.3 Implement simple ESS (Extended Service Set) and with transmitting nodes in wire-less LAN by simulation and determine the throughput with respect to transmission of packets.

Role in simulating the wireless network and measuring throughput.

NS Simulator Object Setup:

```
# Create a NS simulator object
set ns [new Simulator]
```

Trace file Setup:

```
#setup trace support by opening file lab3.tr and call the procedure trace-all set
tf [open lab3.tr w]
$ns trace-all $tf
```

Topology Setup:

```
#create a topology object that keeps track of movements of mobile nodes #within the topological
boundary. set topo [new Topography] $topo load_flatgrid 1000 1000
```

Nam (Network Animator) file Setup: set

```
nf [open lab3.nam w]
$ns namtrace-all-wireless $nf 1000 1000
```

Node Configuration:

```
# creating a wireless node you MUST first select (configure) the node #configuration parameters
to "become" a wireless node.
```

```
$ns node-config -adhocRouting DSDV \
  -llType LL \
  -macType Mac/802_11 \
  -ifqType Queue/DropTail \
  -ifqLen 50 \
  -phyType Phy/WirelessPhy \
  -channelType Channel/WirelessChannel \
  -propType Propagation/TwoRayGround \
  -antType Antenna/OmniAntenna \
  -topoInstance $topo \
  -agentTrace ON \
  -routerTrace ON God
```

Object Creation:

```
Create god object
create-god 3
```

Node Creation and Initialization:

```
set n0 [$ns node] set n1 [$ns node]
set n2 [$ns node] $n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"
```

Setting Node Coordinates:

```
$n0 set X_ 50
$n0 set Y_ 50
$n0 set Z_ 0
$n1 set X_ 100
$n1 set Y_ 100
$n1 set Z_ 0
$n2 set X_ 600
$n2 set Y_ 600
$n2 set Z_ 0
```

Setting Node Movements:

```
$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 100 100 25"
$ns at 0.1 "$n2 setdest 600 600 25"
```

TCP Agent1 and Application Setup:

```
set tcp0 [new Agent/TCP] $ns
attach-agent $n0 $tcp0 set ftp0
[new Application/FTP]
$ftp0 attach-agent $tcp0
```

TCPSink Agent1 Setup: set

```
sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
```

TCP Agent2 and Application Setup:

```
set tcp1 [new Agent/TCP] $ns attach-
agent $n0 $tcp1 set ftp1 [new
Application/FTP]
$ftp0 attach-agent $tcp1
```

TCPSink Agent2 Setup: set
sink2 [new Agent/TCPSink]
\$ns attach-agent \$n2 \$sink2
\$ns connect \$tcp0 \$sink2

Simulation Events:

\$ns at 5 "\$ftp0 start"
\$ns at 5 "\$ftp1 start"
\$ns at 100 "\$n1 setdest 550 550 15"
\$ns at 190 "\$n1 setdest 70 70 15"

Finish Procedure: proc

```
finish { } { global ns  
nf tf $ns flush-trace  
exec nam lab3.nam &  
close $tf exit 0  
}
```

Simulation Run:

\$ns at 250 "finish"
\$ns run

AWK

After the simulation finishes, you can use the AWK script to analyze the trace file:

```
awk -f awk_script.awk lab3.tr
```

Note : Replace "**awk_script.awk**" with the actual name of your AWK script file.

JAVA

Java is a general-purpose computer programming language that is simple, concurrent, class-based, object-oriented language. The compiled Java code can run on all platforms that support Java without the need for recompilation hence Java is called as "write once, run anywhere" (WORA).

The Java compiled intermediate output byte-code Java virtual machine (JVM) regardless of computer architecture. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

s very easy and convenient to compile and run Java programs in Linux environment. To compile and run Java Program is a two-step process:

1. Compile Java Program from Command Prompt [root@host ~]# javac Filename.java

The Java compiler (Javac) compiles java program and generates a byte-code with the same file name and .class extension.

