RAMNIRANJAN JHUNJHUNWALA COLLEGE GHATKOPAR (W), MUMBAI - 400 086

DEPARTMENT OF INFORMATION TECHNOLOGY 2020 - 2021

M.Sc.(I.T.) SEM II
Wireless Sensor Network

Name : Sneha Pawar
Roll No. : 11

Hindi Vidya Prachar Samiti's

RAMNIRANJAN JHUNJHUNWALA COLLEGE

(AUTONOMOUS)

Opposite Ghatkopar Railway Station, Ghatkopar West, Mumbai-400086

CERTIFICATE

This is to certify that Miss. **Sneha Ramchandra Pawar** with Roll No. **11** has successfully completed the necessary course of experiments in the subject of **Wireless Sensor Networks** during the academic year **2020 – 2021** complying with the requirements of **RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE**, for the course of **M.Sc.(IT)** semester -II.

_____ _____

Internal Examiner External Examiner

_____ _____

Head of Department College Seal

INDEX

Practical no 1:

AIM : Introduction of Wireless sensor network applications and its simulation.

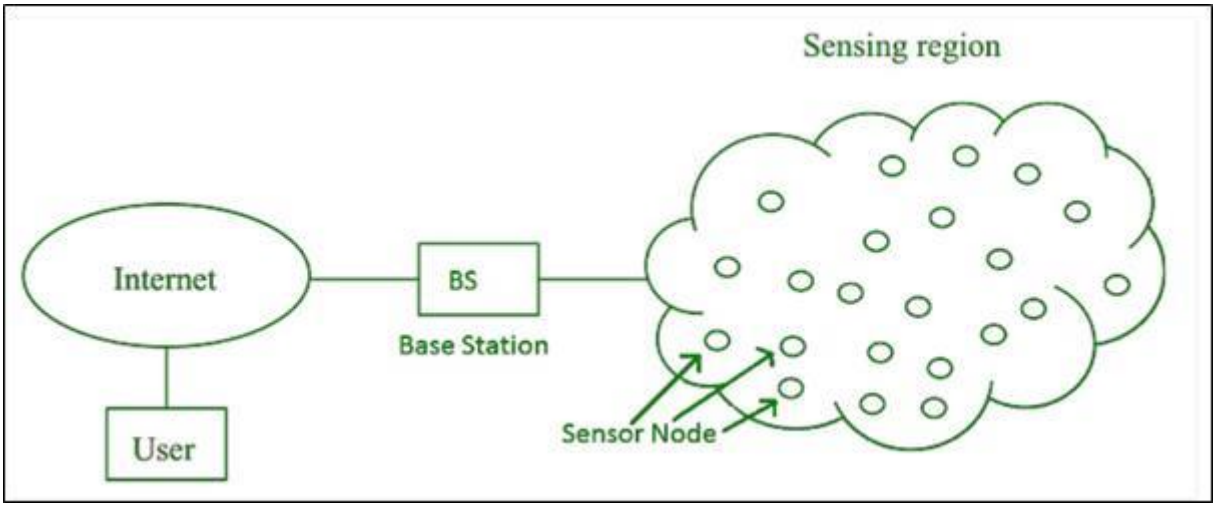
THEORY:

What are Wireless Sensor networks?

Wireless Sensor Network (WSN) is an infrastructure-less wireless network that is deployed in a large number of wireless sensors in an ad-hoc manner that is used to monitor the system, physical or environmental conditions.

Sensor nodes are used in WSN with the onboard processor that manages and monitors the environment in a particular area. They are connected to the Base Station which acts as a processing unit in the WSN System.

Base Station in a WSN System is connected through the Internet to share data. WSN can be used for processing, analysis, storage, and mining of the data.



List various applications of wireless sensor networks.

1. Disaster Relief Operation
2. Military Operation
3. Environmental Applications
4. Medical Applications
5. Home Applications
6. Industrial monitoring and control
7. Sensor and Robots
8. Highway Monitoring
9. Civil and Environmental Engineering Applications
10. Wildfire Instrumentation

Explain any 1 in detail.

Home control applications provide control, conservation, convenience, and safety, as follows:

- Sensing applications facilitate flexible management of lighting, heating, and cooling systems from anywhere in the home.
- Sensing applications automate control of multiple home systems to improve conservation, convenience, and safety.
- Sensing applications capture highly detailed electric, water, and gas utility usage data.
- Sensing applications embed intelligence to optimize consumption of natural resources.
- Sensing applications enable the installation, upgrading, and networking of a home control system without wires.
- Sensing applications enable one to configure and run multiple systems from a single remote control.
- Sensing applications support the straightforward installation of wireless sensors to monitor a wide variety of conditions.
- Sensing applications facilitate the reception of automatic notification upon detection of unusual events.
- Body-worn medical sensors (e.g., heartbeat sensors) are also emerging. These are battery-operated devices with network beacons occurring either every few seconds that could be worn by home-resident elderly or people with other medical conditions.
- These sensors have two ongoing processes: heartbeat time logging and transmission of heart rate and other information (instantaneous and average heart rate, body temperature, and battery voltage)

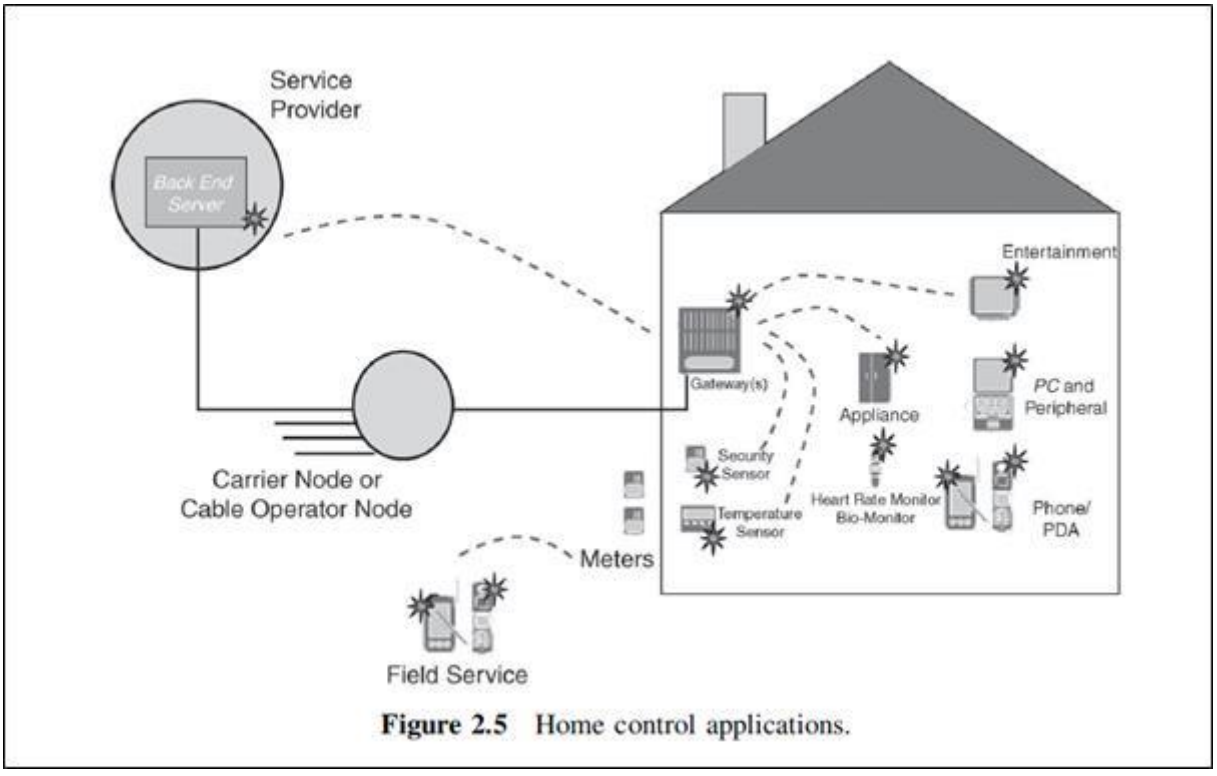


Figure 2.5 Home control applications.

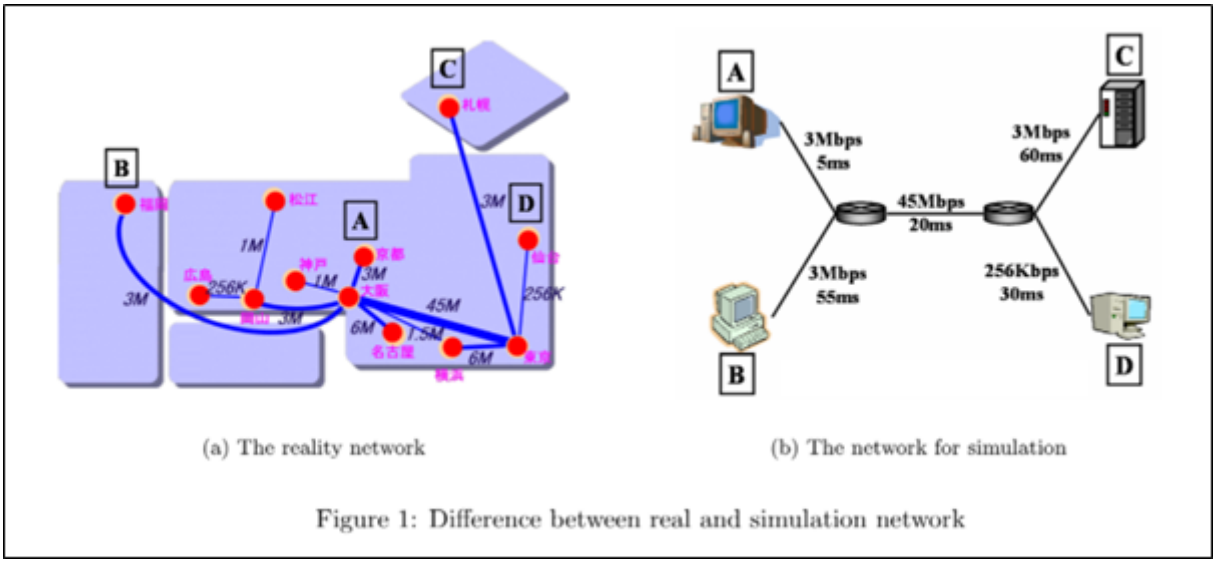
Practical 2

AIM : Network Simulator installation of wireless sensor network.

