



IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2

CSE1004(NETWORK AND COMMUNICATION)LAB:L53-L54



**MARCH 24, 2022
ANIRUDH VADERA
20BCE2940**

SIMULATION AND NS2:

Simulation is the process of *learning by doing*. Whenever there is something new in the world, we try to analyze it first by examining it and in the process get to learn a lot of things. This entire course is called **Simulation**.

Correlating to this process, in order to understand all the complexities one needs to model the entire role-play in form of computer simulation, the need is to build artificial objects and assign them roles dynamically.

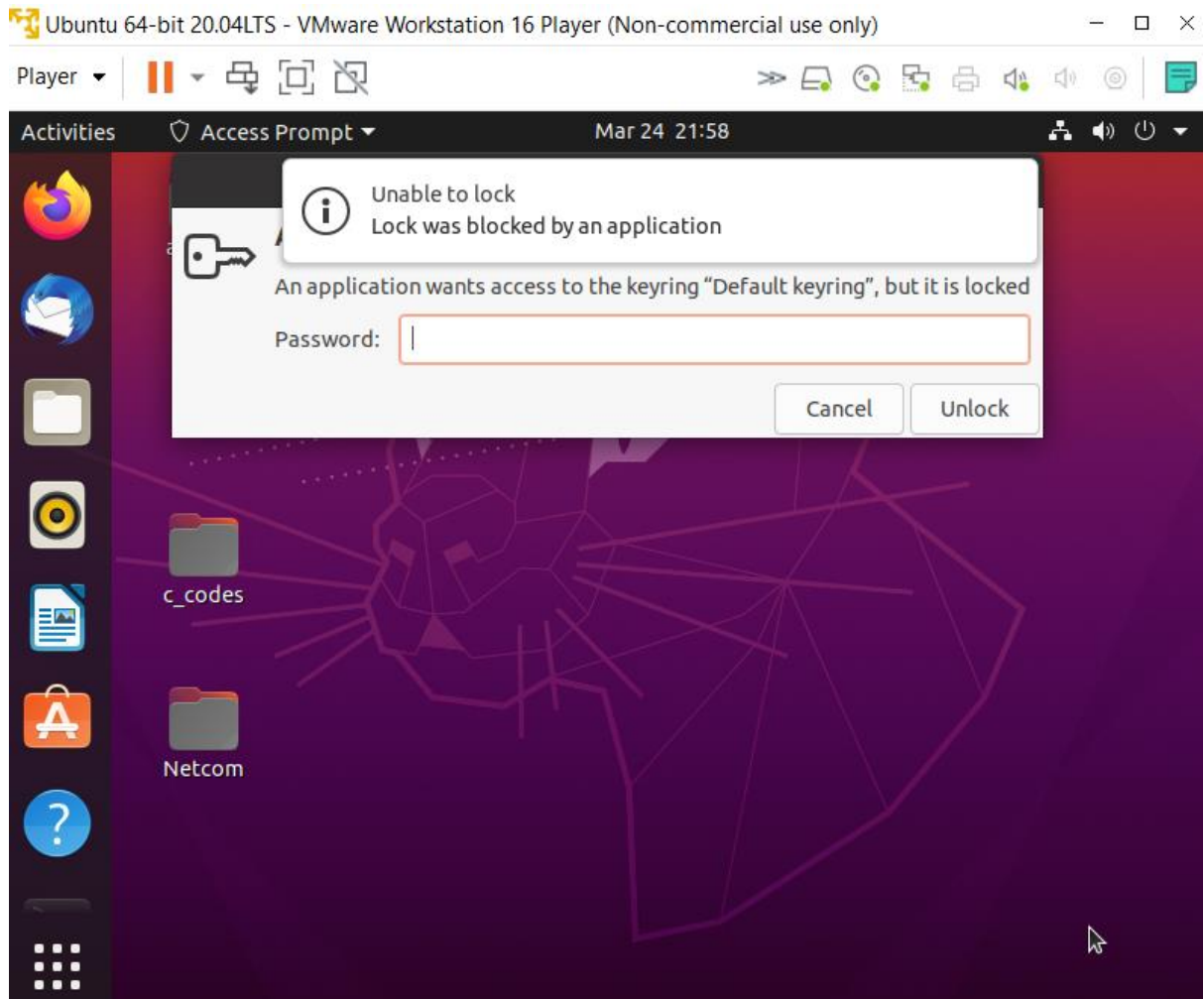
Computer simulation is the designing of a theoretical physical system on a digital computer with emphasis on model designing, execution, and analysis. After the creation of the mathematical model, the most important step is to create a computer program for updating the state and event variables through time (by time slicing or event scheduling). If this simulation is carried out successively in parallel computers, it is called *Parallel* or *Distributed simulation*.