2. Run Java program from Command Prompt [root@host ~]# java Filename

The java interpreter (Java) runs the byte-code and gives the respective output. It is important to note that in above command we have omitted the .class suffix of the byte-code (Filename.class).

Ex No. 4: Write a program for error detecting code using CRC-CCITT (16- bits) using java program.

Aim: To write a program for error detecting code using CRC-CCITT (16- bits) using java.

**General Information about the experiment CRC
(Cyclic Redundancy Check):**

- This is a mathematical algorithm used to detect errors in data.
- It involves dividing the data by a predefined divisor, and the remainder of this division is the CRC checksum.
- The sender appends this checksum to the data, and the receiver uses the same algorithm to check for errors.

CCITT (Consultative Committee for International Telephony and Telegraphy):

- This refers to the international organization that originally defined the CRCCCITT standard.

16-bits:

- This indicates the length of the CRC checksum in bits.
- In a CRC-CCITT (16-bits), the resulting checksum is 16 bits long.

Algorithm:

- Step 1: Define the CRC Parameters (0xFFFF)
- Step 2: Input Message (in binary from the user)
- Step 3: Convert Binary Message to Byte Array
- Step 4: Calculate CRC Checksum
- Step 5: Append CRC to Message
- Step 6: Simulate Transmission
- Step 7: Input Received Message
- Step 8: Convert Binary Received Message to Byte Array
- Step 9: Check for Errors
- Step 10: Display Result

Sample program

```
import java.io.*;
class Crc {
    public static void main(String args[]) throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        int[ ] data; int[ ]div; int[ ]divisor; int[ ]rem; int[ ]crc;
        int data_bits, divisor_bits, tot_length;
        System.out.println("Enter number of data bits : ");
        data_bits=Integer.parseInt(br.readLine()); data=new
        int[data_bits];

        System.out.println("Enter data bits : ");
        for(int i=0; i<data_bits; i++)
            data[i]=Integer.parseInt(br.readLine());

        System.out.println("Enter number of bits in divisor : ");
        divisor_bits=Integer.parseInt(br.readLine()); divisor=new
        int[divisor_bits];
        System.out.println("Enter Divisor bits : ");
        for(int i=0; i<divisor_bits; i++)
            divisor[i]=Integer.parseInt(br.readLine());
        tot_length=data_bits+divisor_bits-1;
        div=new int[tot_length]; rem=new
        int[tot_length]; crc=new int[tot_length];
        for(int i=0;i<data.length;i++)
            div[i]=data[i];
        System.out.print("Dividend (after appending 0's) are : ");
        for(int i=0; i<div.length; i++)
            System.out.print(div[i]);
        System.out.println(); for(int
        j=0; j<div.length; j++)
        {
            rem[j] = div[j];
        }
        rem=divide(div, divisor, rem);
        for(int i=0;i<div.length;i++)
        {
            crc[i]=(div[i]^rem[i]);
        }
    }
}
```



```

System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
System.out.print(crc[i]);
System.out.println();

System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(int i=0; i<crc.length; i++)
crc[i]=Integer.parseInt(br.readLine()); for(int
j=0; j<crc.length; j++)
{
rem[j] = crc[j];
}
rem=divide(crc, divisor, rem);
for(int i=0; i<rem.length; i++)
{
if(rem[i]!=0)
{
System.out.println("Error");
break; }
if(i==rem.length-1)
System.out.println("No Error");
}
}
static int[] divide(int div[],int divisor[], int rem[])
{ int
cur=0;
while(true)
{
for(int i=0;i<divisor.length;i++)
rem[cur+i]=(rem[cur+i]^divisor[i]); while(rem[cur]==0
&& cur!=rem.length-1)
cur++;
if((rem.length-cur)<divisor.length)
break; } return rem;
} }

```

Output:

