

---

# **IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2 AND IPV4 ADDRESSING**

---

**CSE1004(NETWORK AND COMMUNICATION)LAB:L53-L54**



**APRIL 19, 2022  
ANIRUDH VADERA  
20BCE2940**

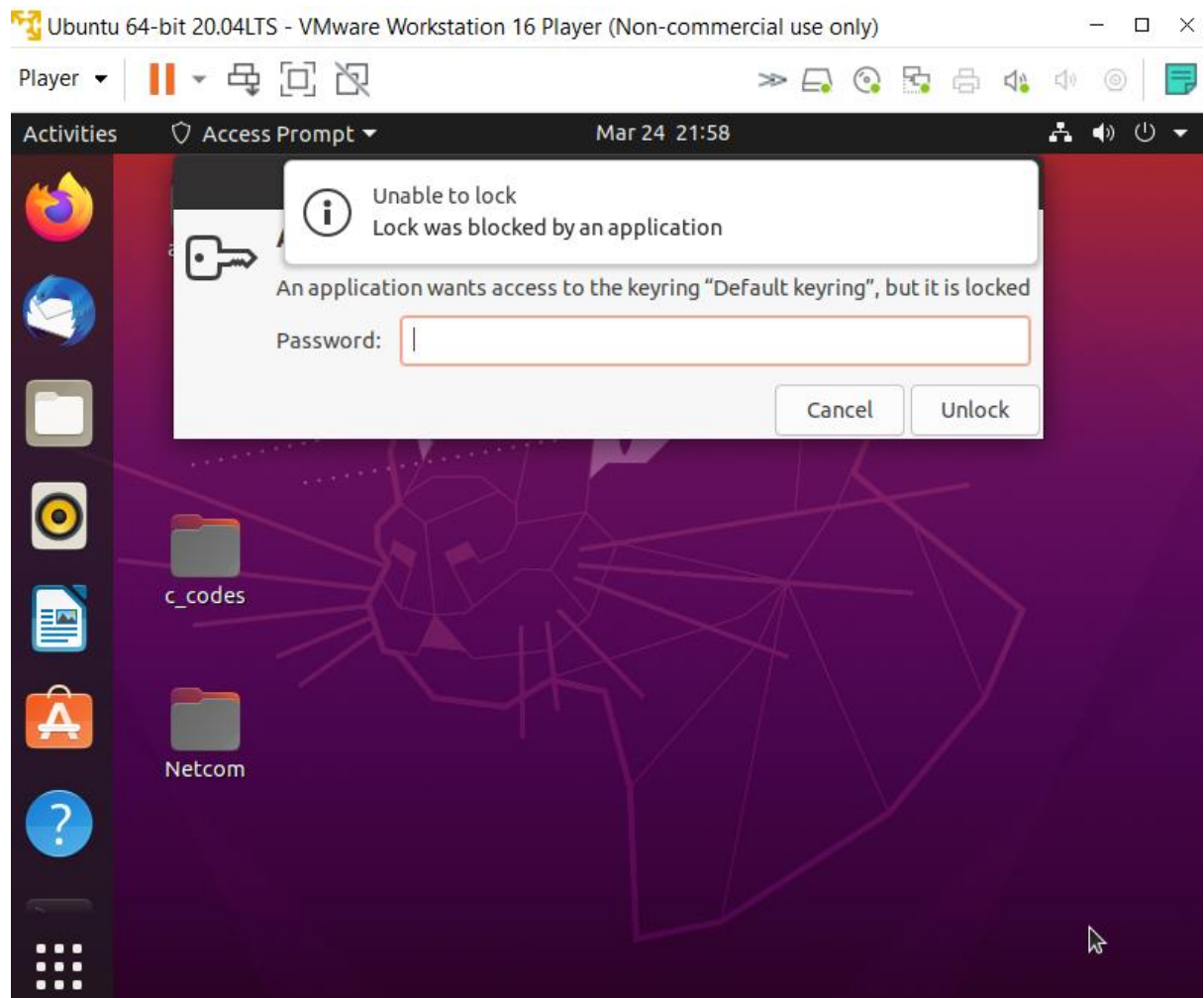
## SIMULATION AND NS2:

**Network simulation** (NS) is one of the types of simulation, which is used to simulate the networks such as in MANETs, VANETs, etc. It provides simulation for routing and multicast protocols for both wired and wireless networks. NS is licensed for use under version 2 of the GNU (General Public License) and is popularly known as **NS2**. It is an object-oriented, discrete event-driven simulator written in C++ and Otcl/Tcl.

NS-2 can be used to implement network protocols such as TCP and UDP, traffic source behavior such as FTP, Telnet, Web, CBR, and VBR, router queues management mechanism such as Drop Tail, RED, and CBQ, routing algorithms, and many more. In ns2, C++ is used for detailed protocol implementation and Otcl is used for the setup. The compiled C++ objects are made available to the Otcl interpreter and in this way, the ready-made C++ objects can be controlled from the OTcl level.

## PROCEDURE:

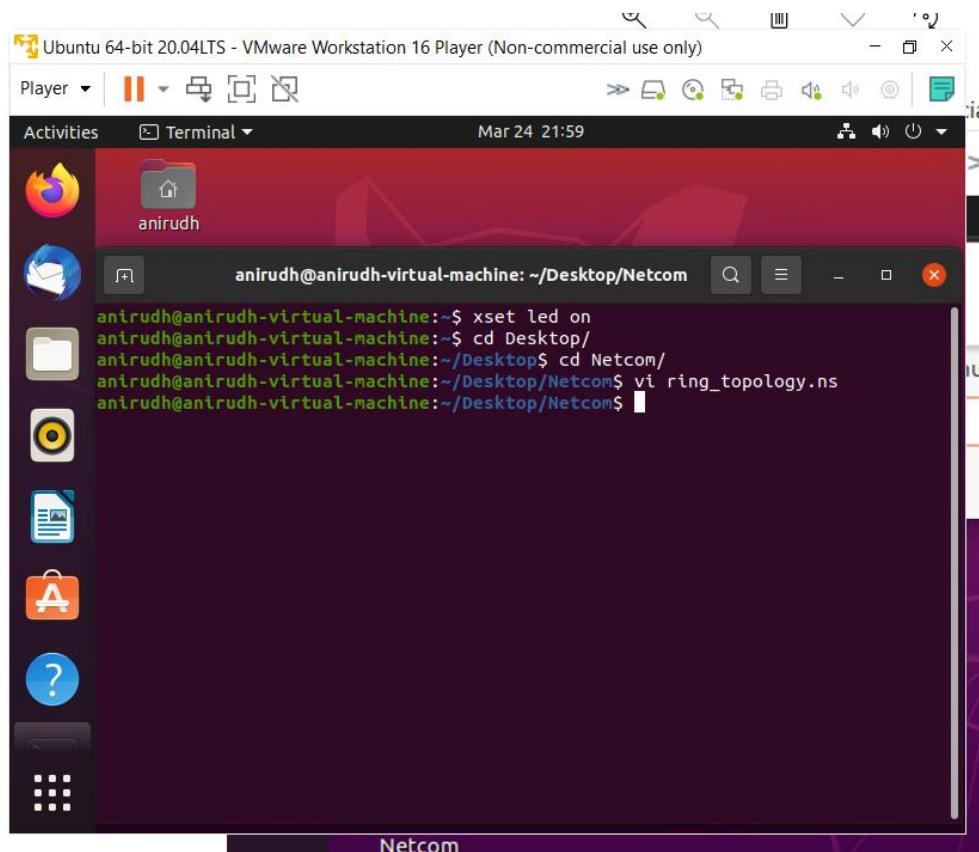
- **Open ur vmware having a Ubuntu linux distribution:**