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:

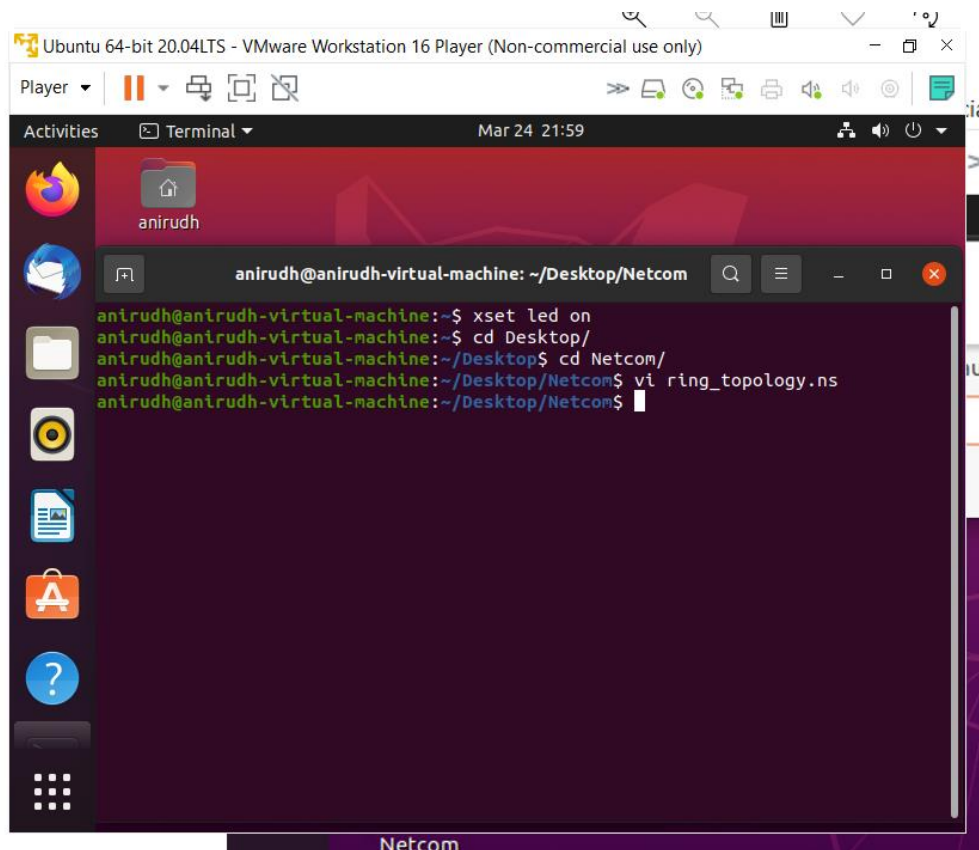
- **Open ur vmware having a Ubuntu linux distribution:**