Command Prompt

```
E:\java>javac Crc.java
```

```
E:\java>java Crc
```

```
Enter number of data bits :
```

```
7
```

```
Enter data bits :
```

```
1
```

```
0
```

```
0
```

```
0
```

```
0
```

```
0
```

```
1
```

```
Enter number of bits in divisor :
```

```
3
```

```
Enter Divisor bits :
```

```
1
```

```
0
```

```
1
```

```
Dividend (after appending 0's) are : 100000100
```

```
CRC code :
```

```
100000100
```

```
Enter CRC code of 9 bits :
```

```
1
```

```
0
```

```
0
```

```
0
```

```
0
```

```
0
```

```
1
```

```
0
```

```
0
```

```
No Error
```

```
E:\java>_
```

Step by Step Explanation

Install java jdk

Step 1: Importing Necessary Libraries

```
import java.io.*;
```

This line imports the necessary libraries for input and output handling.

Step 2: Main Class Definition

```
class Crc {  
    public static void main(String args[]) throws IOException  
    {
```

This declares the main class Crc and the main method. The throws IOException is used to handle input/output exceptions.

Step 3: Variable Declaration

```
        int[] data;  
int[] div;    int[]  
divisor;    int[]  
rem;    int[]  
crc;  
        int data_bits, divisor_bits, tot_length;
```

Here, the program declares arrays (data, div, divisor, rem, crc) and integer variables (data_bits, divisor_bits, tot_length) to store data, divisor, remainder, CRC, and other information.

Step 4: Input Data Bits

```
        System.out.println("Enter number of data bits : ");  
data_bits=Integer.parseInt(br.readLine());    data=new  
int[data_bits];
```

```
        System.out.println("Enter data bits : ");  
for(int i=0; i<data_bits; i++)  
data[i]=Integer.parseInt(br.readLine());
```

This section prompts the user to enter the number of data bits and the actual data bits. It reads the input using BufferedReader and stores the data in the data array.

Step 5: Input Divisor Bits

```
System.out.println("Enter number of bits in divisor : ");
divisor_bits=Integer.parseInt(br.readLine());  divisor=new
int[divisor_bits];
```

```
System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
    divisor[i]=Integer.parseInt(br.readLine());
```

This section prompts the user to enter the number of bits in the divisor and the actual divisor bits. It reads the input using BufferedReader and stores the divisor in the divisor array.

Step 6: Total Length Calculation

```
tot_length=data_bits+divisor_bits-1;
div=new int[tot_length];  rem=new
int[tot_length];  crc=new
int[tot_length];
```

It calculates the total length required for the division and initializes arrays div, rem, and crc accordingly.

Step 7: Dividend Initialization

```
for(int i=0;i<data.length;i++)
    div[i]=data[i];
```

It initializes the dividend array div with the data bits.

Step 8: Display Dividend

```
System.out.print("Dividend (after appending 0's) are : ");
for(int i=0; i<div.length; i++)
    System.out.print(div[i]);
System.out.println();
```

It displays the dividend after appending 0's.

Step 9: Copy Data Bits to Remainder

```
for(int j=0; j<div.length; j++)
```

```

{
    rem[j] = div[j];
}

```

It copies the data bits into the remainder array.

Step 10: CRC Calculation

```

rem=divide(div, divisor, rem);
for(int i=0;i<div.length;i++)
{
    crc[i]=(div[i]^rem[i]);
}

```

It calls the divide method (explained later) to perform the CRC calculation and then calculates the CRC bits by XORing the dividend and remainder.

Step 11: Display CRC

```

System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
    System.out.print(crc[i]);
System.out.println();

```

It displays the calculated CRC code.

Step 12: User Input for CRC

```

System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(int i=0; i<crc.length; i++)
    crc[i]=Integer.parseInt(br.readLine());

```

It prompts the user to enter the CRC code for verification.