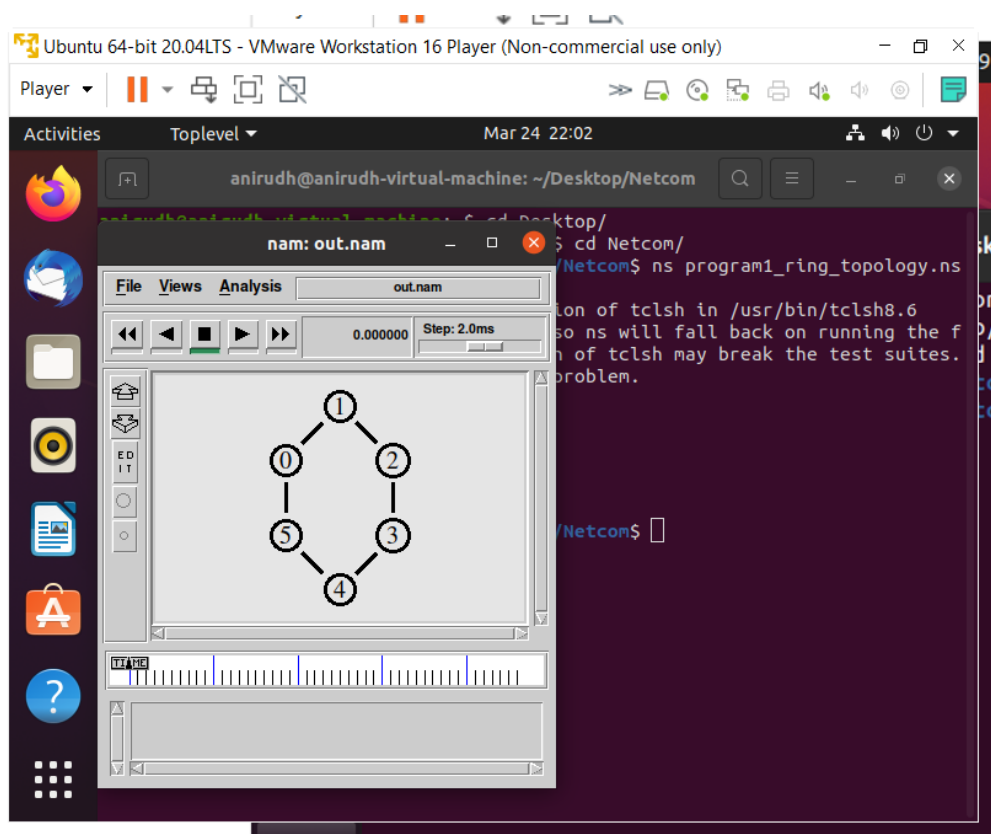
- **U must have all the necessary c and ns2 compilers already installed in your linux system.**
- **Using terminal open a vi editor and write ur ns2 code with the command ns filename.ns**

## ANIRUDH VADERA

### IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2 AND IPV4 ADDRESSING



- After saving the file run **ns filename.ns** command to execute it using **nam**

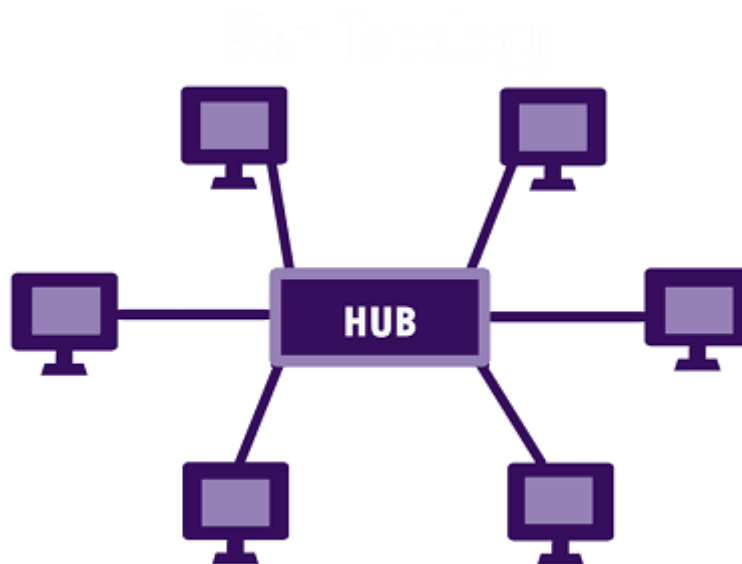


## QUESTION:

**To simulate and study the link state routing algorithm using simulation**

### Star Topology:

A star topology, sometimes known as a star network, is a network topology in which each device is connected to a central hub. It is one of the most prevalent computer network configurations, and it's by far the most popular Network Topology. In this network arrangement, all devices linked to a central network device are displayed as a star.



### THEORY:

In link state routing, each router shares its knowledge of its neighborhood with every other router in the internet work. (i) Knowledge about Neighborhood: Instead of sending its entire routing table a router sends info about its neighborhood only. (ii) To all Routers: each router sends this information to every other router on the internet work not just to its neighbor .It does so by a process called flooding. (iii)Information sharing when there is a change: Each router sends out information about the neighbors when there is change.

## PROCEDURE:

The Dijkstra algorithm follows four steps to discover what is called the shortest path tree(routing table) for each router: The algorithm begins to build the tree by identifying its roots. The root router's trees the router itself. The algorithm then attaches all nodes that can be reached from the root. The algorithm compares the tree's temporary arcs and identifies the arc with the lowest cumulative cost. This arc and the node to which it connects are now a permanent part of the shortest path tree. The algorithm examines the database and identifies every node that can be reached from its chosen node. These nodes and their arcs are added temporarily to the tree. The last two steps are repeated until every node in the network has become a permanent part of the tree.

## ALGORITHM:

1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose Link state routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

## CODE:

### Step-1: Initializing the network :

**The first step is to initialize the network simulator, and by creating a network simulator object. Initialize rtpeto (routing protocol) to Link State (LS).**

```
#Create a simulator object  
set ns [new Simulator]  
  
#Routing Protocol used is Link State  
  
$ns rtpeto LS
```

### **Step-2: Creating the trace file :**

**Create the trace file and nam file. The nam file is used to view simulator output whereas the trace file traces all the routing information in the process. For this we create trace file and nam file objects and then open the files in write mode. The trace-all instance is used to trace all routing information into the trace file and similarly namtrace-all for the nam file.**

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

### **Step-3: Adding a finish procedure :**

**The next step is to add a finish procedure to flush all data into trace file and then and then run the nam file.**

```
#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
```

}

#### **Step-4: Creating number of nodes :**

**Create a number of nodes, let's say 5. Use the node instance to create these nodes as follows.**

```
#Create nodes  
  
set n0 [$ns node]  
set n1 [$ns node]  
set n2 [$ns node]  
set n3 [$ns node]  
set n4 [$ns node]
```

#### **Step-5: Labeling the nodes:**

**Customize the labels by assigning different colors to them and thus viewing the simulation much more clearly. Use red and blue and green.**

```
$ns color 1 red  
$ns color 2 green  
$ns color 3 blue  
$n0 shape square
```