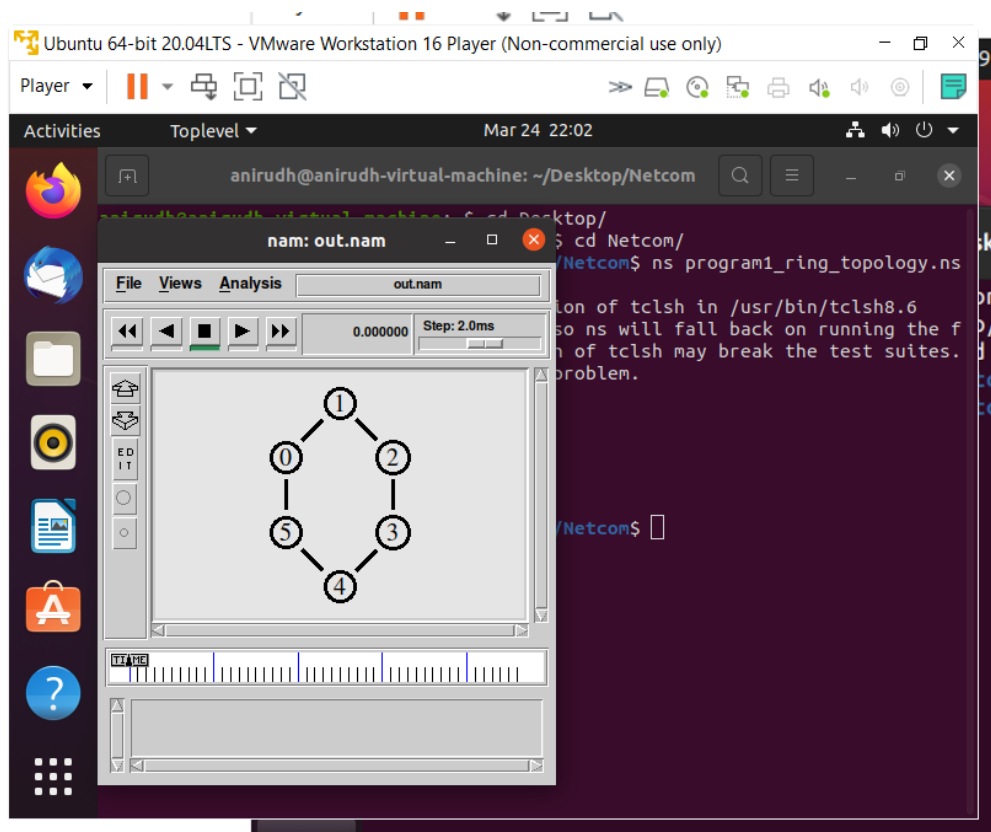
- **U must have all the necessary c and ns2 compilers already installed in your linux system.**
- **Using terminal open a vi editor and write ur ns2 code with the command ns filename.ns**

ANIRUDH VADERA

IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2



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

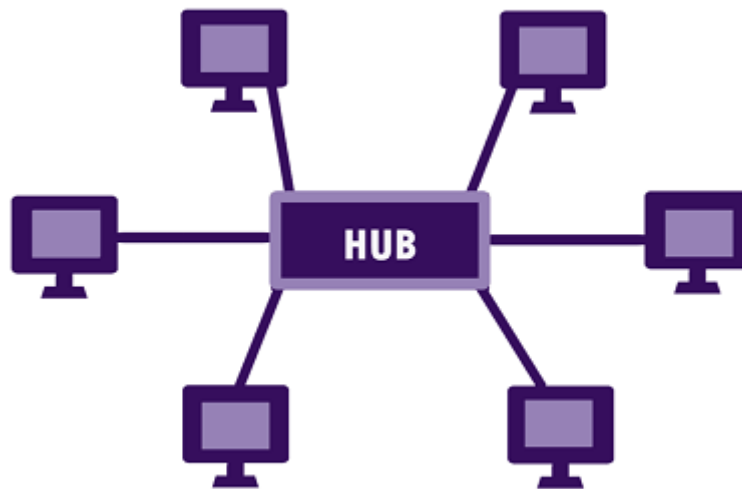


QUESTION:

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).

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

```
#Create a simulator object  
set ns [new Simulator]  
  
#Routing Protocol used is Link State  
  
$ns rtproto 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 random number of nodes, let's say 7. 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 (1.5Mb), delay (10ms) and kind of queue (DropTail).