Step 13: Copy CRC Bits to Remainder

```

for(int j=0; j<crc.length; j++)
{
    rem[j] = crc[j];
}

```

It copies the CRC bits into the remainder array for verification.

Step 14: Check for Errors

```

    rem=divide(crc, divisor, rem);
    for(int i=0; i<rem.length; i++)
    {
        if(rem[i]!=0)
        {
            System.out.println("Error");
            break;
        }
        if(i==rem.length-1)
            System.out.println("No Error");
    }

```

It calls the divide method to perform division and checks if there are any remainder bits. If there are, it prints an error message; otherwise, it indicates no error.

Step 15: Closing Braces

```

    }
    static int[] divide(int div[],int divisor[], int rem[])
    {
        // Implementation of the divide method
    }
}

```

This concludes the main method and the Crc class. The program also includes a divide method, which performs the actual polynomial division.

Steps to execute the program

- Edit the java program in note pad and save with extension of .java (Crc.java)

- Compile java program **javac Crc.java**

- Run the program

Java Crc

Ex No. 5: Write a program to find the shortest path between vertices using bellman ford algorithm.

Aim: To write a program to find the shortest path between vertices using bellman-ford algorithm

General Information about the experiment

Distance Vector Algorithm is a decentralized routing algorithm that requires that each router simply inform its neighbors of its routing table. For each network path, the receiving routers pick the neighbor advertising the lowest cost, then add this entry into its routing table for readvertisement. To find the shortest path, Distance Vector Algorithm is based on one of two basic algorithms: the Bellman-Ford and the Dijkstra algorithms.

Routers that use this algorithm have to maintain the distance tables (which is a one-dimension array - - "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always up date by exchanging information with the neighboring nodes. The amount of data in the table equals to that of all nodes in networks (excluded itself).

The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc.

The Bellman Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm.

If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman Ford algorithm can detect negative cycles and report their existence.

Algorithm:

Step 1. Send my routing table to all my neighbors whenever my link table changes Step

2. When I get a routing table from a neighbor on port P with link metric M:

a. add L to each of the neighbor's metrics

b. for each entry (D, P', M') in the updated neighbor's table:

i. if I do not have an entry for D, add (D, P, M') to my routing table

ii. if I have an entry for D with metric M", add (D, P, M') to my routing table if
M'

< M"

Step 3. If my routing table has changed, send all the new entries to all my neighbor.

	1	2	3	4
1	0	99	99	5
2	5	0	66	1
3	99	88	0	6
4	99	55	99	0

Sample Program

```
import java.util.*;
```

```
public class BellmanFord {  
    private int D[];  
    public static final int max_value = 999;
```

```
    public BellmanFord(int n) {  
        D = new int[n + 1];  
    }
```

```
    public void shortest(int s, int a[][]) {  
        int n = a.length - 1;  
        for (int i = 1; i <= n; i++) {  
            D[i] = max_value;  
        }  
        D[s] = 0;  
        for (int k = 1; k <= n - 1; k++) {  
            for (int i = 1; i <= n; i++) {  
                for (int j = 1; j <= n; j++) {  
                    if (a[i][j] != max_value) {  
                        if (D[j] > D[i] + a[i][j]) {  
                            D[j] = D[i] + a[i][j];  
                        }  
                    }  
                }  
            }  
        }  
        for (int i = 1; i <= n; i++) {  
            for (int j = 1; j <= n; j++) {
```



```

        if (a[i][j] != max_value) {
            if (D[j] > D[i] + a[i][j]) {
                System.out.println("The graph contains a negative edge cycle.");
            }
            return;
        }
    }
}

for (int i = 1; i <= n; i++) {
    System.out.println("Distance from source " + s + " to " + i + " is " + D[i]);
}
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of vertices:");
    int n = sc.nextInt();
    int a[][] = new int[n + 1][n + 1];
    System.out.println("Enter the weighted matrix:");
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            a[i][j] = sc.nextInt();
            if (i == j) {
                a[i][j] = 0;
            }
            if (a[i][j] == 0) {
                a[i][j] = max_value;
            }
        }
    }
    System.out.println("Enter the source vertex:");
    int s = sc.nextInt();
    BellmanFord b = new BellmanFord(n);
    b.shortest(s, a);
    sc.close();
}
}