#### **Step-6: Creating duplex links :**

**The next step is to create duplex links between the nodes forming a ring in the end. This can be achieved by using the duplex-link instance along with specifying three parameters: data rate (1Mb), delay (10ms) and kind of queue (DropTail/SFQ).**

```
#Create links between the nodes  
  
$ns duplex-link $n0 $n1 1Mb 10ms DropTail  
$ns duplex-link $n0 $n2 1Mb 10ms DropTail  
$ns duplex-link $n0 $n3 1Mb 10ms DropTail  
$ns duplex-link $n0 $n4 1Mb 10ms SFQ
```



### **Step-7: Orient the links between the nodes :**

**To orient the links between the nodes appropriately to obtain proper alignment. The duplex-link-op instance is used for the same.**

```
$ns duplex-link-op $n0 $n1 orient right-up
```

```
$ns duplex-link-op $n0 $n2 orient right-down
```

```
$ns duplex-link-op $n0 $n3 orient left-down
```

```
$ns duplex-link-op $n0 $n4 orient left-up
```

### **Step-8: Attaching TCP agents :**

**The next step is to attach UDP agents. Creating the source and sink objects and connecting them using connect instance.**

```
#Create a UDP agent and attach it to sources
```

```
set udp1 [new Agent/UDP]
```

```
$udp1 set class_ 1
```

```
$ns attach-agent $n1 $udp1
```

```
set udp2 [new Agent/UDP]
```

```
$udp2 set class_ 2
```

```
$ns attach-agent $n2 $udp2
```

```
set udp3 [new Agent/UDP]
```

```
$udp3 set class_ 3
```

```
$ns attach-agent $n3 $udp3
```

```
#Create a UDP Sink agent (a traffic sink) for UDP and attach it to node n4
```

```
set null0 [new Agent/Null]
```

```
$ns attach-agent $n4 $null0
```

```
#Connect the traffic sources with the traffic sink
```

```
$ns connect $udp1 $null0
```

```
$ns connect $udp2 $null0
```

```
$ns connect $udp3 $null0
```

### **Step-9: Creating CBR traffic :**

# Create a CBR traffic source and attach it to udp1

```
set cbr1 [new Application/Traffic/CBR]
```

```
$cbr1 set packetSize_ 500
```

```
$cbr1 set interval_ 0.01
```

```
$cbr1 attach-agent $udp1
```

```
set cbr2 [new Application/Traffic/CBR]
```

```
$cbr2 set packetSize_ 500
```

```
$cbr2 set interval_ 0.01
```

```
$cbr2 attach-agent $udp2
```

```
set cbr3 [new Application/Traffic/CBR]
```

```
$cbr3 set packetSize_ 500
```

```
$cbr3 set interval_ 0.01
```

```
$cbr3 attach-agent $udp3
```

### **Step-10: Scheduling the CBR Agents:**

The final step is to schedule the traffic at the required time intervals. We can also disable the link between any pair of nodes at a certain timestamp using `rtmodel-at` instance and then enable it after a certain time. This is majorly done for testing purposes. Here we have disabled the link between nodes 0 and 1. The program ends with the `run` command.

#Schedule events for the CBR agents

```
$ns at 0.2 "$cbr1 start"
```

```
$ns at 0.4 "$cbr2 start"
```

```
$ns at 0.6 "$cbr3 start"
```

```
$ns at 4.2 "$cbr3 stop"
```

```
$ns at 4.4 "$cbr2 stop"
```

```
$ns at 4.6 "$cbr1 stop"
```

#Call the finish procedure after 5 seconds of simulation time

ANIRUDH VADERA  
IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2 AND IPV4 ADDRESSING

\$ns at 5.0 "finish"

#Simulate node failure/restoration

\$ns rtmodel-at 1.6 down \$n2

#Node failure

\$ns rtmodel-at 2.5 up \$n2

#Node restoration

#Simulating link failure

\$ns rtmodel-at 1.0 down \$n0 \$n1

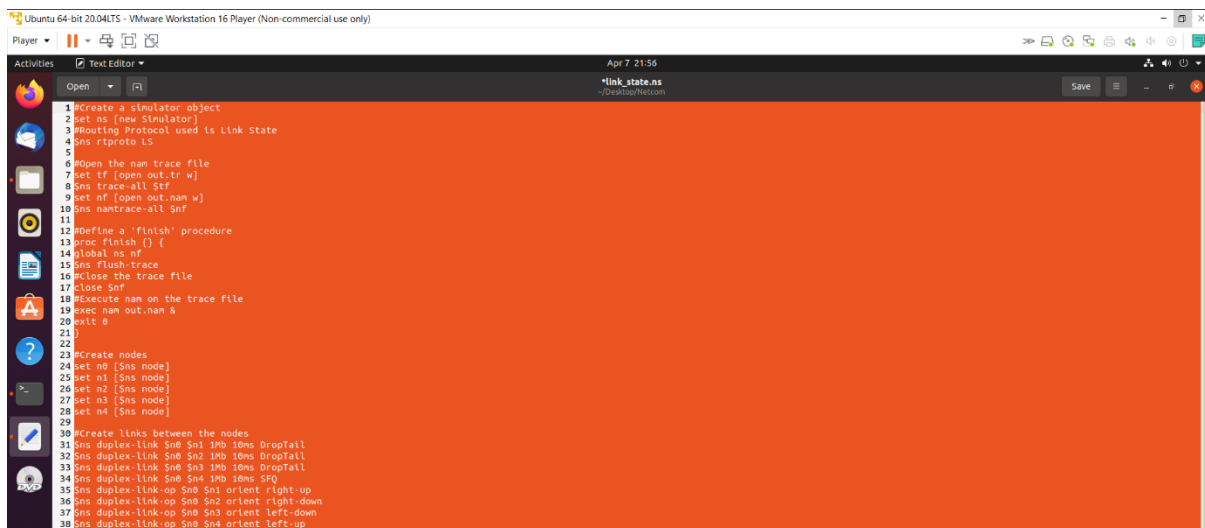
\$ns rtmodel-at 2.0 up \$n0 \$n1

#Run the simulation

\$ns run

#ANIRUDH VADERA(20BCE2940)

## CODE SNAPSHOTS:



```
1 #create a simulator object
2 set ns [new Simulator]
3 Routing Protocol used is Link State
4 $ns rtrproto ls
5
6 #open the nam trace file
7 set tr [open out.tr w]
8 $ns trace-all $tr
9 set nf [open out.nam w]
10 $ns namtrace-all $nf
11
12 #define a 'finish' procedure
13 proc finish {} {
14 global ns nf
15 $ns flush-trace
16 #close the trace file
17 close $nf
18 #execute nam on the trace file
19 exec nam out.nam &
20 exit 0
21 }
22
23 #create nodes
24 set n0 [$ns node]
25 set n1 [$ns node]
26 set n2 [$ns node]
27 set n3 [$ns node]
28 set n4 [$ns node]
29
30 #create links between the nodes
31 $ns duplex-link $n0 $n1 1Mb 10ms DropTail
32 $ns duplex-link $n0 $n2 1Mb 10ms DropTail
33 $ns duplex-link $n0 $n3 1Mb 10ms DropTail
34 $ns duplex-link $n0 $n4 1Mb 10ms SFO
35 $ns duplex-link-op $n0 $n1 orient right-up
36 $ns duplex-link-op $n0 $n2 orient right-down
37 $ns duplex-link-op $n0 $n3 orient left-down
38 $ns duplex-link-op $n0 $n4 orient left-up
39
40 #run the simulation
41 $ns run
```