```
#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 tcp0

```
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

\$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

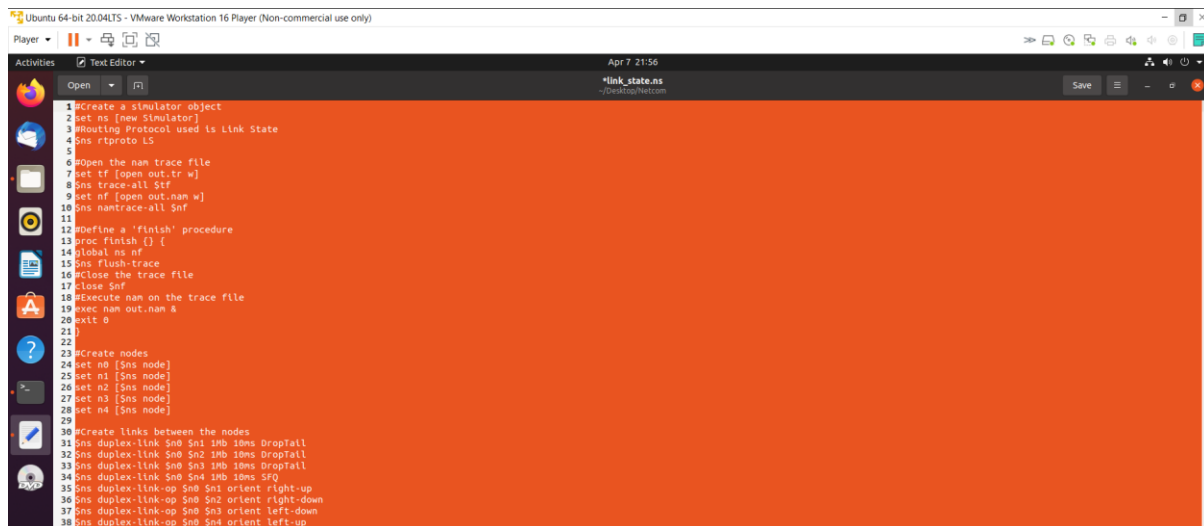
\$ns rtmodel-at 1.0 down \$n0 \$n1

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

#Run the simulation

\$ns run

CODE SNAPSHOTS:



```
1#Create a simulator object
2set ns [new Simulator]
3Routing Protocol used is Link State
4ns rtproto ls
5
6#Open the nam trace file
7set tf [open out.tr w]
8$ns trace-all $tf
9set nf [open out.nam w]
10$ns namtrace-all $nf
11
12#Define a 'finish' procedure
13proc finish {} {
14    global ns nf
15    $ns flush-trace
16    close the trace file
17    close $nf
18    #Execute nam on the trace file
19    exec nam out.nam &
20    exit 0
21}
22
23#Create nodes
24set n0 [$ns node]
25set n1 [$ns node]
26set n2 [$ns node]
27set n3 [$ns node]
28set n4 [$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 $n4 1Mb 10ms SFG
35$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
```

ANIRUDH VADERA

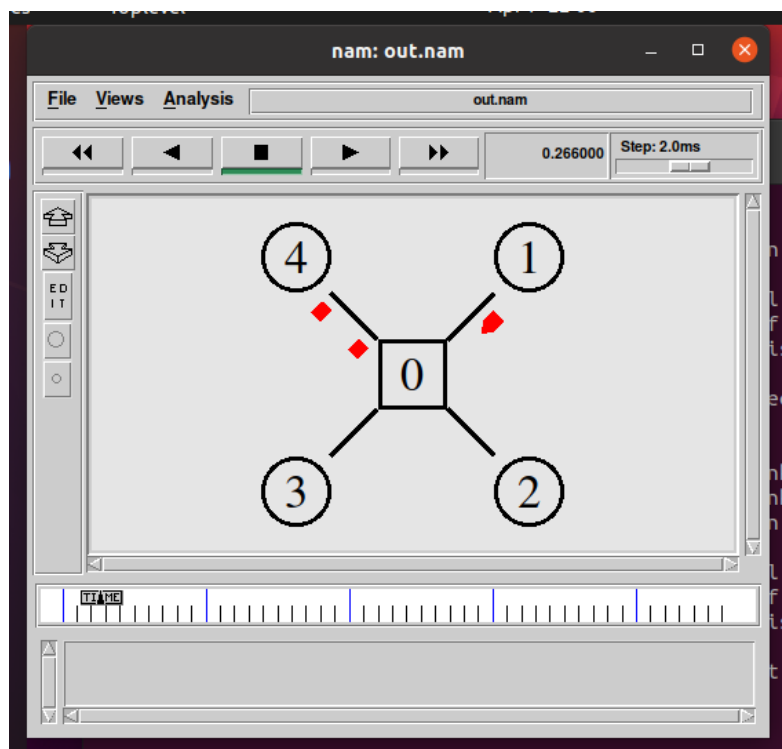
IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2