THEORY:

What is a Network Simulator?

Setting up a network to do some real experiments is the best way for studying about communication in internet. However, setting a network is not easy and costly. For this reason, a virtual network provided by network simulator is used for experiment in only one computer. Specially, NS2 which is free and easy to use is the popular all over the world.
For example, to observe the communication between A,B,C,D in a network as shown in Fig. 1(a), we can set up a network topology as shown in Fig.1(b) for simulator to do experiment.



(a) The reality network (b) The network for simulation

Figure 1: Difference between real and simulation network

Explain NS2 in detail & its use.

- NS2 use Tcl language for creating simulation scenario file (for example, sample.tcl).
- Network topology, transmission time, using protocol etc... are defined in scenario file.
- If we execute this scenario file, the simulation results will be outputed to out.tr and out.nam file.
- out.tr all the information about communication is written in this file. We can find out the way a packet was forwarded. This file is called as trace file.
- out.nam contains the data for animation of the experiment result. This file can be execute by Nam, an animation software.

To write a scenario file, we need to understand about the network inside NS2.

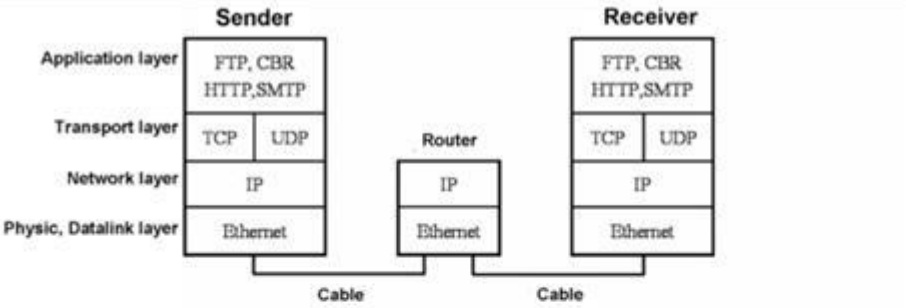


Figure 3: The realistic network

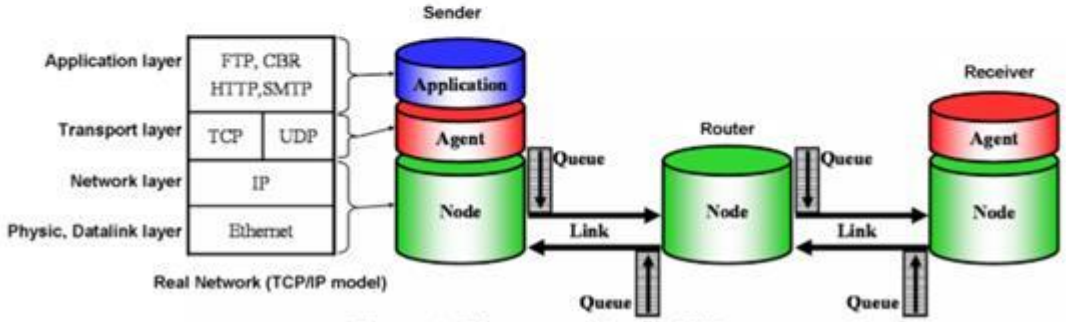


Figure 4: The network inside NS2

The node (terminal, router) in real network have 4 layer using TCP/IP model as shown in Fig. 3.

- Actually, forwarding router have only 2 bottom layer. In the otherwise, 4 layers of TCP/IP model in NS2 are shown in Fig. 4.
- Two bottom layers are ready automatically when a node setting up. TCP or UDP in transport layer are shown as Agent.
- FTP, CBR are in application layers. Actually, the sender will set the application for creating data while the receive will not because it is no need to simulate the receiving data in application layer.
- Realistic networks use cable for link between 2 node. One cable is available for biconnection. In NS2, a link is use for one connection. Two lines are needed for biconnection.
- Each link has a queue which similar to buffer in realistic network. Packets sent from node should be queuing in queue. When queue is empty, it will be send to other node via link.

The following is steps to create a scenario file. Step0 Declare Simulator and setting output file Step1 Setting Node and Link
Step2 Setting Agent
Step3 Setting Application
Step4 Setting Simulation time and schedules Step5 Declare finish.

Explain OTCL script.

- 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 UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue 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.

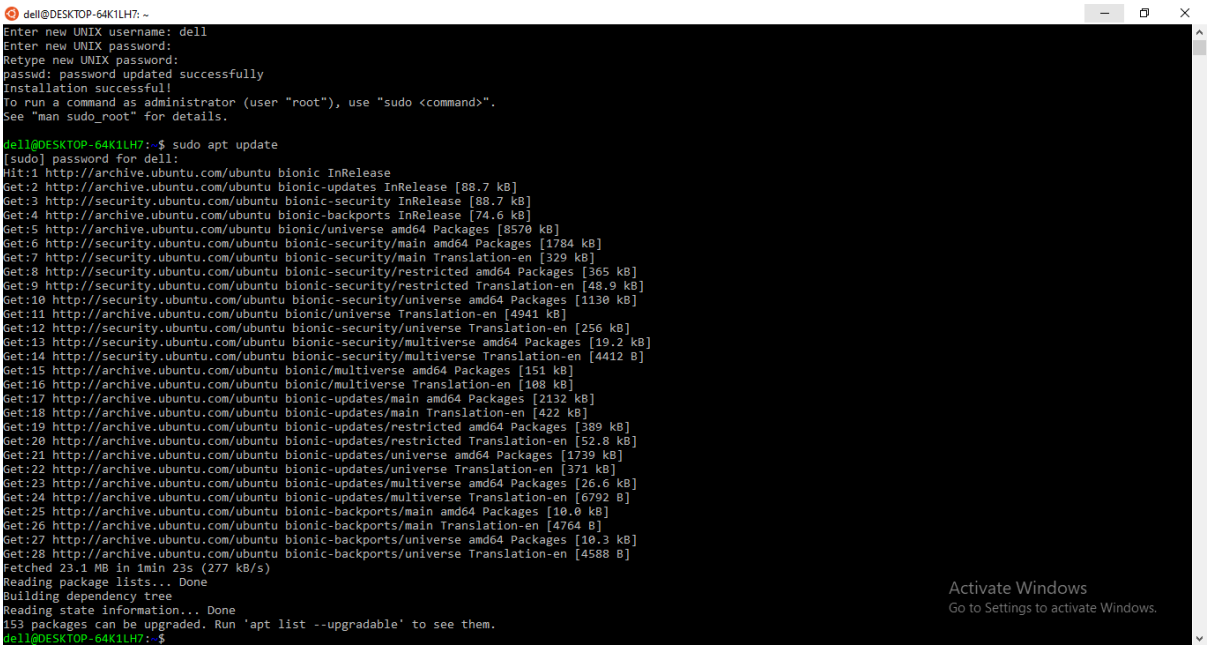
Steps (in detail) for installation with Screenshot of every step:

Pre-requisites Software Required:

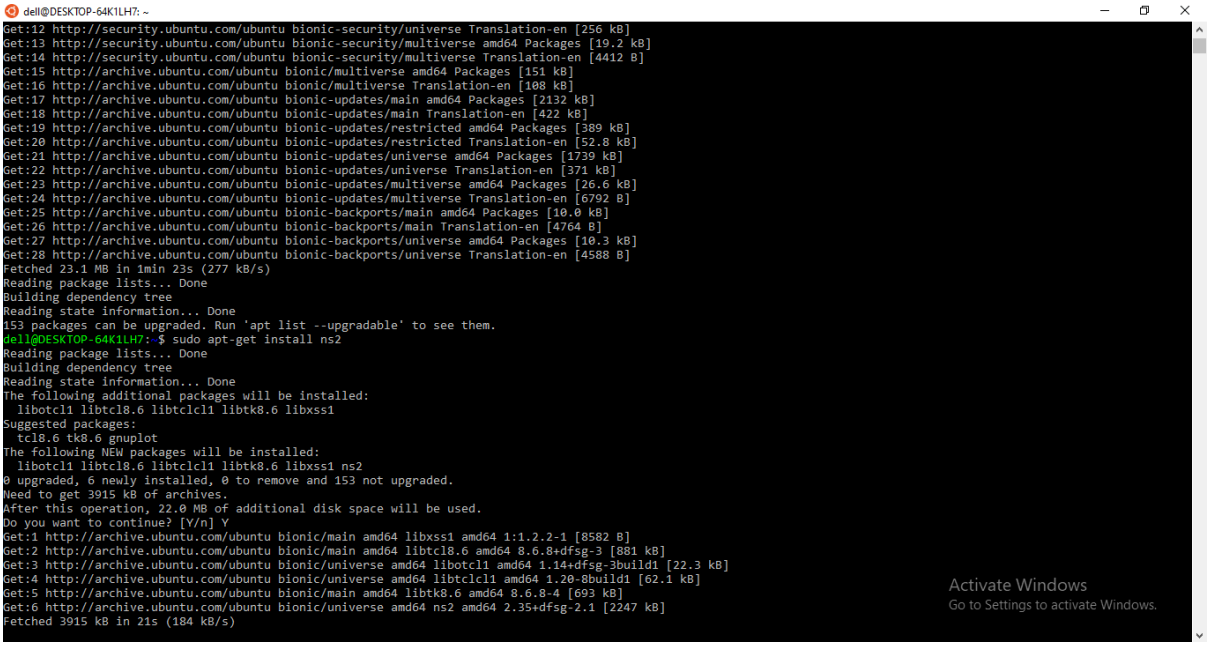
1. Ubuntu 18.04/20.04 should be installed in the system before starting installation of NS-2 Simulator.
2. Download and Install Xming

Step 1: Write following command and press enter.