# ANIRUDH VADERA

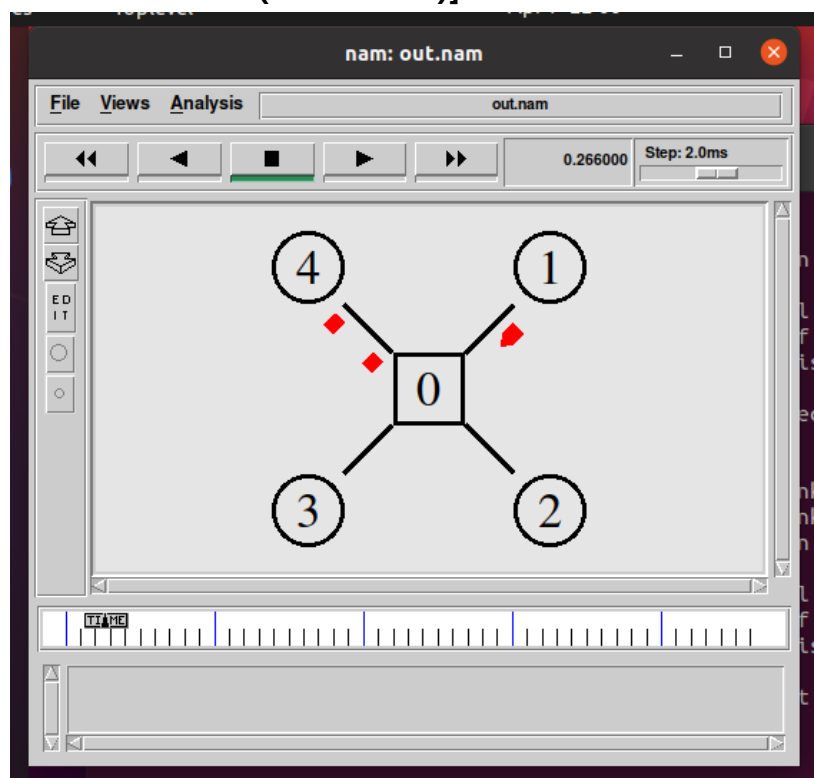
## IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2 AND IPV4 ADDRESSING

```
39
40 #Create a UDP agent and attach it to sources
41 set udp1 [new Agent/UDP]
42 Sudp1 set class 1
43 Sns attach-agent Sns1 Sudp1
44 set udp2 [new Agent/UDP]
45 Sudp2 set class 2
46 Sns attach-agent Sns2 Sudp2
47 set udp3 [new Agent/UDP]
48 Sudp3 set class 3
49 Sns attach-agent Sns3 Sudp3
50
51 Sns color 1 red
52 Sns color 2 green
53 Sns color 3 blue
54
55 #create a UDP sink agent (a traffic sink) for UDP and attach it to node n4
56 set null0 [new Agent/null]
57 Sns attach-agent Sns4 Snull0
58 #connect the traffic sources with the traffic sink
59 Sns connect Sudp1 Snull0
60 Sns connect Sudp2 Snull0
61 Sns connect Sudp3 Snull0
62
63
64 Sns shape square
65
66 # Create a CBR traffic source and attach it to tcp0
67 set cbr1 [new Application/Traffic/CBR]
68 Sscr1 set packetSize 500
69 Sscr1 set interval 0.01
70 Sscr1 attach-agent Sscr1
71 set cbr2 [new Application/Traffic/CBR]
72 Sscr2 set packetSize 500
73 Sscr2 set interval 0.01
74 Sscr2 attach-agent Sscr2
75 set cbr3 [new Application/Traffic/CBR]
76 Sscr3 set packetSize 500
77 Sscr3 set interval 0.01
78 Sscr3 attach-agent Sscr3
79
80
81 #Schedule events for the CBR agents
82 Sns at 0.2 "Sscr1 start"
83 Sns at 0.4 "Sscr2 start"
84 Sns at 0.6 "Sscr3 start"
85 Sns at 4.2 "Sscr3 stop"
86 Sns at 4.4 "Sscr2 stop"
87 Sns at 4.6 "Sscr1 stop"
88 #call the finish procedure after 5 seconds of simulation time
89 Sns at 5.0 "finish"
90
91 #simulate node failure/restoration
92 Sns rtmodel-at 1.0 down Sns2
93 Sns rtmodel-at 2.5 up Sns2
94 #Node restoration
95
96 #Simulating link failure
97 Sns rtmodel-at 1.0 down Sns1
98 Sns rtmodel-at 2.0 up Sns1
99
100 #run the simulation
101 Sns run
```