```
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 $n1 Sudp1
44 set udp2 [new Agent/UDP]
45 Sudp2 set class 2
46 Sns attach-agent $n2 Sudp2
47 set udp3 [new Agent/UDP]
48 Sudp3 set class 3
49 Sns attach-agent $n3 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 $n4 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 Sudp1
71 set cbr2 [new Application/Traffic/CBR]
72 Sscr2 set packetSize 500
73 Sscr2 set Interval 0.01
74 Sscr2 attach-agent Sudp2
75 set cbr3 [new Application/Traffic/CBR]
76 Sscr3 set packetSize 500
77 Sscr3 set Interval 0.01
78 Sscr3 attach-agent Sudp3
79
80 #Schedule events for the CBR agents
81 Sns at 0.2 "Sscr1 start"
82 Sns at 0.4 "Sscr2 start"
83 Sns at 0.6 "Sscr3 start"
84 Sns at 4.2 "Sscr3 stop"
85 Sns at 4.4 "Sscr2 stop"
86 Sns at 4.6 "Sscr1 stop"
87 #Call the finish procedure after 5 seconds of simulation time
88 Sns at 5.0 "finish"
89
90 #Simulate node failure/restoration
91 Sns rtmodel-at 1.0 down $n2
92 #Node failure
93 Sns rtmodel-at 2.5 up $n2