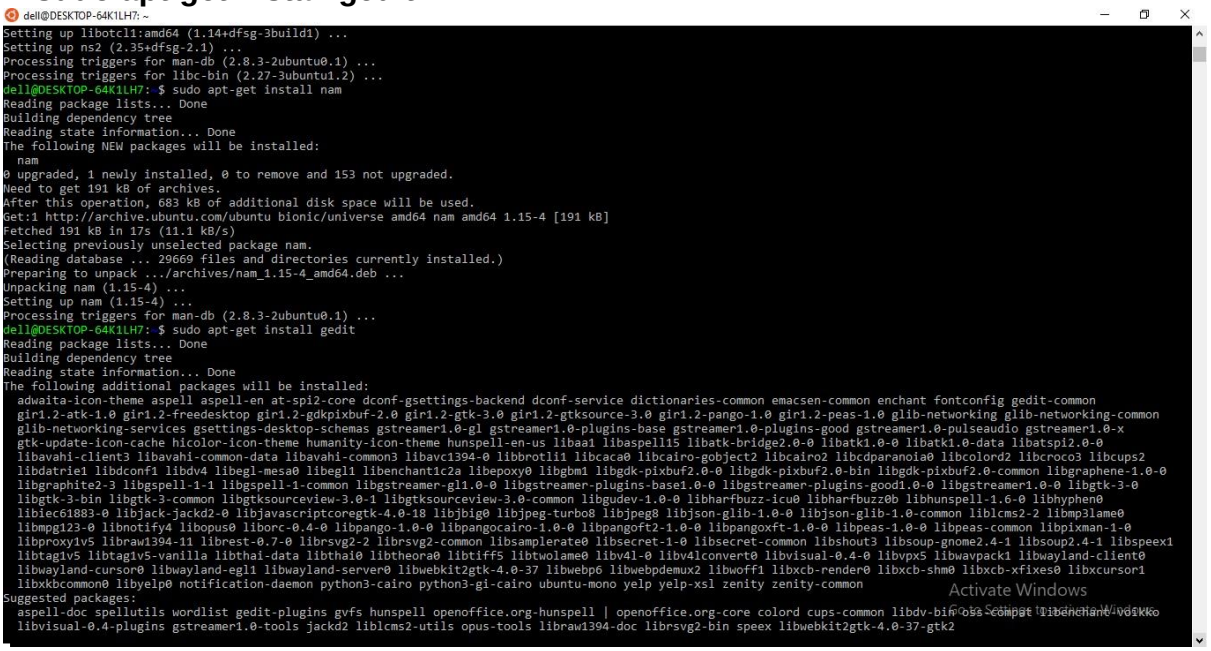
1. sudo apt update



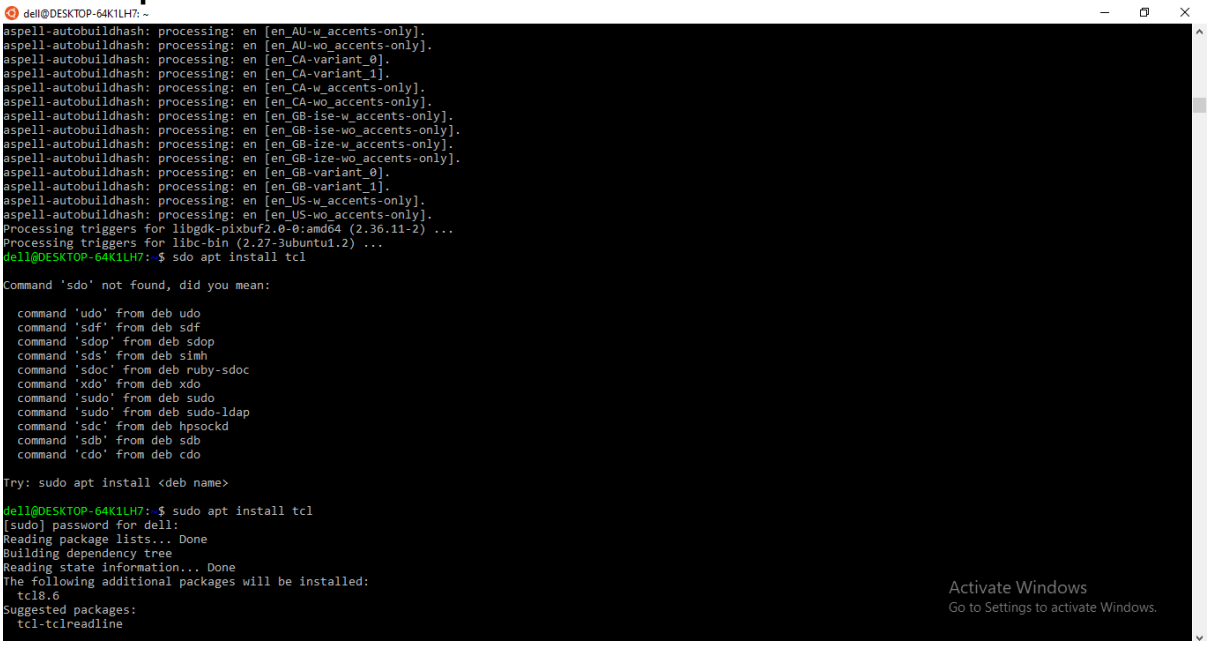
2..sudo apt-get install ns2



3. sudo apt-get install nam
4. sudo apt-get install gedit



5. sudo apt install tcl



Step 2: Type the following in Ubuntu:

- **export DISPLAY=0:0** – For display
- **gedit filename.tcl --** For creating a file
- **ns filename.tcl** -- To run the file

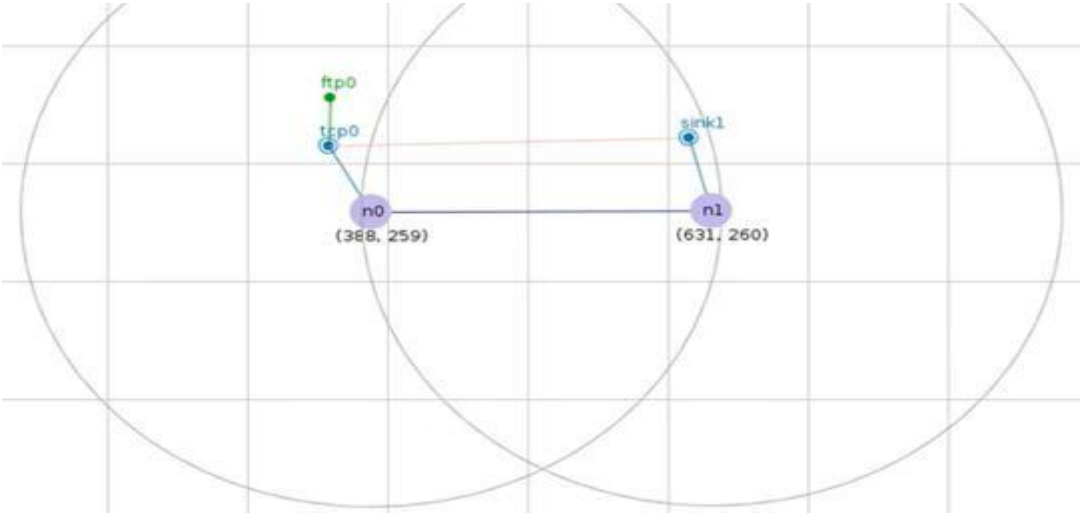
Practical 3

AIM : Write TCL script for transmission between mobile nodes.

THEORY:

A mobile node is an Internet-connected device whose location and point of attachment to the Internet may frequently be changed. This kind of node is often a cellular telephone or handheld or laptop computer, although a mobile node can also be a router.

- In a wireless network, nodes communicate using the communication model that consists of TCP agent, TCPSink agent, and FTP application.
- The sender node is attached to the TCP agent while the receiver node is attached to the TCPSink agent.
- The connection between TCP agent and TCPSink agent is established using the keyword “connect”.
- Transport agent (TCP) and application (FTP) are connected using the keyword “attach-agent”.
- TCP agent sends data to TCPSink agent. On receiving the data packet, TCPSink agent sends the acknowledgement to the TCP agent that in turn processes the acknowledgements.