```

Step by Step Explanation

Importing Necessary Package:

The code begins by importing the java.util package, which contains utility classes such as Scanner.

```
import java.util.*;
```

Defining the BellmanFord Class:

```
public class BellmanFord {
```

Private Variables: The class includes a private integer distance array `D[]` and a constant `max_value`.

```
private int D[];  
public static final int max_value = 999;
```

Constructor: A constructor is defined to initialize the distance array `D[]` based on the number of vertices passed as a parameter. `public BellmanFord(int n) {`

```
    D = new int[n + 1];  
}
```

Shortest Path Method: The shortest method calculates the shortest paths from a source vertex `s` to all other vertices in the graph represented by the adjacency matrix `a[][]`.

```
public void shortest(int s, int a[][]) {
```

Initializing Variables: The method initializes some variables, including `n`, the number of vertices. `int n = a.length - 1;`

Initializing Distance Array: The distance array `D[]` is initialized with a maximum value for all vertices except the source vertex, which is set to 0.

```
    for (int i = 1; i <= n; i++) {  
        D[i] = max_value;  
    }  
    D[s] = 0;
```

Bellman-Ford Algorithm: The nested loops implement the Bellman-Ford algorithm to find the shortest paths. The outer loop runs `n - 1` times (where `n` is the number of vertices), and the inner loops iterate through all edges of the graph.

```

        for (int k = 1; k <= n - 1; k++) {
    for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
    if (a[i][j] != max_value) {
    if (D[j] > D[i] + a[i][j]) {
                D[j] = D[i] + a[i][j];
    }
        }
    }
    }
}

```

Detecting Negative Cycles: After the main loop, the method checks for negative cycles in the graph.

```

    for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
    if (a[i][j] != max_value) {
    if (D[j] > D[i] + a[i][j]) {
                System.out.println("The graph contains a negative edge cycle.");
    return;
    }
    }
    }
}

```

Printing Results: Finally, the method prints the shortest distances from the source vertex *s* to all other vertices.

```

    for (int i = 1; i <= n; i++) {
        System.out.println("Distance from source " + s + " to " + i + " is " + D[i]);
    }

```

Main Method: The main method reads input from the user, creates an instance of the BellmanFord class, and calls the shortest method to find the shortest paths.

```

    public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of vertices:");
    int n = sc.nextInt();
        int a[][] = new int[n + 1][n + 1];
        System.out.println("Enter the weighted matrix:");
    }

```

```

        // Code for inputting the weighted matrix
        System.out.println("Enter the source vertex:");    int
        s = sc.nextInt();
        BellmanFord b = new BellmanFord(n);
        b.shortest(s, a);
        sc.close();
    }
}

```

Output 1

```

Enter the number of vertices:
4
Enter the weighted matrix:( 4X4 matrix)
0 33 33 33
9 0 33 66
33 66 0 99
22 44 88 0 Enter the
source vertex:
2
Distance from source 2 to 1 is 9
Distance from source 2 to 2 is 0
Distance from source 2 to 3 is 33
Distance from source 2 to 4 is 42

```

Output 2

```

Enter the number of vertices:
4
Enter the weighted matrix:( 4X4 matrix)
0 99 99 99
5 0 3 4
99 99 0 2
99 99 -15 0 Enter the
source vertex:
1
The graph contains a negative edge cycle.