94 #Node restoration
95
96 #Simulating link failure
97 Sns rtmodel-at 1.0 down $n0 $n1
98 Sns rtmodel-at 2.0 up $n0 $n1
99
100 #Run the simulation
101 Sns run
```

OUTPUT (NAM SCREENSHOTS):

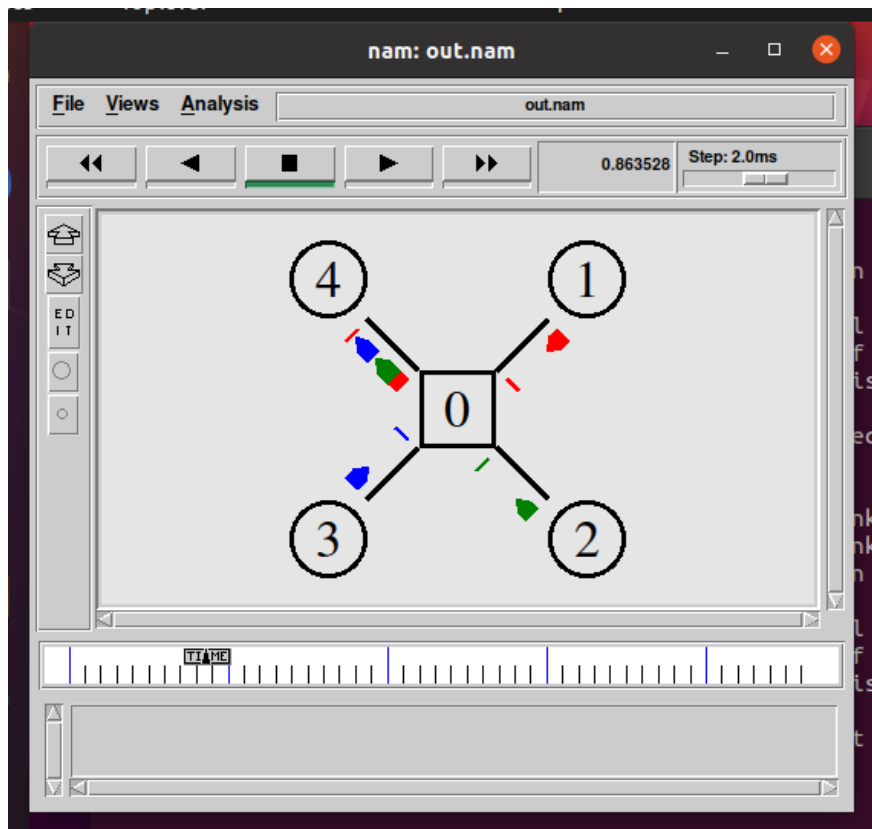
BEFORE LINK BREAKAGE:

Initially

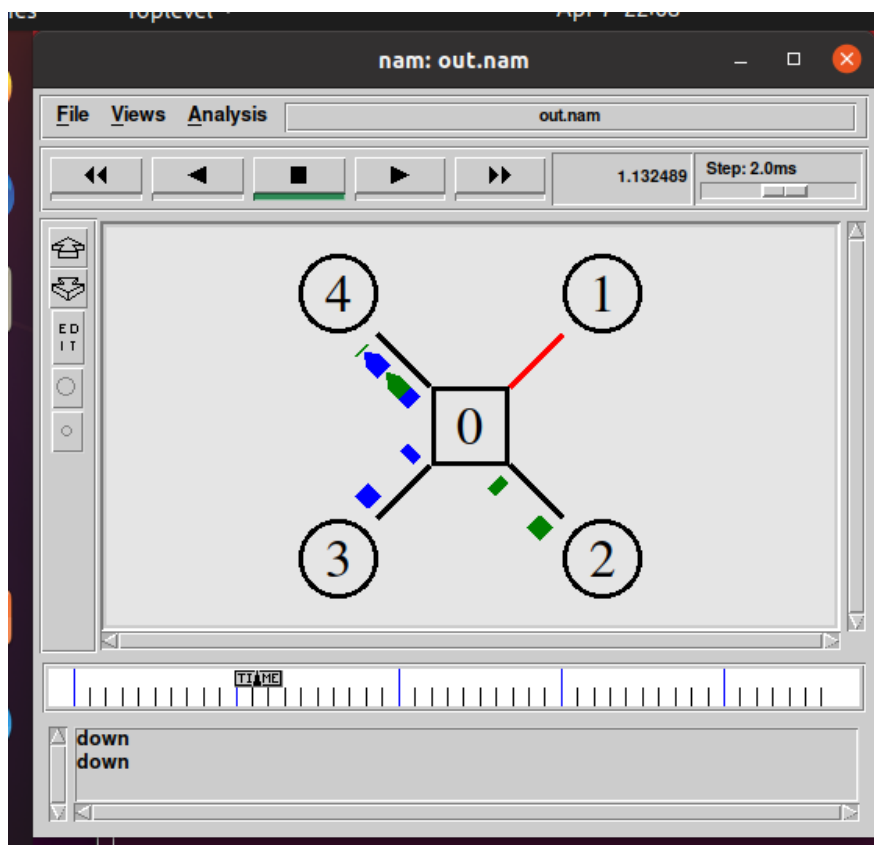


ANIRUDH VADERA
IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2

After few seconds

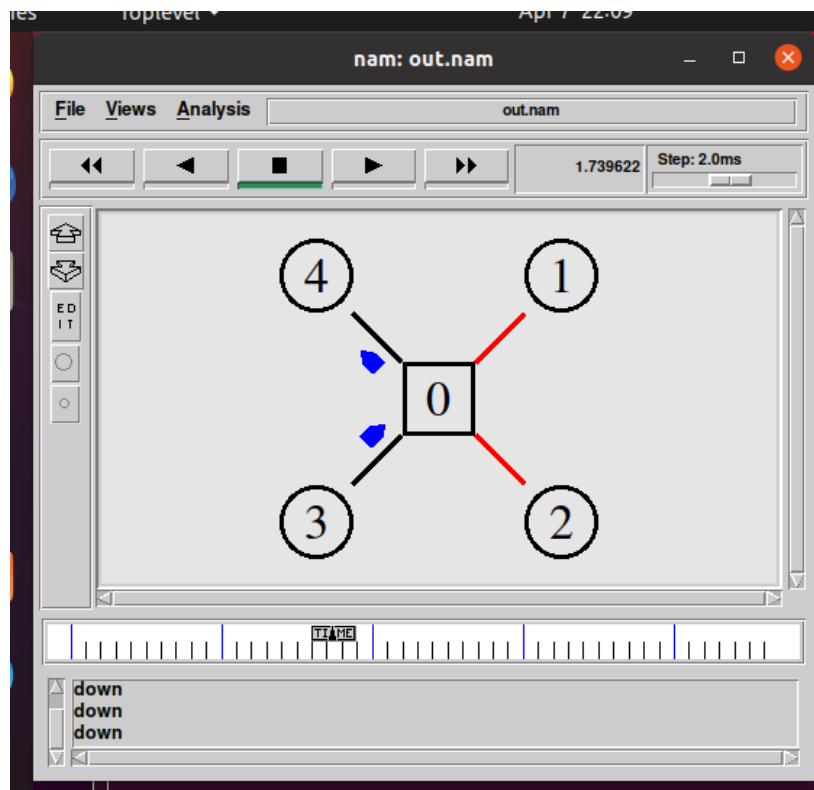


Link Down



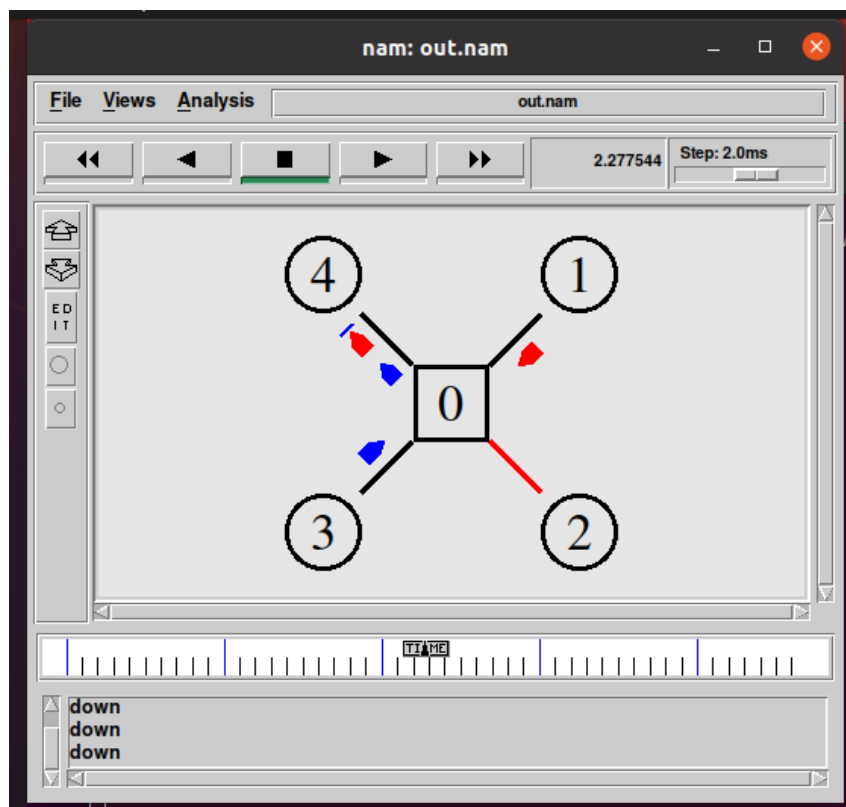