NS2 Script generator

Steps:

1. to create the file
   gedit filename.tcl // P3 is file name
2. Write TCL script & click on 'save' to save the file.
3. To run the file:

   ns filename.tcl

TCL script:

#TCP (Transmission Control Protocol) and FTP (File

TransferProtocol) #defining Communication between

node0 and node1

#--Event scheduler object creation#

set ns [new

Simulator] set a

[open out.tr w]

$ns trace-all $a

set fr [open out.nam w]

$ns namtrace-all $fr

set n0 [$ns node]

set n1 [$ns node]

$ns duplex-link $n0 $n1 2Mb 2ms DropTail

 # Defining a transport agent for sending

set tcp0 [new Agent/TCP]

 # Defining a transport agent for receiving

set tcp1 [new Agent/TCPSink]

 # Attaching transport agent

$ns attach-agent $n0 $tcp0

$ns attach-agent $n1 $tcp1

#Connecting sending and receiving transport agents

$ns connect $tcp0 $tcp1

#Defining Application instance

set ftp [new Application/FTP]

 # Attaching transport agent to application agent

$ftp attach-agent

$tcp0 **#finish**

procedure# proc

finish { } {

global ns a fr

$ns flush-

trace close

$a

close $fr

exec nam out.nam

& exit

}

 # data packet generation starting time

$ns at .1 "$ftp start"

 # data packet generation ending time

$ns at 2 "$ftp stop"

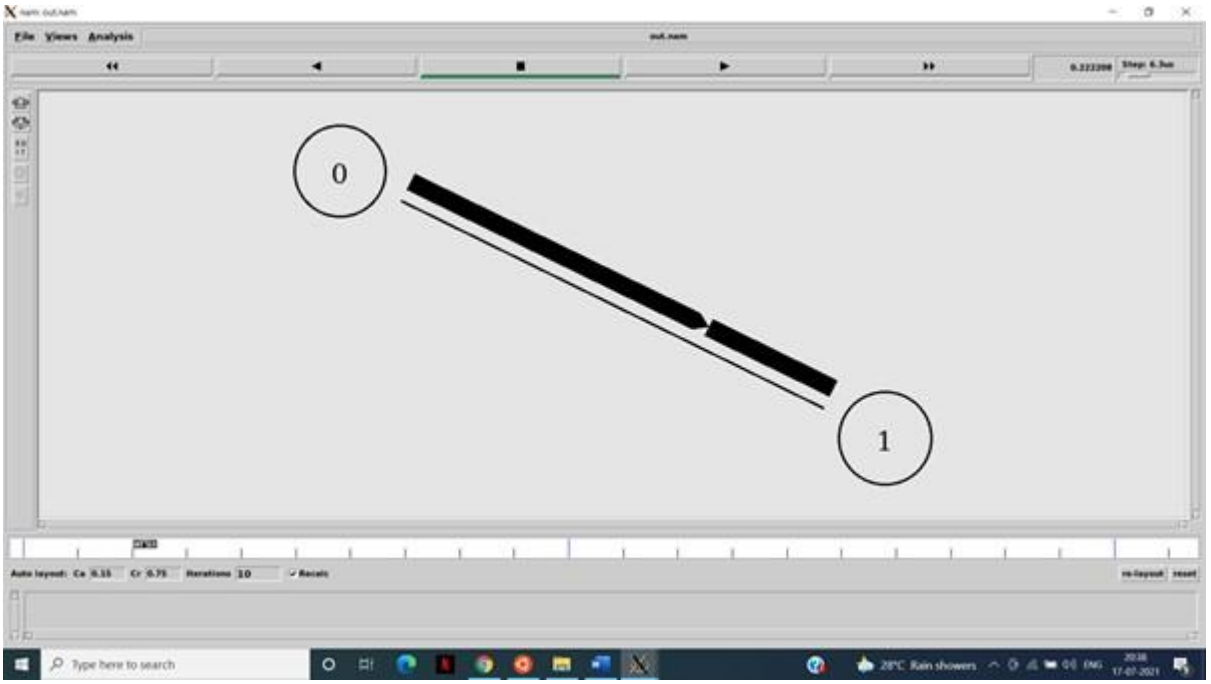
$ns at 2.1 "finish"

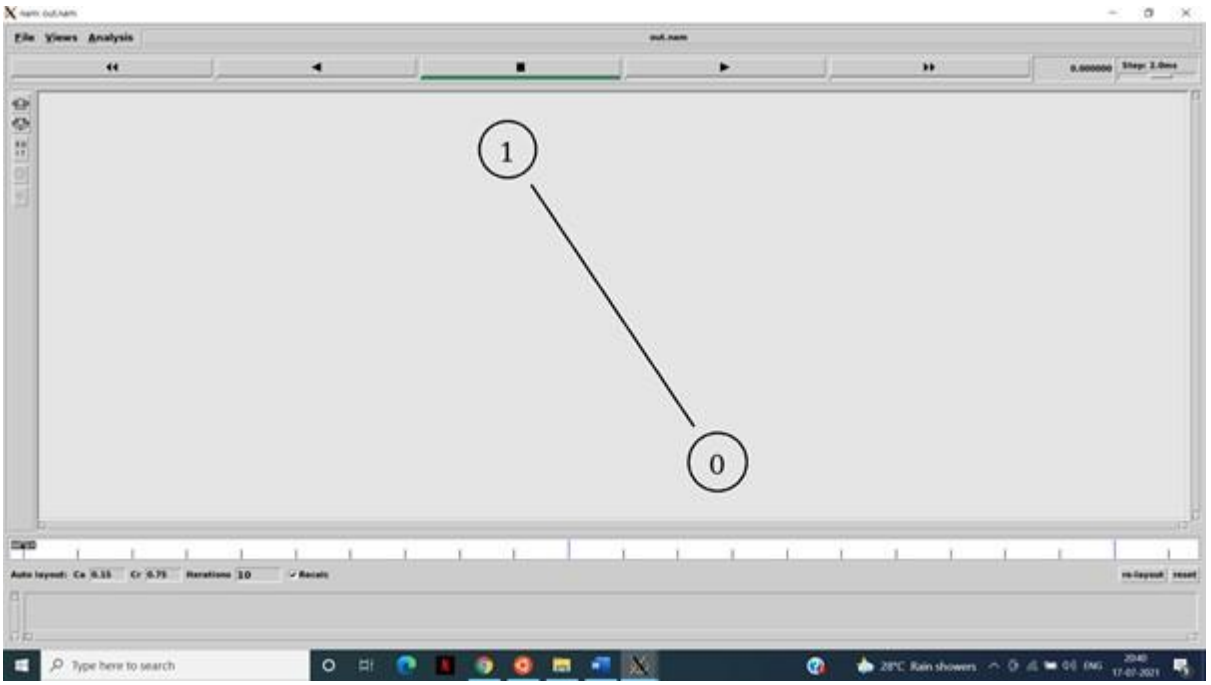
$ns run

#Run the file

ns filename.tcl

Output:





PRACTICAL 4

Aim: Write a TCL script for sensor nodes with following different parameters. Consider the scenario of connection between 4 mobile nodes for TCP using FTP traffic and UDP using CBR traffic.

Theory:

- The simulator object has member function which enables to create the nodes and define the links between them. The class simulator contains all the basic functions. Since ns was defined to handle the Simulator object, the command $ns is used for using the functions belonging to the simulator class.
- Traffic agents (TCP, UDP etc.) and traffic sources (FTP, CBR etc.) must be set up if the node is not a router. It enables to create CBR traffic source using UDP as transport protocol or an FTP traffic source using TCP as a transport protocol.

Steps for execution:

Step 1: Create file by using the below command.
gedit Pract4.tcl //Pract4 is file name

Step 2: Code with explanation

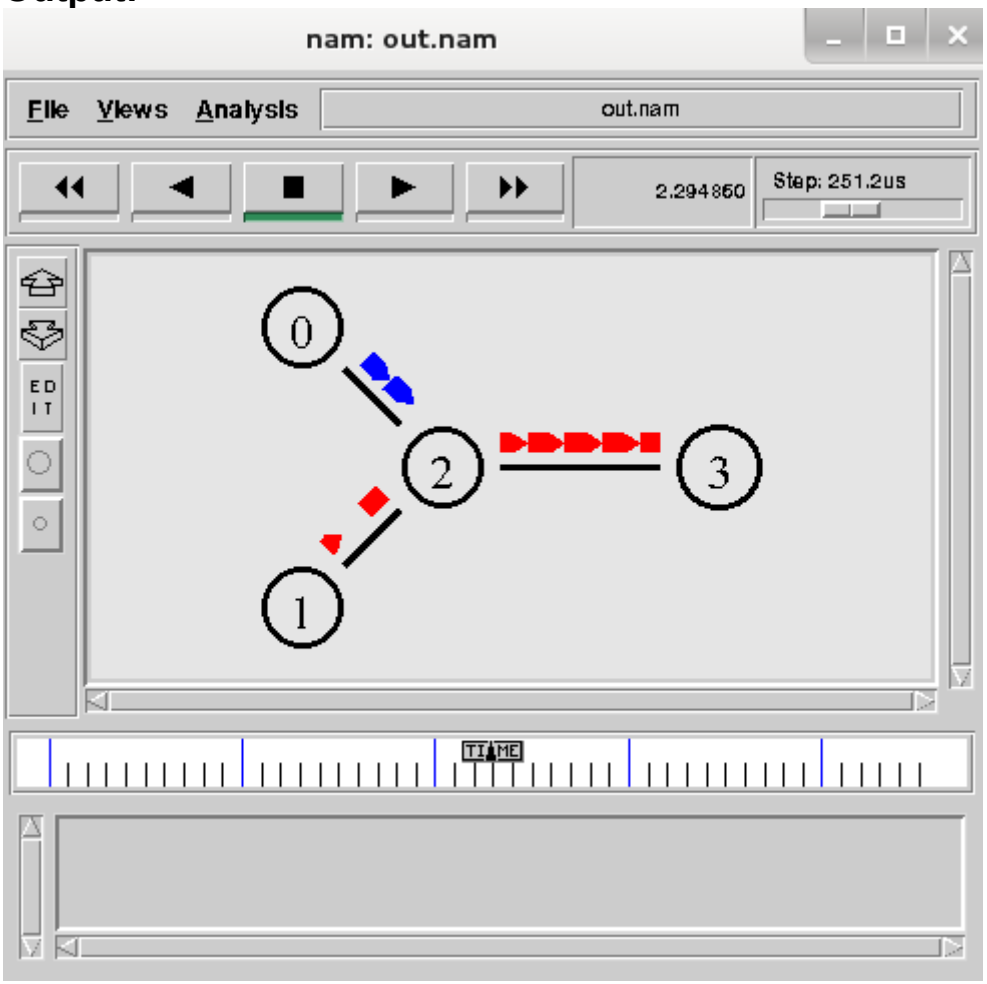
```
# Create a simulator object
set ns [new Simulator]

# Define different colors
# for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
# Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf
# Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        # Close the NAM trace
        file close $nf
        # Execute NAM on the trace
        file exec nam out.nam &
        exit 0
}
# Create four nodes
```

```
set n0 [$ns
node] set n1
[$ns node] set
n2 [$ns node]
set n3 [$ns
node]
# Create links between the nodes

$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail
# Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10
# Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
# Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5
# Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new
Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
# Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
# Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1
$udp set null [new
Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2
# Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


# Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
```

```
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
# Detach tcp and sink agents
# (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"
# Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
# Print CBR packet size and interval
puts "CBR packet size = [$cbr set
packet_size_]" puts "CBR interval = [$cbr
set interval_]"
$ns run
```