```

Result:

Ex No. 6: Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

Aim: To implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination

Algorithm

- Step1: Initializes the simulator and opens trace files for nam (network animator) and trace data.
- Step 2: Finish procedure to close trace files and execute the network animator.
- Step 3: Creates nodes and shapes them as required.
- Step 4: Sets up links between nodes with specified characteristics like bandwidth, delay, and queue type.
- Step 5: Defines TCP agents and attaches them to nodes.
- Step 6: Defines a TCP Sink agent to receive TCP traffic.
- Step 7: Attaches the FTP application to the TCP agent.
- Step 8: Sets up tracing for congestion window changes.
- Step 9: Schedules the start and stop of FTP traffic.
- Step 10: Runs the simulation.

Sample Program

```
set ns [new Simulator] set nf [open lab3.nam w]
$ns namtrace-all $nf set nd [open lab3.tr w]
$ns trace-all $nd
$ns color 1 Blue
$ns color 2 Red proc finish { } { global ns nf nd
$ns flush-trace close $nf close $nd
exec nam lab3.nam & exit 0
}
set n0 [$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] set n7 [$ns node] set n8 [$ns node]
$n7 shape box
$n7 color Blue
$n8 shape hexagon
$n8 color Red
$ns duplex-link $n1 $n0 2Mb 10ms DropTail
$ns duplex-link $n2 $n0 2Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 20ms DropTail
$ns make-lan "$n3 $n4 $n5 $n6 $n7 $n8" 512Kb 40ms LL Queue/DropTail Mac/802_3
$ns duplex-link-op $n1 $n0 orient right-down
```

```

$ns duplex-link-op $n2 $n0 orient right-up
$ns duplex-link-op $n0 $n3 orient right
$ns queue-limit $n0 $n3 20
set tcp1 [new Agent/TCP/Vegas]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n7 $sink1
$ns connect $tcp1 $sink1
$tcp1 set class_ 1
$tcp1 set packetSize_ 55
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set tfile [open cwnd.tr w]
$tcp1 attach $tfile
$tcp1 trace cwnd_
$ns at 0.5 "$ftp1 start"
$ns at 1.0 "$ftp2 start"
$ns at 5.0 "$ftp2 stop"
$ns at 5.0 "$ftp1 stop"
$ns at 5.5 "finish"
$ns run

```

AWK File:

```

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

```

Output

```

[root@localhost ~]# ns lab3.tcl
[root@localhost ~]# awk -f lab3.awk file1.tr>tcp1
[root@localhost ~]# awk -f lab3.awk file2.tr>tcp2
[root@localhost ~]# xgraph -x "time" -y "convalue" tcp1 tcp2

```

Explanation about the Program

Initialization:

The script begins by creating a new simulator (ns) object and opening files (lab3.nam and lab3.tr) for namtrace and trace output, respectively.

Setting Up Nodes and Links:

Nodes (n0 to n8) are created using the node command.

Different shapes and colors are assigned to nodes n7 and n8.

Duplex links are created between nodes using the duplex-link command, specifying bandwidth, delay, and queuing discipline (e.g., DropTail).

LAN Creation:

A LAN is created using the make-lan command, connecting multiple nodes (n3 to n8) with a specified bandwidth and delay.

Setting Link Orientation and Queue Limit:

Link orientations are specified using the duplex-link-op command.

A queue limit is set between nodes n0 and n3.

Defining Traffic Sources and Sinks:

A TCP agent (tcp1) and a TCP sink (sink1) are created.

An FTP application (ftp1) is attached to tcp1.

Setting Simulation Events:

The ftp1 application is started and stopped at specific times using the at command.

A function finish is called at the end of the simulation to finalize and close resources.