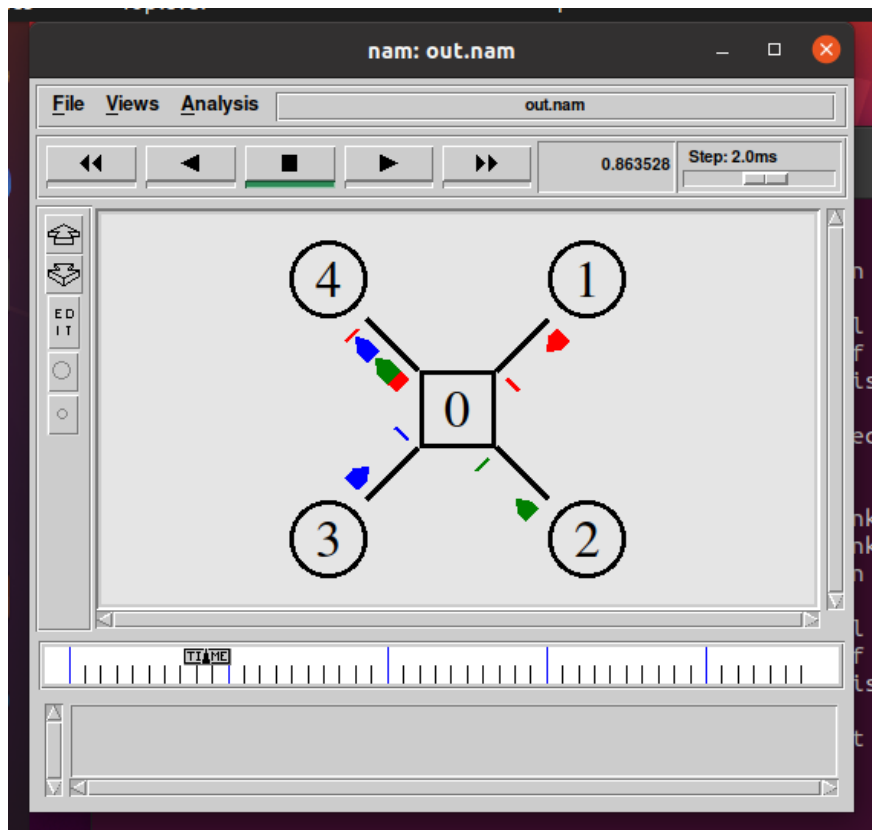
**OUTPUT (NAM SCREENSHOTS):**

**BEFORE LINK BREAKAGE:**

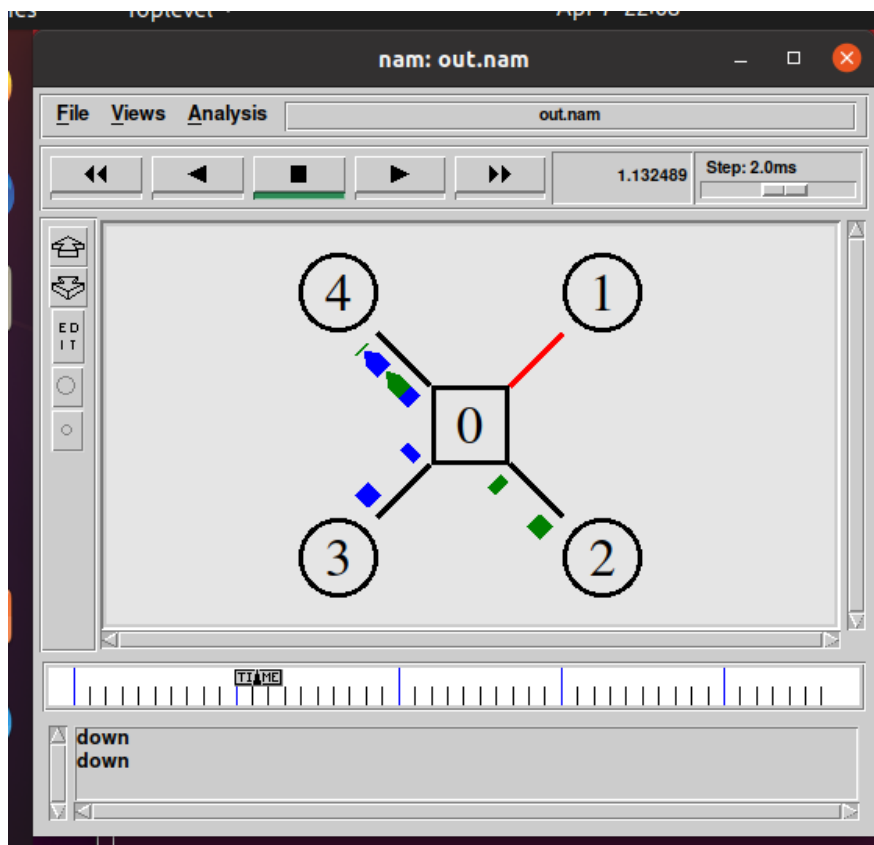
**Initially [ANIRUDH VADERA(20BCE2940)]**