Step 3: Run the Pract4.tcl file with the below command:
ns Pract4.tcl

Output:



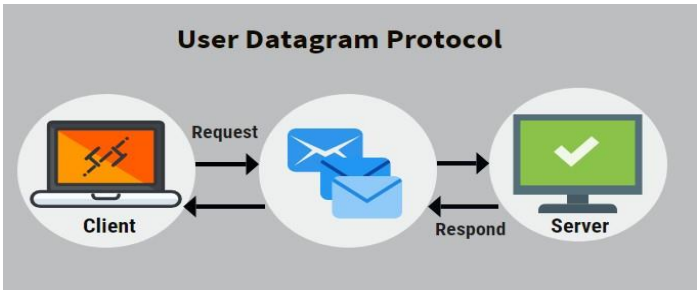
Practical 5

Aim: Generate tcl script for udp and CBR traffic in WSN nodes.

THEORY:

What is UDP?

- User Datagram Protocol (UDP) is one of the core members of the **Internet Protocol Suite**
- With UDP, computer applications can send messages, in this case referred to as ***datagrams***, to other hosts on an Internet Protocol (IP) network. Prior communications are not required in order to set up communication channels or data paths.
- UDP uses a simple connectionless communication model with a minimum of protocol mechanisms
- UDP provides **checksums** for data integrity, and port numbers for addressing different functions at the source and destination of the datagram



What is TCL simulation script?

A configuration file in NS2 is called "**TCL Simulation script**". It also contains information about what we would like to **simulate** like node creation, also topology creation, setting up links etc. A **TCL** file is also an input configuration file for C++ file. To run the **TCL** file, use the command. ns filename.

Features of TCL :

- Reduced development time.
- Powerful and simple user interface.
- Write once, run anywhere. It runs on Windows, Mac OS X, and almost on every Unix platform.
- Easy to get started for experienced programmers since the language is so simple.
- Has a powerful set of networking functions.

- Finally, it's an open source, free, and can be used for commercial applications without any limit.

Applications :

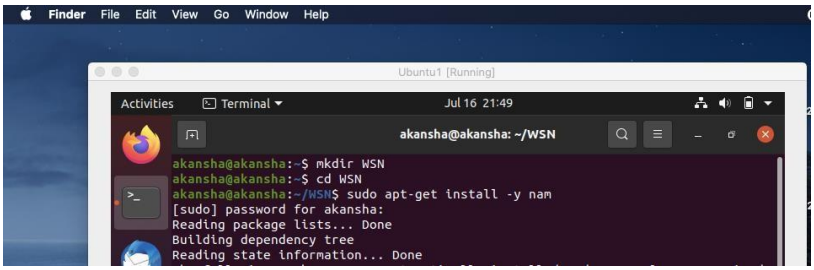
- Scalable websites that are often backed by databases.
- High performance web servers.
- Tcl with CGI based websites.
- Desktop GUI applications.
- Embedded applications.

What is CBR?

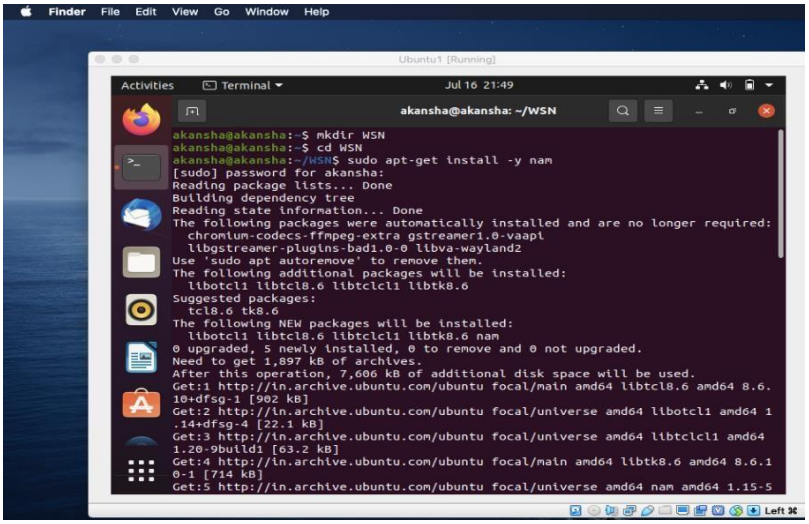
The CBR service is used for connections that transport traffic at a **constant bit rate**. Where there is an inherent reliance on time synchronisation between the traffic **source** and **destination**.These applications include services such as video conferencing, telephony or any type of on- demand service ,such as interactive voice and audio.

Step 1: Open Virtualbox -> Open Ubuntu OS(20.04)

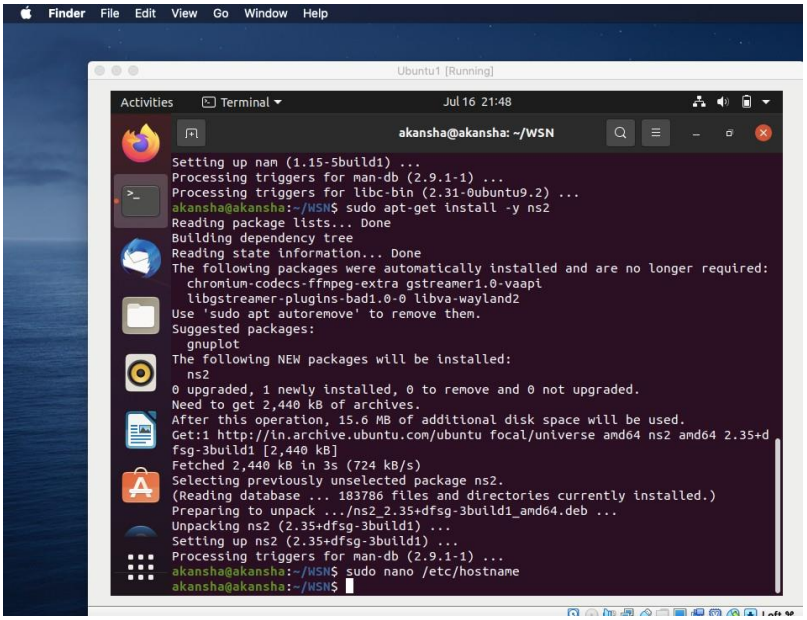
a. Open terminal window.
b. Make folder using **mkdir 'folder name'** (eg: mkdir WSN)
c. Open that folder using **cd 'folder name'**



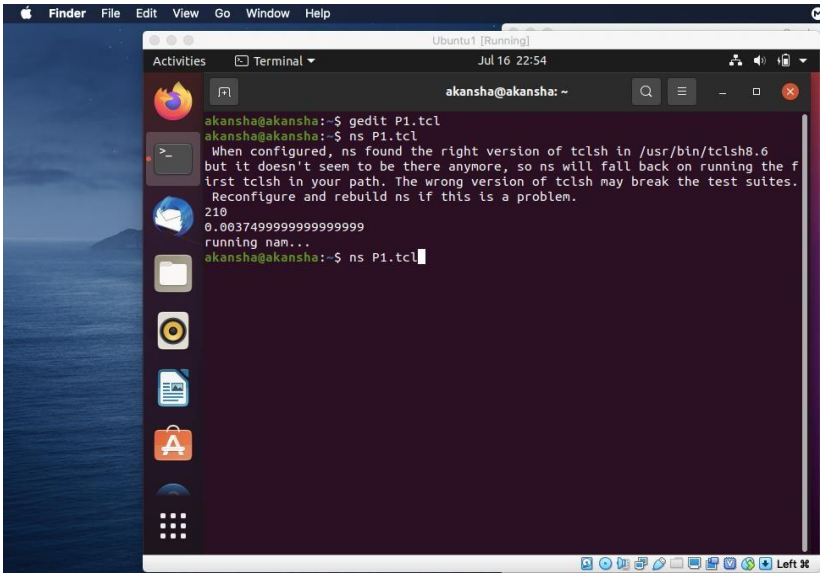
Step 2: Type the following command and press enter: **sudo apt-get install -y nam**

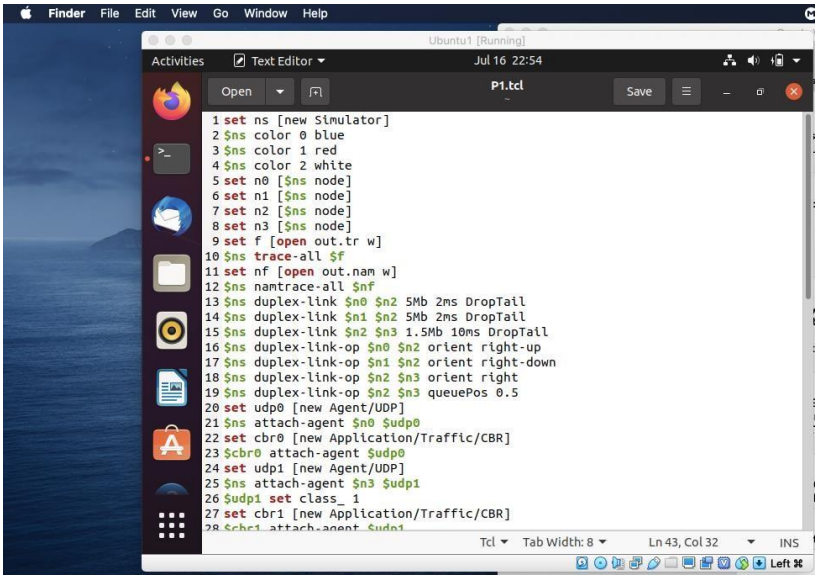


Step 3: Type the following command and press enter: **sudo apt-get install -y ns2**