ANIRUDH VADERA
IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2

Node Down



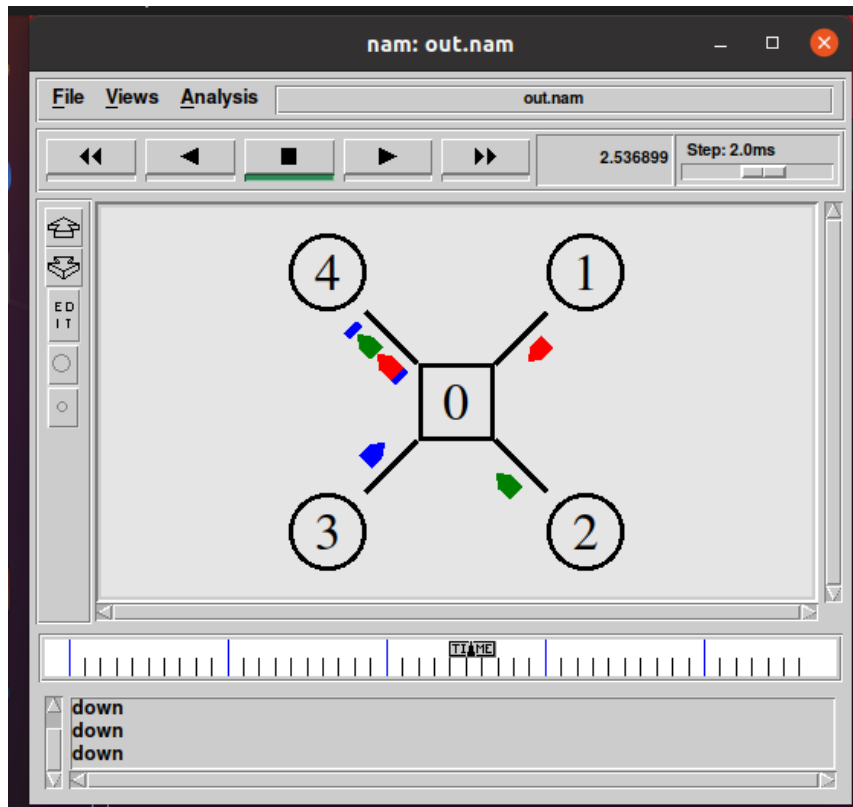
AFTER LINK BREAKAGE:

Link up

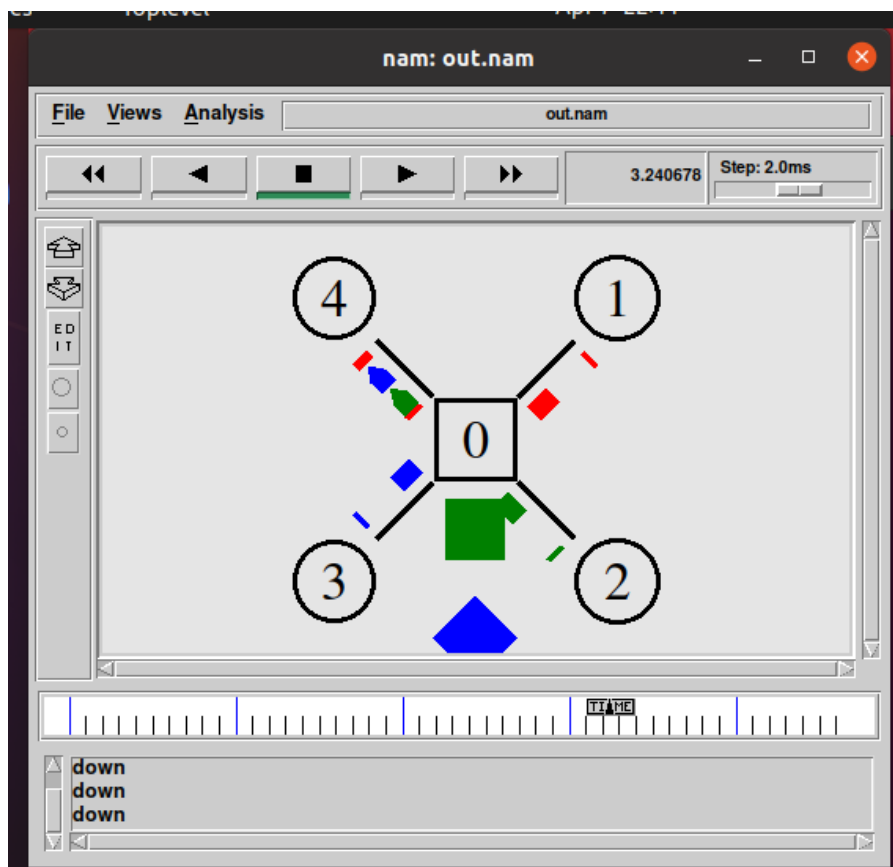


ANIRUDH VADERA
IMPLEMENTING LINK STATE ROUTING (STAR TOPOLOGY) USING NS2

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

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

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 - 4.534 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
```