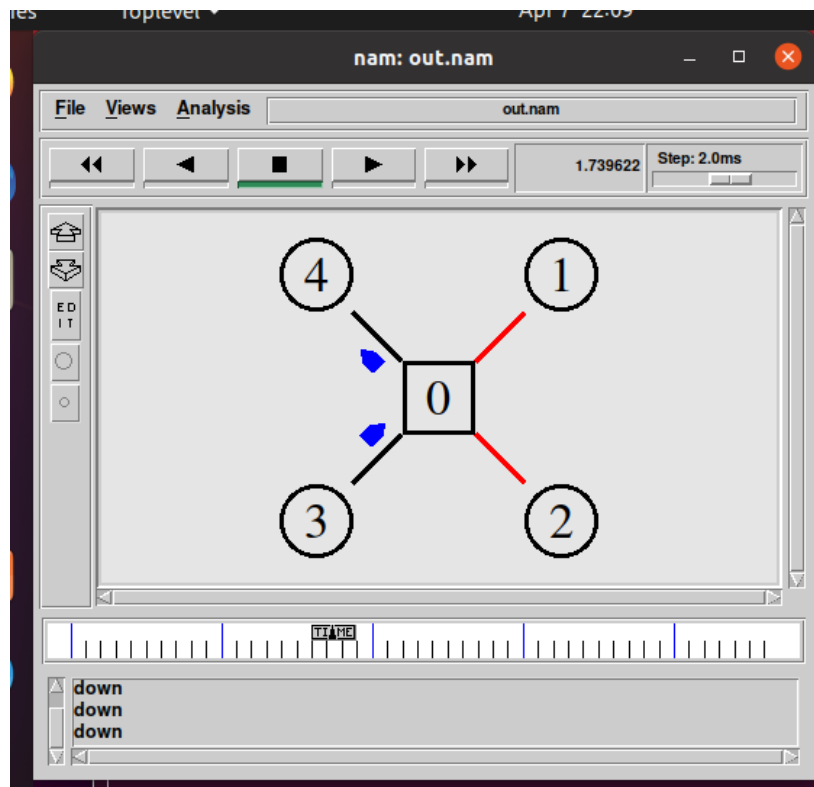
After few seconds



Link Down

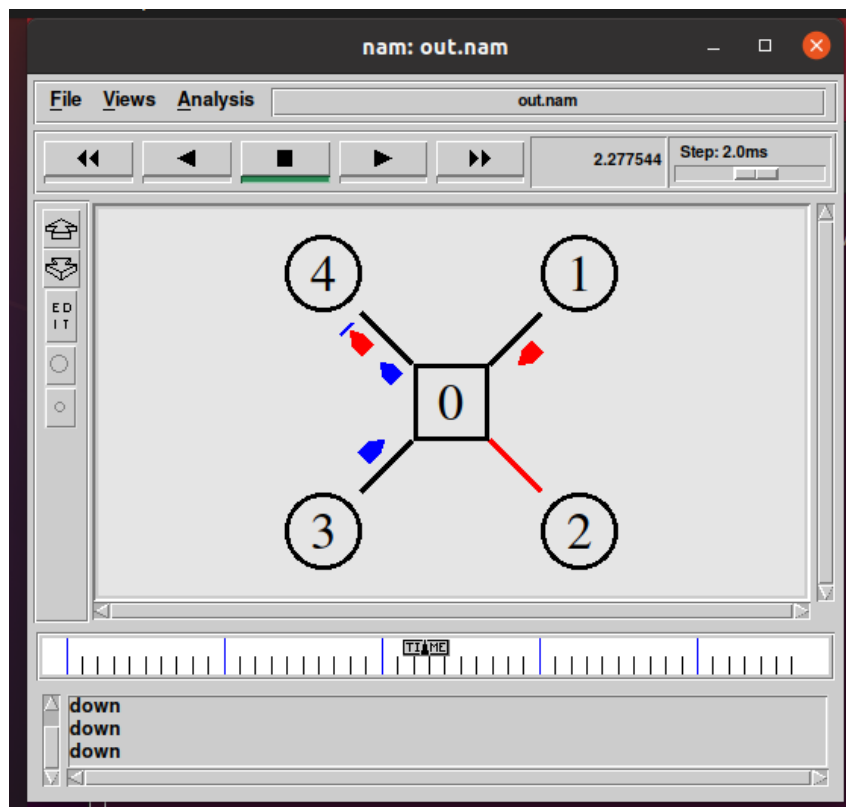


## Node Down

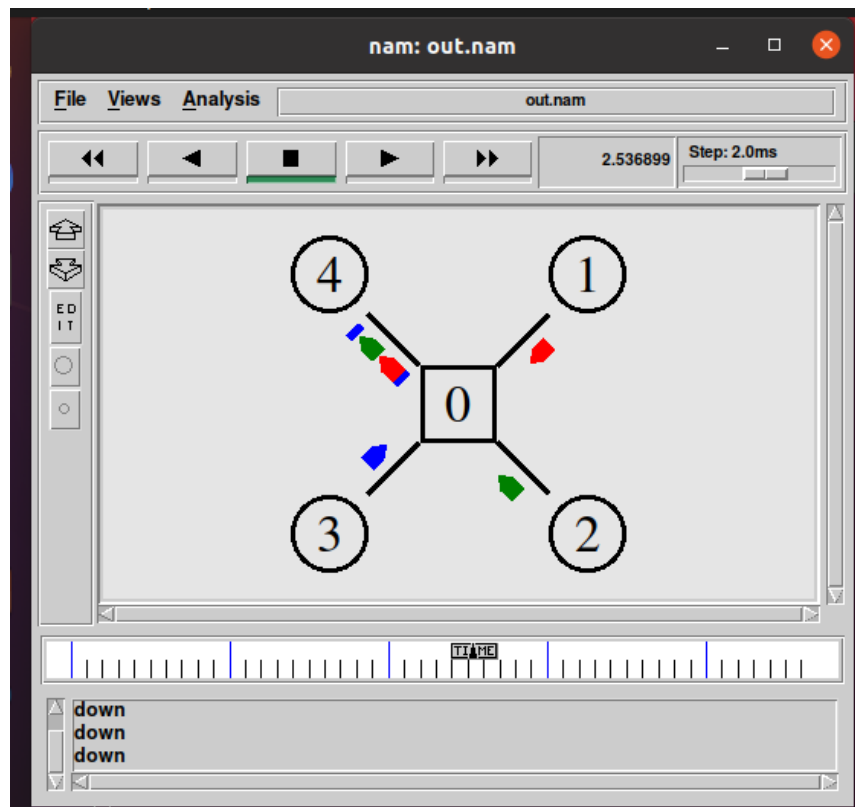


## AFTER LINK BREAKAGE:

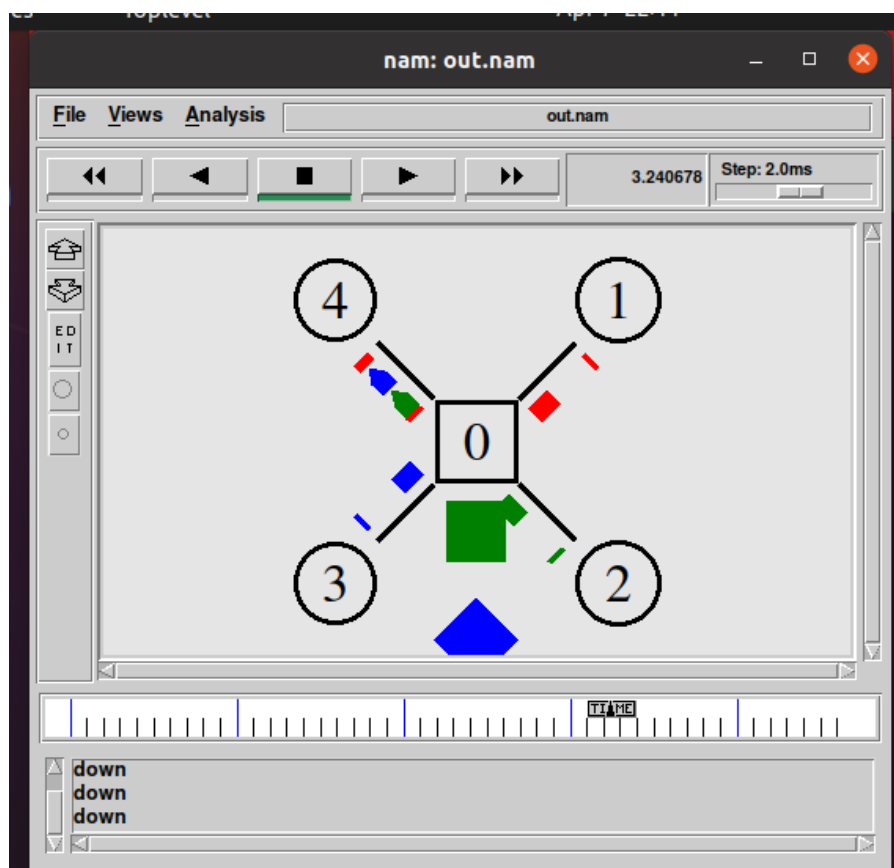
### Link up



## Node Up



## Finally Packet loss



## TRACE FILE:

AS THE TRACE FILE IS VERY LONG WE ARE ONLY ATTACHING A PART OF IT:

AT STARTING:

link_state.ns													
1	+	0.00017	0	1	rtProtoLS	100	-----	0	0.1	1.2	-1	0	
2	-	0.00017	0	1	rtProtoLS	100	-----	0	0.1	1.2	-1	0	
3	+	0.00017	0	2	rtProtoLS	100	-----	0	0.1	2.2	-1	1	
4	-	0.00017	0	2	rtProtoLS	100	-----	0	0.1	2.2	-1	1	
5	+	0.00017	0	3	rtProtoLS	100	-----	0	0.1	3.2	-1	2	
6	-	0.00017	0	3	rtProtoLS	100	-----	0	0.1	3.2	-1	2	
7	+	0.00017	0	4	rtProtoLS	100	-----	0	0.1	4.2	-1	3	
8	-	0.00017	0	4	rtProtoLS	100	-----	0	0.1	4.2	-1	3	
9	+	0.007102	2	0	rtProtoLS	100	-----	0	2.2	0.1	-1	4	
10	-	0.007102	2	0	rtProtoLS	100	-----	0	2.2	0.1	-1	4	
11	r	0.01097	0	1	rtProtoLS	100	-----	0	0.1	1.2	-1	0	
12	+	0.01097	1	0	rtProtoLS	20	-----	0	1.2	0.1	-1	5	
13	-	0.01097	1	0	rtProtoLS	20	-----	0	1.2	0.1	-1	5	
14	r	0.01097	0	2	rtProtoLS	100	-----	0	0.1	2.2	-1	1	
15	+	0.01097	2	0	rtProtoLS	20	-----	0	2.2	0.1	-1	6	
16	-	0.01097	2	0	rtProtoLS	20	-----	0	2.2	0.1	-1	6	
17	r	0.01097	0	3	rtProtoLS	100	-----	0	0.1	3.2	-1	2	
18	+	0.01097	3	0	rtProtoLS	20	-----	0	3.2	0.1	-1	7	
19	-	0.01097	3	0	rtProtoLS	20	-----	0	3.2	0.1	-1	7	
20	r	0.01097	0	4	rtProtoLS	100	-----	0	0.1	4.2	-1	3	
21	+	0.01097	4	0	rtProtoLS	20	-----	0	4.2	0.1	-1	8	
22	-	0.01097	4	0	rtProtoLS	20	-----	0	4.2	0.1	-1	8	
23	r	0.017902	2	0	rtProtoLS	100	-----	0	2.2	0.1	-1	4	
24	+	0.017902	0	2	rtProtoLS	20	-----	0	0.1	2.2	-1	9	
25	-	0.017902	0	2	rtProtoLS	20	-----	0	0.1	2.2	-1	9	
26	+	0.017902	0	1	rtProtoLS	100	-----	0	0.1	1.2	-1	10	
27	-	0.017902	0	1	rtProtoLS	100	-----	0	0.1	1.2	-1	10	
28	+	0.017902	0	3	rtProtoLS	100	-----	0	0.1	3.2	-1	11	
29	-	0.017902	0	3	rtProtoLS	100	-----	0	0.1	3.2	-1	11	
30	+	0.017902	0	4	rtProtoLS	100	-----	0	0.1	4.2	-1	12	
31	-	0.017902	0	4	rtProtoLS	100	-----	0	0.1	4.2	-1	12	
32	r	0.02113	1	0	rtProtoLS	20	-----	0	1.2	0.1	-1	5	
33	r	0.02113	2	0	rtProtoLS	20	-----	0	2.2	0.1	-1	6	
34	r	0.02113	3	0	rtProtoLS	20	-----	0	3.2	0.1	-1	7	
35	r	0.02113	4	0	rtProtoLS	20	-----	0	4.2	0.1	-1	8	
36	r	0.028062	0	2	rtProtoLS	20	-----	0	0.1	2.2	-1	9	
37	r	0.028702	0	1	rtProtoLS	100	-----	0	0.1	1.2	-1	10	
38	+	0.028702	1	0	rtProtoLS	20	-----	0	1.2	0.1	-1	13	
39	-	0.028702	1	0	rtProtoLS	20	-----	0	1.2	0.1	-1	13	
40	r	0.028702	0	3	rtProtoLS	100	-----	0	0.1	3.2	-1	11	
41	+	0.028702	3	0	rtProtoLS	20	-----	0	3.2	0.1	-1	14	
42	-	0.028702	3	0	rtProtoLS	20	-----	0	3.2	0.1	-1	14	
43	r	0.028702	0	4	rtProtoLS	100	-----	0	0.1	4.2	-1	12	
44	+	0.028702	4	0	rtProtoLS	20	-----	0	4.2	0.1	-1	15	
45	-	0.028702	4	0	rtProtoLS	20	-----	0	4.2	0.1	-1	15	