Step 4 : Create File, **gedit P1.tcl**





Code:

#Setting the new simulator
set ns [new Simulator]

#Set the color for the packet within frame
$ns color 0 blue
$ns color 1 red
$ns color 2 white

#Create 4 nodes, n0, n1, n2, n3
set n0 [$ns node] set n1 [$ns node] set n2 [$ns node] set n3 [$ns node]

#Create the trace file
set f [open out.tr w]
$ns trace-all $f
set nf [open out.nam w]
$ns namtrace-all $nf

#Create Duplex link between nodes, it takes 3 parameters Data rate, Delay and Type of queue.
$ns duplex-link $n0 $n2 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
#Orient the links between the nodesrielinks
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right

#To monitor between node n2 and n3 and angle is 0.5
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Create udp0 and attach it to n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

#**Set cbr0 and attach it to udp0**
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0

#Create udp1 and attach it to n3
set udp1 [new Agent/UDP]
$ns attach-agent $n3 $udp1

#Set the udp1 packets to class_1(red)
$udp1 set class_ 1

#Set cbr1 and attach to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1

#Set null0 agent and attach it
to n3 set null0 [new Agent/Null]
$ns attach-agent $n3 $null0

#Set null1 agent and attach it to n1
set null1 [new Agent/Null]
$ns attach-agent $n1 $null1

 # Connect the udp0 to null0 & udp 1 to null1
$ns connect $udp0 $null0
$ns connect $udp1 $null1

 # Set the interval of the packets cbr0 & cbr1
$ns at 1.0 "$cbr0 start"
$ns at 1.1 "$cbr1 start"

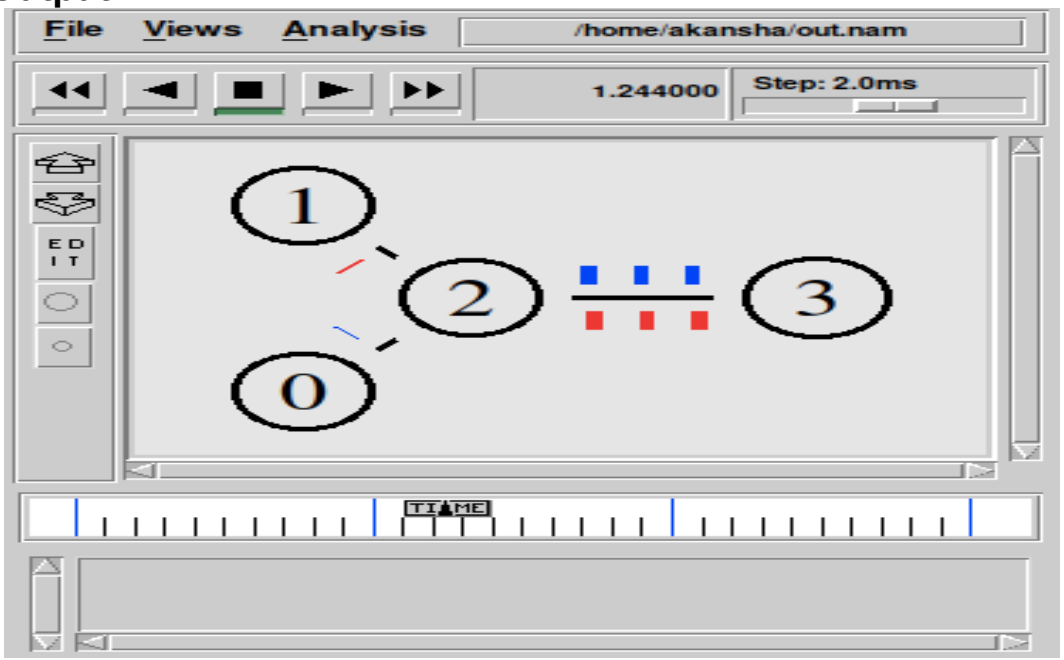
#Set tcp Agent
#set tcp [new Agent/TCP]

#Set the tcp packets to class_2(blue)
#$tcp set class_ 2
#Set the packet size of cbr0 puts [$cbr0 set packetSize_] puts [$cbr0 set interval_]

```
#Set the end time
$ns at 3.0
"finish" proc
finish {} { global
ns f nf
$ns flush-
trace close $f
close $nf
puts          "running
nam..."   exec   nam
out.nam & exit 0
}
$ns run
```

Step 3: Run the P4.tcl file type command: **ns P1.tcl**

Output:



Practical 6

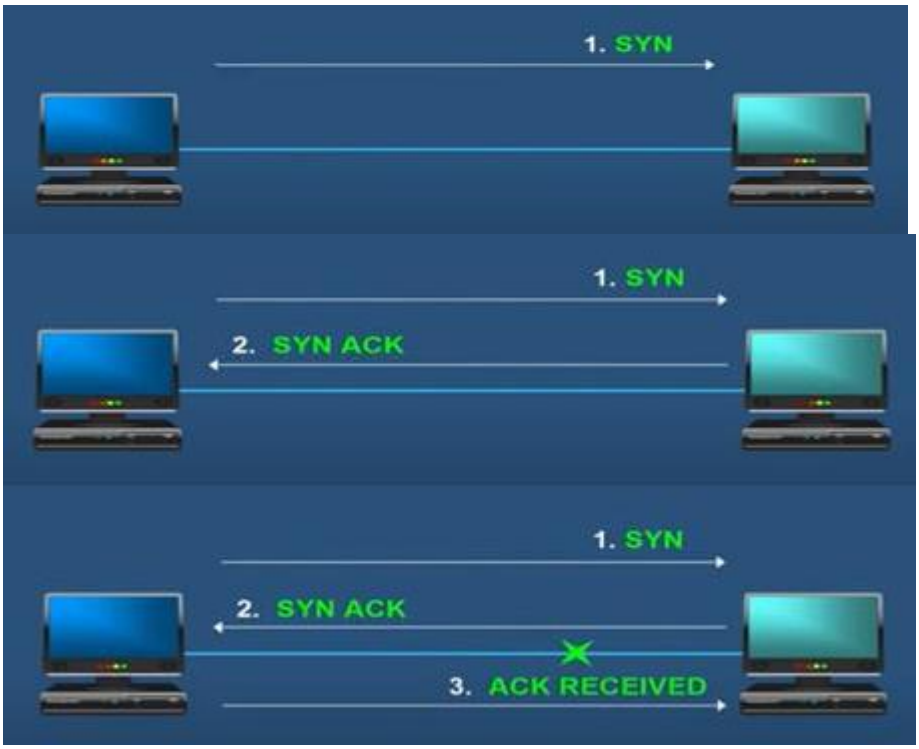
Aim: Generate tcl script for TCP and CBR traffic in WSN nodes.

Theory:

1. **TCP**

- Stands for Transmission Control Protocol.
- TCP is a commonly used connection-oriented transport control protocol for the internet.
- TCP uses network services provided by IP layer, with the objective of offering reliable, orderly, controllable, and elastic transmission.
- TCP operation consists of three phases:

1. Connection Establishment.
2. Data Transmission.
3. Disconnect.



2) CBR

- CBR stands for constant bit rate .
- It plays a vital role in terms of Network traffic.
- Provides QOS guarantee.

CBR-NS2

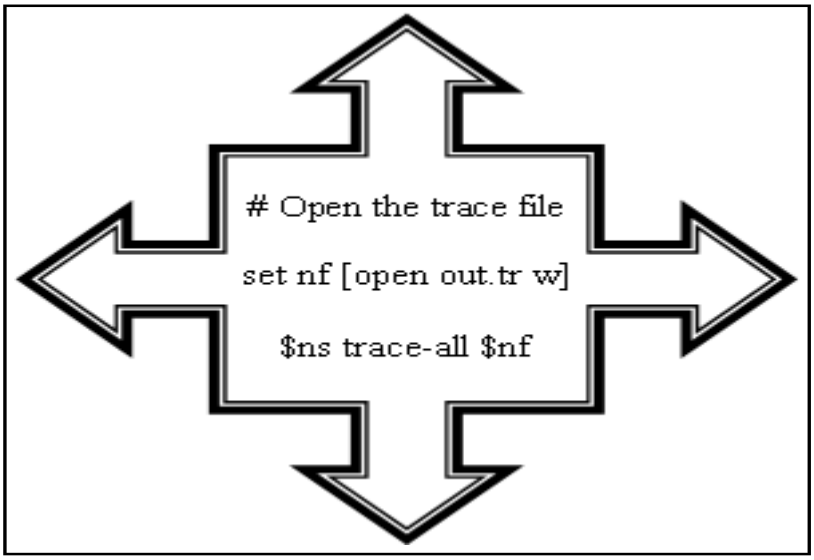
Constant bit rate [CBR] Ns2 is used along with TCP and UDP to design the traffic source behaviour of packets.

Setting up NS2.