AWK Script:

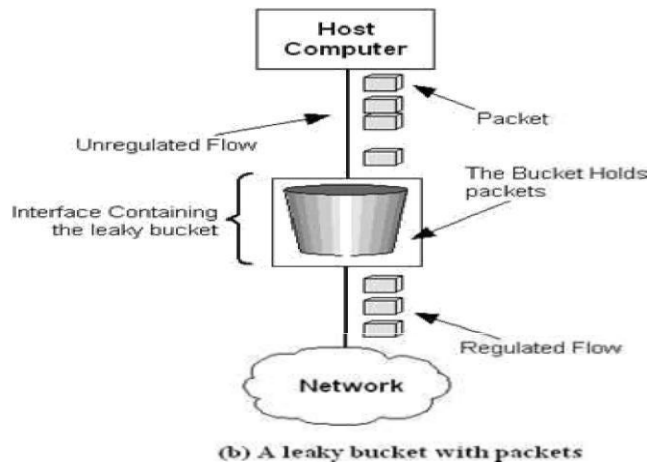
An AWK script is provided to process the trace output (cwnd.tr) generated during the simulation. It prints the time and congestion window (cwnd) whenever a line containing "cwnd_" is encountered.

Ex No. 7: Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

Aim: To write a program for congestion control using leaky bucket algorithm

The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spillover. In a similar way if the bucket is empty the output will be zero.

From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick). In the following figure we can notice the main rationale of leaky bucket algorithm, for both the two approaches (e.g. leaky bucket with water (a) and with packets (b)).



While leaky bucket eliminates completely Bursty traffic by regulating the incoming data flow its main drawback is that it drops packets if the bucket is full. Also, it does not take into account the idle process of the sender which means that if the host dose not transmit data for some time the bucket becomes empty without permitting the transmission of any packet.

The leaky-bucket algorithm:

The algorithm can be conceptually understood as follows:

- Consider a bucket with a hole in the bottom.
- The empty space of the bucket represents an amount of credit available measured in bytes.
- The size of the bucket is b bytes. This means that if the bucket is empty, b bytes of credit is available.

- If a packet arrives and its size is less than the available credit, the packet can be forwarded. Otherwise, it is discarded or queued depending on the application.
- The bucket leaks through the hole in its bottom at a constant rate of r bytes per second, this indicates credit accumulation.

Sample Code

```

/* Leaky Bucket */
public class LeakyBucket
{
    static int min(int x,int y)
    {
        if(x<y) return x; else return y;
    }
    public static void main(String[] args)
    {
        int drop=0,mini,nsec,cap,count=0,i,process;
        int inp[]=new int[25];
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter The Bucket Size\n");
        cap= sc.nextInt();
        System.out.println("Enter The Operation Rate\n");
        process= sc.nextInt();
        System.out.println("Enter The No. Of Seconds You Want To Stimulate\n");
        nsec=sc.nextInt();
        for(i=0;i<nsec;i++)
        {
            System.out.print("Enter The Size Of The Packet Entering At "+ i+1+"sec");
            inp[i] = sc.nextInt();
        }
        System.out.println("\nSecond | Packet Recieved | Packet Sent | Packet Left | Packet Dropped|\n");
        //System.out.println(" \n");
        for(i=0;i<nsec;i++)
        {
            count+=inp[i]; if(count>cap)
            {
                drop=count-cap; count=cap;
            }
            System.out.print(i+1);
            System.out.print("\t\t"+inp[i]);
            mini=min(count,process);
            System.out.print("\t\t"+mini);
            count=count-mini;

```

```

System.out.print("\t\t"+count);
System.out.print("\t\t"+drop);
drop=0; System.out.println();
}
for(;count!=0;i++)
{
if(count>cap)
{
drop=count-cap; count=cap;
}
System.out.print(i+1);
System.out.print("\t\t0");
mini=min(count,process);
System.out.print("\t\t"+mini);
count=count-mini;
System.out.print("\t\t"+count);
System.out.print("\t\t"+drop);
System.out.println();
}
}
}