## AT END:

Ubuntu 64-bit 20.04 LTS - VMware Workstation 16 Player (Non-commercial use only)

Player ▾ || ▾ □ □

Activities Text Editor ▾

Open ▾ +

link\_state.ns

```
6232 - 4.54 1 0 cbr 500 ----- 1 1.0 4.0 434 1282
6233 r 4.544 1 0 cbr 500 ----- 1 1.0 4.0 433 1281
6234 + 4.544 0 4 cbr 500 ----- 1 1.0 4.0 433 1281
6235 - 4.544 0 4 cbr 500 ----- 1 1.0 4.0 433 1281
6236 r 4.548 0 4 cbr 500 ----- 1 1.0 4.0 432 1280
6237 + 4.55 1 0 cbr 500 ----- 1 1.0 4.0 435 1283
6238 - 4.55 1 0 cbr 500 ----- 1 1.0 4.0 435 1283
6239 r 4.554 1 0 cbr 500 ----- 1 1.0 4.0 434 1282
6240 + 4.554 0 4 cbr 500 ----- 1 1.0 4.0 434 1282
6241 - 4.554 0 4 cbr 500 ----- 1 1.0 4.0 434 1282
6242 r 4.558 0 4 cbr 500 ----- 1 1.0 4.0 433 1281
6243 + 4.56 1 0 cbr 500 ----- 1 1.0 4.0 436 1284
6244 - 4.56 1 0 cbr 500 ----- 1 1.0 4.0 436 1284
6245 r 4.564 1 0 cbr 500 ----- 1 1.0 4.0 435 1283
6246 + 4.564 0 4 cbr 500 ----- 1 1.0 4.0 435 1283
6247 - 4.564 0 4 cbr 500 ----- 1 1.0 4.0 435 1283
6248 r 4.568 0 4 cbr 500 ----- 1 1.0 4.0 434 1282
6249 + 4.57 1 0 cbr 500 ----- 1 1.0 4.0 437 1285
6250 - 4.57 1 0 cbr 500 ----- 1 1.0 4.0 437 1285
6251 r 4.574 1 0 cbr 500 ----- 1 1.0 4.0 436 1284
6252 + 4.574 0 4 cbr 500 ----- 1 1.0 4.0 436 1284
6253 - 4.574 0 4 cbr 500 ----- 1 1.0 4.0 436 1284
6254 r 4.578 0 4 cbr 500 ----- 1 1.0 4.0 435 1283
6255 + 4.58 1 0 cbr 500 ----- 1 1.0 4.0 438 1286
6256 - 4.58 1 0 cbr 500 ----- 1 1.0 4.0 438 1286
6257 r 4.584 1 0 cbr 500 ----- 1 1.0 4.0 437 1285
6258 + 4.584 0 4 cbr 500 ----- 1 1.0 4.0 437 1285
6259 - 4.584 0 4 cbr 500 ----- 1 1.0 4.0 437 1285
6260 r 4.588 0 4 cbr 500 ----- 1 1.0 4.0 436 1284
6261 + 4.59 1 0 cbr 500 ----- 1 1.0 4.0 439 1287
6262 - 4.59 1 0 cbr 500 ----- 1 1.0 4.0 439 1287
6263 r 4.594 1 0 cbr 500 ----- 1 1.0 4.0 438 1286
6264 + 4.594 0 4 cbr 500 ----- 1 1.0 4.0 438 1286
6265 - 4.594 0 4 cbr 500 ----- 1 1.0 4.0 438 1286
6266 r 4.598 0 4 cbr 500 ----- 1 1.0 4.0 437 1285
6267 + 4.6 1 0 cbr 500 ----- 1 1.0 4.0 440 1288
6268 - 4.6 1 0 cbr 500 ----- 1 1.0 4.0 440 1288
6269 r 4.604 1 0 cbr 500 ----- 1 1.0 4.0 439 1287
6270 + 4.604 0 4 cbr 500 ----- 1 1.0 4.0 439 1287
6271 - 4.604 0 4 cbr 500 ----- 1 1.0 4.0 439 1287
6272 r 4.608 0 4 cbr 500 ----- 1 1.0 4.0 438 1286
6273 r 4.614 1 0 cbr 500 ----- 1 1.0 4.0 440 1288
6274 + 4.614 0 4 cbr 500 ----- 1 1.0 4.0 440 1288
6275 - 4.614 0 4 cbr 500 ----- 1 1.0 4.0 440 1288
6276 r 4.618 0 4 cbr 500 ----- 1 1.0 4.0 439 1287
6277 r 4.628 0 4 cbr 500 ----- 1 1.0 4.0 440 1288
```

## **(IPV4 ADDRESSING)DESCRIPTION:**

### **Class of an IP Address**

#### **Network Address and Mask:**

**Network address** – It identifies a network on internet. Using this, we can find range of addresses in the network and total possible number of hosts in the network.

**Mask** – It is a 32-bit binary number that gives the network address in the address block when AND operation is bitwise applied on the mask and any IP address of the block.

The default mask in different classes are:

Class A – 255.0.0.0

Class B – 255.255.0.0

Class C – 255.255.255.0

**Example: Given IP address 132.6.17.85 and default class B mask, find the beginning address (network address).**

**Solution:** The default mask is 255.255.0.0, which means that the only the first 2 bytes are preserved and the other 2 bytes are set to 0. Therefore, the network address is 132.6.0.0.

**Subnetting:** Dividing a large block of addresses into several contiguous sub-blocks and assigning these sub-blocks to different smaller networks is called subnetting. It is a practice that is widely used when classless addressing is done.