NS2 (Network Simulator version 2):

- Network simulators provide a virtual network used for experiment in only one computer.
- NS2 is free and easy to use and is the popular all over the world.
- To observe the communication between different nodes in a network, we can set up a network topology or simulator to do experiments.
- NS2 use Tcl language for creating simulation scenario file (for example, sample.tcl).
- If we execute this scenario file, the simulation results will be outputted to out.tr and out.nam file.

- **out.tr** all the information about communication is written in this file. We can find out the way a packet was forwarded.This file is called as trace file.



Syntax for generating trace file inNS2:

File name is out and also said stored data in trace

format. nf ->file handler

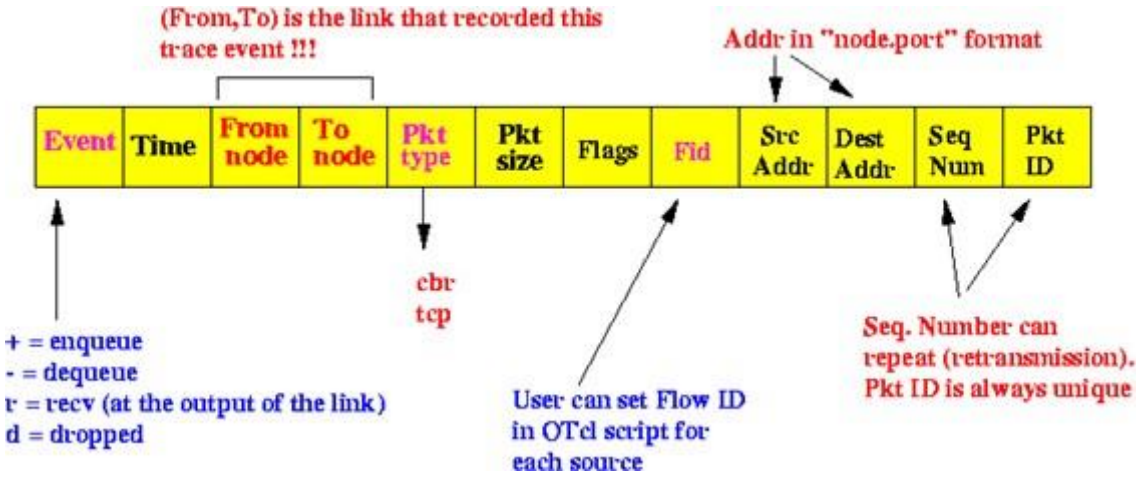
w -> write which means file out.tr is opened for

writing r -> reading

a ->appending

The second line tells the simulator to trace each packet on every link in the topology and for that we give file handler nf for the simulator ns

Structure of trace file: -



out.nam contains the data for animation of the experiment result. This file can be executed by Nam, an animation software.

A performance evaluation of TCP is presented, handling network heterogeneity and QoS tenability. This work focuses on the study of critical
transport layer parameters and their impact on QoS and power consumption of the overall system.

Conclusion: -
Summarizing, the transport layer behaviour regarding TCP protocols was evaluated. Simulation results show that TCP suffers on multihop wireless routes, managing to deliver minimum amount of packets on destination. Therefore, fewer data packets can be sent over the multihop wireless routes compared to UDP protocol. Furthermore, delay is short, because the transmit window of TCP was minimal. On the other hand, UDP achieved better results in throughput, although

its mean delay was higher compared to TCP. The reason UDP is faster than TCP is because there is no form of
flow control or error correction which also explains the fact that delay over UDP is higher compared to TCP. Finally, as power is concerned, TCP is a more power consuming protocol than UDP, due to the complexity of TCP's structure, i.e., acknowledgment packets must be sent.

Steps:

0) Declare Simulator and setting output file

1) Setting Node and Link
2) Setting Agent
3) Setting Application
4) Setting Simulation time and schedules
5) Declare finish.

In general, an NS script starts with making a Simulator object instance.
set ns [new Simulator]: generates an NS simulator object instance, and assigns it to variable ns.
set n0 [$ns node]: The member function node creates a node.
ns duplex-link node1 node2 bandwidth delay queue-type: creates two simplex links of specified bandwidth and delay, and connects the two specified nodes.

set tcp [new Agent/TCP]: This line shows how to create a TCP agent.
$ns attach-agent node agent: The attach-agent member function attaches an agent object created to a node object. Actually, what this function does is call the attach member function of specified node, which attaches the given agent to itself.

$ns connect agent1 agent2: After two agents that will communicate with each other are created, the next thing is to establish a logical network connection between them.
$ns at time "string": This member function of a Simulator object makes the scheduler to schedule the execution of the specified string at given simulation time.
$ns run: To run the simulation.

Code:

 # Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Create 4
nodes set n0
[$ns node] set
n1 [$ns node]
set n2 [$ns
node]

set n3 [$ns node]

#Open the NAM trace file
set f [open out.tr w]
$ns trace-all $f
set nf [open out.nam w]

```
$ns namtrace-all $nf

#Create links between nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10


#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link n2 to n3, and angle is 0.5 (0.5pi)
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Set up a Transport layer Connection
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new
Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup an FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$tcp set type_ FTP


#Set up a UDP Connection
set udp [new Agent/UDP]
$ns attach-agent $n1
$udp set null [new
Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2
#Setup an CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000

#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
```

```
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
```

#Detach tcp and sink agents

```
$ns at 4.5 "$ns detach-agent $n0 $tcp; $ns detach-agent $n3 $sink"
```

#Print CBR packet size and interval

```
puts [$cbr set
packet_size_] puts [$cbr
set interval_]
```

#Call the finish procedure after 5 seconds of simulation time

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

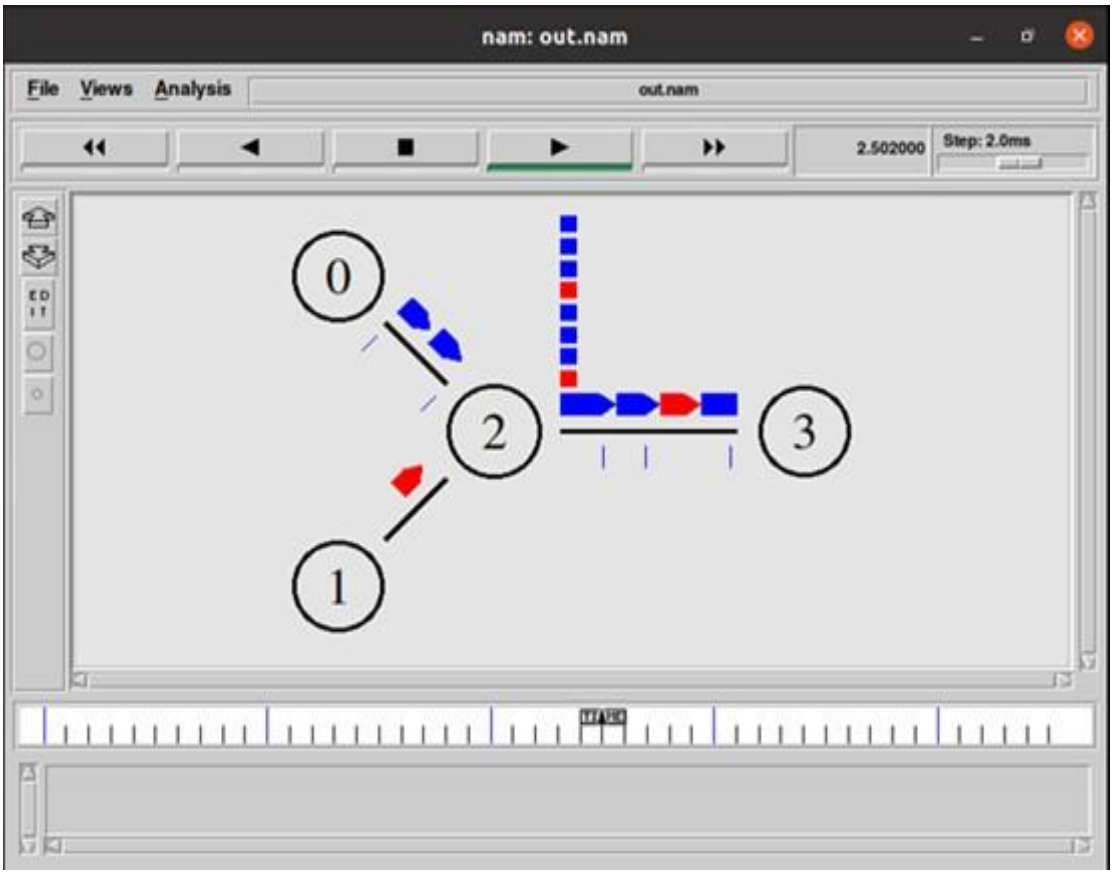
#Define a 'finish' procedure

```
proc finish {}
{ global ns f
nf
$ns flush-trace

close  $f
close
$nf
puts          "running
nam..."   exec   nam
out.nam & exit 0
}
```

#Run the simulation

```
$ns run
```

Output:


nam: out.nam
File
Views
Analysis
out.nam
2.502000
Step: 2.0ms
0
1
2
3

Practical 7

AIM : Implementation of routing protocol in NS2 for AODV protocol

THEORY:

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 UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue 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.

The AODV protocol builds routes between nodes only if they are requested by source nodes. AODV is therefore considered an on-demand algorithm and does not create any extra traffic for communication along links. The routes are maintained as long as they are required by the sources. They also form trees to connect **AODV:**

AODV (Ad-hoc On-demand Distance Vector) is a loop-free routing protocol for ad-hoc networks.