```

Output 1

```

Enter The Bucket Size
100
Enter The Operation Rate
10
Enter The No. Of Seconds You Want To Stimulate
5
Enter The Size Of The Packet Entering At 1 sec: 20
Enter The Size Of The Packet Entering At 2 sec: 30
Enter The Size Of The Packet Entering At 3 sec: 40
Enter The Size Of The Packet Entering At 4 sec: 50
Enter The Size Of The Packet Entering At 5 sec: 60

```

Second	Packet Received	Packet Sent	Packet Left	Packet Dropped
1	20	10	10	0
2	30	10	30	0
3	40	10	40	0
4	50	10	50	0
5	60	10	60	0
6	0	0	50	0

7	0	0	40	0
8	0	0	30	0
9	0	0	20	0
10	0	0	10	0

Output 2

Enter The Bucket Size 5

Enter The output Rate 2

Enter The No. of Seconds You Want To Stimulate 3

Enter The Size of Packet entering at 01sec 5

Enter The Size of Packet entering at 11sec 4

Enter The Size of Packet entering at 21sec 3

Second | Packet Recieved | Packet Sent | Packet Left | Packet Dropped|

1	5	2	3	0
2	4	2	3	2
3	3	2	3	1
4	0	2	1	0
5	0	1	0	0

Step by Step Explanation of program

Step 1: Import Required Packages

```
import java.util.Scanner;
```

This line imports the Scanner class from the java.util package.

Scanner is used to read input from the user.

Step 2: Define the LeakyBucket Class

```
public class LeakyBucket {  
    // Methods and variables will be defined within this class  
}
```

Defines a class named LeakyBucket.

This class contains the main method where the program execution starts.

Step 3: Define a Static min Method

```
static int min(int x, int y) {  
    if (x < y)  
        return x;  
    else  
        return y;  
}
```

Defines a static method min that returns the minimum of two integers x and y.

This method is later used to find the minimum of two values.

Step 4: Define the main Method

```
public static void main(String[] args) {  
    // Main logic of the program will be written here  
}
```

Defines the main method, which is the entry point of the program.

Step 5: Declare Variables

```
int drop = 0, mini, nsec, cap, count = 0, i, process;  
int inp[] = new int[25];
```

Declares variables to hold various values used in the program.

These include the number of packets dropped (drop), the minimum value (mini), the number of seconds to simulate (nsec), the bucket capacity (cap), the count of packets (count), the operation rate (process), and an array to store packet sizes (inp).

Step 6: Create a Scanner Object

```
Scanner sc = new Scanner(System.in);
```

Creates a Scanner object named sc to read input from the console.

Step 7: Prompt User for Input

```

System.out.println("Enter The Bucket Size\n");
cap = sc.nextInt();
System.out.println("Enter The Operation Rate\n");
process = sc.nextInt();
System.out.println("Enter The No. Of Seconds You Want To Stimulate\n");
nsec = sc.nextInt();

```

Prompts the user to enter the bucket size, operation rate, and the number of seconds to simulate.
Reads the input values using the Scanner object.

Step 8: Input Packet Sizes

```

for (i = 0; i < nsec; i++) {
    System.out.print("Enter The Size Of The Packet Entering At " + (i + 1) + " sec: ");
    inp[i] = sc.nextInt();
}

```

Prompts the user to enter the size of the packet entering the bucket at each second.
Reads the input values and stores them in the inp array.

Step 9: Simulate Leaky Bucket Algorithm

```

for (i = 0; i < nsec; i++) {
    // Logic to simulate the leaky bucket algorithm for each second
}
for (; count != 0; i++) {
    // Logic to handle remaining packets after simulation
}

```

Contains two loops to simulate the leaky bucket algorithm.
The first loop processes each second of the simulation, while the second loop handles any remaining packets after the simulation is complete.

Step 10: Output Results

```

System.out.println("\nSecond | Packet Recieved | Packet Sent | Packet Left |
Packet Dropped\n");

```

Prints a header for the output table.

Step 11: Finalize and Close Resources

```

sc.close();

```

Closes the Scanner object to release system resources.