### **Classless Addressing:**

To reduce the wastage of IP addresses in a block, we use sub-netting. What we do is that we use host id bits as net id bits of a classful IP address. We give the IP address and define the number of bits for mask along with it (usually

followed by a '/' symbol), like, 192.168.1.1/28. Here, subnet mask is found by putting the given number of bits out of 32 as 1, like, in the given address, we need to put 28 out of 32 bits as 1 and the rest as 0, and so, the subnet mask would be 255.255.255.240.

Some values calculated in subnetting:

1. **Number of subnets: Given bits for mask – No. of bits in default mask**
2. **Subnet address: AND result of subnet mask and the given IP address**
3. **Broadcast address: By putting the host bits as 1 and retaining the network bits as in the IP address**
4. **Number of hosts per subnet:  $2(32 - \text{Given bits for mask}) - 2$**
5. **First Host ID: Subnet address + 1 (adding one to the binary representation of the subnet address)**
6. **Last Host ID: Subnet address + Number of Hosts**

**Example: Given IP Address – 172.16.0.0/25, find the number of subnets and the number of hosts per subnet. Also, for the first subnet block, find the subnet address, first host ID, last host ID and broadcast address.**

**Solution:** This is a class B address. So, no. of subnets =  $2(25-16) = 2^9 = 512$ .

No. of hosts per subnet =  $2(32-25) - 2 = 2^7 - 2 = 128 - 2 = 126$

For the first subnet block, we have subnet address = 0.0, first host id = 0.1, last host id =

0.126 and broadcast address = 0.127

## AIM:

**Validate the class of an IP Addressing Schemes using Python/JAVA Compiler.**

## SOURCE CODE:

```
ip=input("Input the IPv4 Address which you want to validate : ")
l=ip.split('.')
temp=l[-1].split('/')
l[-1]=temp[0]
l.append(temp[1])
b=[]
for i in range(len(l)):
    l[i]=int(l[i])
for i in range(len(l)-1):
    if l[i]>=0 and l[i]<=255:
        b.append(bin(l[i])[2:])
    else:
        print("No")
        exit()
for i in range(len(l)-1):
    while len(b[i])<8:
        b[i]='0'+b[i]
cl=""
if b[0]=='0':
    cl="A"
elif b[0][0:2]=="10":
    cl="B"
elif b[0][0:3]=="110":
    cl="C"
elif b[0][0:4]=="1110":
    cl="D"
elif b[0][0:4]=="1111":
    cl="E"
if cl=="A":
    nbdm=8
elif cl=="B":
    nbdm=16
elif cl=="C":
    nbdm=24
nsubnets=2**((l[-1]-nbdm)
nhs=(2**(32-l[-1]))-2
z=32-l[-1]
```

```

c=0
for i in range(len(l)-2, -1, -1):
    if z>=8:
        b[i]="00000000"
        z-=8
    else:
        b[i]=b[i][:8-z]
        while z>0:
            b[i]+='0'
            z-=1
sa=[]
for i in range(len(b)):
    sa.append(int(b[i], 2))
temp=""
for i in range(len(sa)):
    temp+=str(sa[i])
    if i!=len(sa)-1:
        temp+="."
temp1=""
temp2=""
for i in range(len(sa)):
    if i == len(sa)-1:
        temp1+=str(sa[i]+1)
        temp2+=str(sa[i]+nhs+1)
    else:
        temp1+=str(sa[i])
        temp2+=str(sa[i])
    if i!=len(sa)-1:
        temp1+='.'
        temp2+='.'
broad_Id = temp2
temp2 = temp2[0:-1] + str(int(temp2[-1])-1)
print("(If this was a classfull addressing)Class: ", cl)
print("1. Number of Subnets: ", nsubnets)
print("2. Subnet address: ", temp)
print("3. Broadcast Address: ", broad_Id)
print("4. Number of hosts per subnet: ", nhs)
print("5. First Host id: ", temp1)
print("6. Last Host id: ", temp2)

```

## CODE SNAPSHOTS:

```
ip_address.py X
1  ip=input("Input the IPv4 Address which you want to validate : ")
2  l=ip.split('.')
3  temp=l[-1].split('/')
4  l[-1]=temp[0]
5  l.append(temp[1])
6  b=[]
7  for i in range(len(l)):
8      l[i]=int(l[i])
9  for i in range(len(l)-1):
10     if l[i]>=0 and l[i]<=255:
11         b.append(bin(l[i])[2:])
12     else:
13         print("No")
14         exit()
15  for i in range(len(l)-1):
16     while len(b[i])<8:
17         b[i]='0'+b[i]
18  cl=''
19  if b[0]=='0':
20     cl="A"
21  elif b[0][0:2]=="10":
22     cl="B"
23  elif b[0][0:3]=="110":
24     cl="C"
25  elif b[0][0:4]=="1110":
26     cl="D"
27  elif b[0][0:4]=="1111":
28     cl="E"
29  if cl=="A":
30     nbdm=8
31  elif cl=="B":
32     nbdm=16
33  elif cl=="C":
34     nbdm=24
35  nsubnets=2**(l[-1]-nbdm)
36  nhs=(2**(32-l[-1]))-2
37  z=32-l[-1]
38  c=0
```

ANIRUDH VADERA  
IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2 AND IPV4 ADDRESSING

C:\Users\Anirudh\OneDrive\Desktop\python\ip\_address.py

```
ip_address.py X
38  c=0
39  for i in range(len(l)-2, -1, -1):
40      if z>=8:
41          b[i]="00000000"
42          z-=8
43      else:
44          b[i]=b[i][:8-z]
45          while z>0:
46              b[i]+='0'
47              z-=1
48  sa=[]
49  for i in range(len(b)):
50      sa.append(int(b[i], 2))
51  temp=""
52  for i in range(len(sa)):
53      temp+=str(sa[i])
54      if i!=len(sa)-1:
55          temp+="."
56  temp1=""
57  temp2=""
58  for i in range(len(sa)):
59      if i == len(sa)-1:
60          temp1+=str(sa[i]+1)
61          temp2+=str(sa[i]+nhs+1)
62      else:
63          temp1+=str(sa[i])
64          temp2+=str(sa[i])
65      if i!=len(sa)-1:
66          temp1+='.'
67          temp2+='.'
68  broad_Id = temp2
69  temp2 = temp2[0:-1] + str(int(temp2[-1])-1)
70  print("(If this was a classlfull addressing)Class: ", c1)
71  print("1. Number of Subnets: ", nsubnets)
72  print("2. Subnet address: ", temp)
73  print("3. Broadcast Address: ", broad_Id)
74  print("4. Number of hosts per subnet: ", nhs)
75  print("5. First Host id: ", temp1)
76  print("6. Last Host id: ", temp2)
77  print("ANIRUDH VADERA(20BCE2940)")
```

## OUTPUT:

```
In [8]: runfile('C:/Users/Anirudh/OneDrive/Desktop/python/ip_address.py', wdir='C:/Users/Anirudh/OneDrive/Desktop/python')
```

Input the IPv4 Address which you want to validate : 172.16.0.0/25

(If this was a classlfull addressing)Class: B

1. Number of Subnets: 512

2. Subnet address: 172.16.0.0

3. Broadcast Address: 172.16.0.127

4. Number of hosts per subnet: 126

5. First Host id: 172.16.0.1

6. Last Host id: 172.16.0.126

ANIRUDH VADERA(20BCE2940)