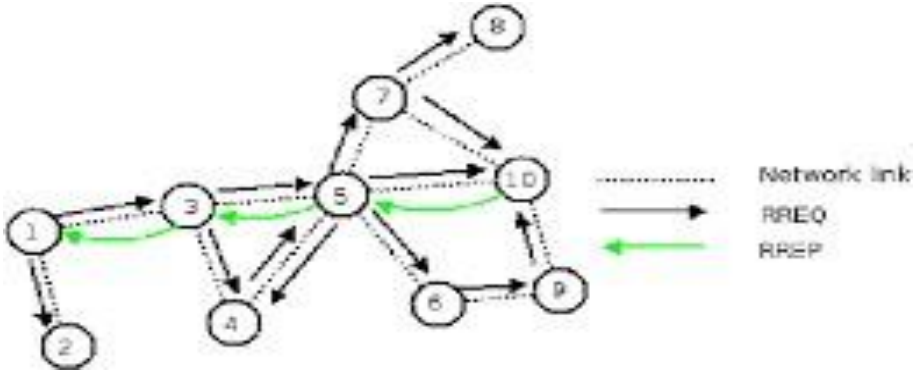
It is designed to be self-starting in an environment of mobile nodes, withstanding a variety of network behaviors such as node mobility, link failures and packet losses.

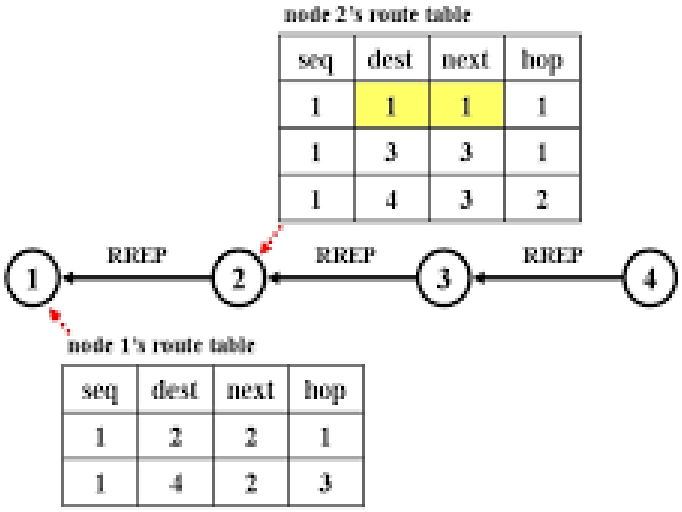
At each node, AODV maintains a routing table. The routing table entry for a destination contains three essential fields:

1. Next hop node,
2. Sequence number
3. The hop count.

multicast group members. AODV makes use of sequence numbers to ensure route freshness. They are self-starting and loop-free besides scaling to numerous mobile nodes.

All packets destined to the destination are sent to the next hop node. The sequence number acts as a form of time-stamping, and is a measure of the freshness of a route. The hop count represents the current distance to the destination node





CODE:

Step 1 : To create the file

gedit adov.tcl

Step 2 : add code in same

file Code:-

 # A 12-node example for ad-hoc simulation with AODV

 # Setting the parameters/Define the parameter options

set val(chan) Channel/WirelessChannel ;# channel type

set val(prop) Propagation/TwoRayGround ;# radio-propagation

model set val(netif) Phy/WirelessPhy ;# network

interface type

```
set val(mac)        Mac/802_11            ;# MAC type
set val(ifq)        Queue/DropTail/PriQueue;# interface queue
type set val(ll)  LL                  ;# link layer type
set val(ant)        Antenna/OmniAntenna     ;# antenna model
set val(ifqlen)      50                  ;# max packet in ifq
set val(nn)         12                  ;# number of mobilenodes
set val(rp)         AODV                 ;# routing protocol
set val(x)          500                 ;# X dimension of
                                        topography
set val(y)          400                 ;# Y dimension of
                                        topography
set val(stop)       150                  ;# time of simulation end
```

 # Simulator Instance Creation

```
set ns       [new Simulator]
set tracefd     [open testAODV.tr
w] set windowVsTime2 [open
win.tr w]
set namtrace    [open testAODV.nam w]

$ns trace-all $tracefd
$ns namtra# set up topography object
set topo     [new Topography]

$topo load_flatgrid $val(x) $val(y)

create-god $val(nn)

#
# Create nn mobilenodes [$val(nn)] and attach them to the
channel. #
ce-all-wireless $namtrace $val(x) $val(y)
```

```
        # configure the
                    nodes
        $ns node-config -adhocRouting $val(rp) \
          -llType $val(ll) \
            -macType $val(mac) \
            -ifqType $val(ifq)
\ ifqLen $val(ifqlen) \
            -antType $val(ant) \
            -propType $val(prop) \
            -phyType $val(netif) \
            -channelType $val(chan) \
            -topoInstance $topo \
            -agentTrace ON \
            -routerTrace ON \
            -macTrace OFF \
            -movementTrace ON


    for {set i 0} {$i < $val(nn) } { incr
       i } { set node_($i) [$ns node]
       $node_($i) set X_ [ expr 10+round(rand()*480) ]
       $node_($i) set Y_ [ expr 10+round(rand()*380) ]
       $node_($i) set Z_ 0.0
    }


    for {set i 0} {$i < $val(nn) } { incr i } {
       $ns at [ expr 15+round(rand()*60) ] "$node_($i) setdest [ expr
10+round(rand()*480) ] [ expr 10+round(rand()*380) ] [ expr 2+round(rand()*15) ]"


    }
# Generation of movements
 $ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
```

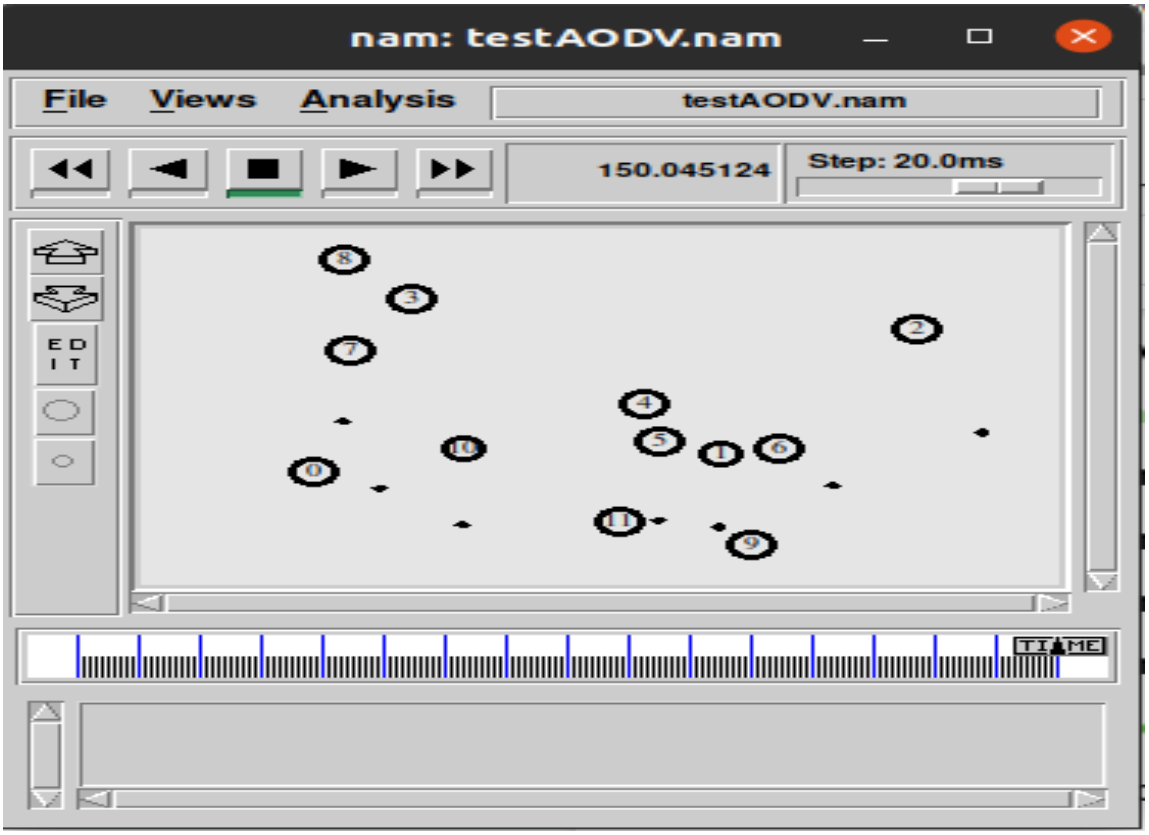
```
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 70.0 "$node_(2) setdest 480.0 300.0 5.0"
$ns at 20.0 "$node_(3) setdest 200.0 200.0 5.0"
$ns at 25.0 "$node_(4) setdest 50.0 50.0 10.0"
$ns at 60.0 "$node_(5) setdest 150.0 70.0 2.0"
$ns at 90.0 "$node_(6) setdest 380.0 150.0 8.0"
$ns at 42.0 "$node_(7) setdest 200.0 100.0 15.0"
$ns at 55.0 "$node_(8) setdest 50.0 275.0 5.0"
$ns at 19.0 "$node_(9) setdest 250.0 250.0 7.0"
$ns at 90.0 "$node_(10) setdest 150.0 150.0 20.0"
```

 # Set a TCP connection between node_(2) and node_(8)

```
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(2) $tcp
$ns attach-agent $node_(11)) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"
# Printing the window size
proc plotWindow {tcpSource
file} { global ns
set time 0.01
set now [$ns now]
set cwnd [$tcpSource set
cwnd_] puts $file "$now
$cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 10.1 "plotWindow $tcp $windowVsTime2"
```

```
# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr
i } { # 30 defines the node size
for nam
$ns initial_node_pos $node_($i) 30
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
   $ns at $val(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150 "puts \"end simulation\" ; $ns
halt" proc stop {} {
   global ns tracefd namtrace
   $ns flush-trace
   close $tracefd
   close
   $namtrace
}

$ns run
```

Output:

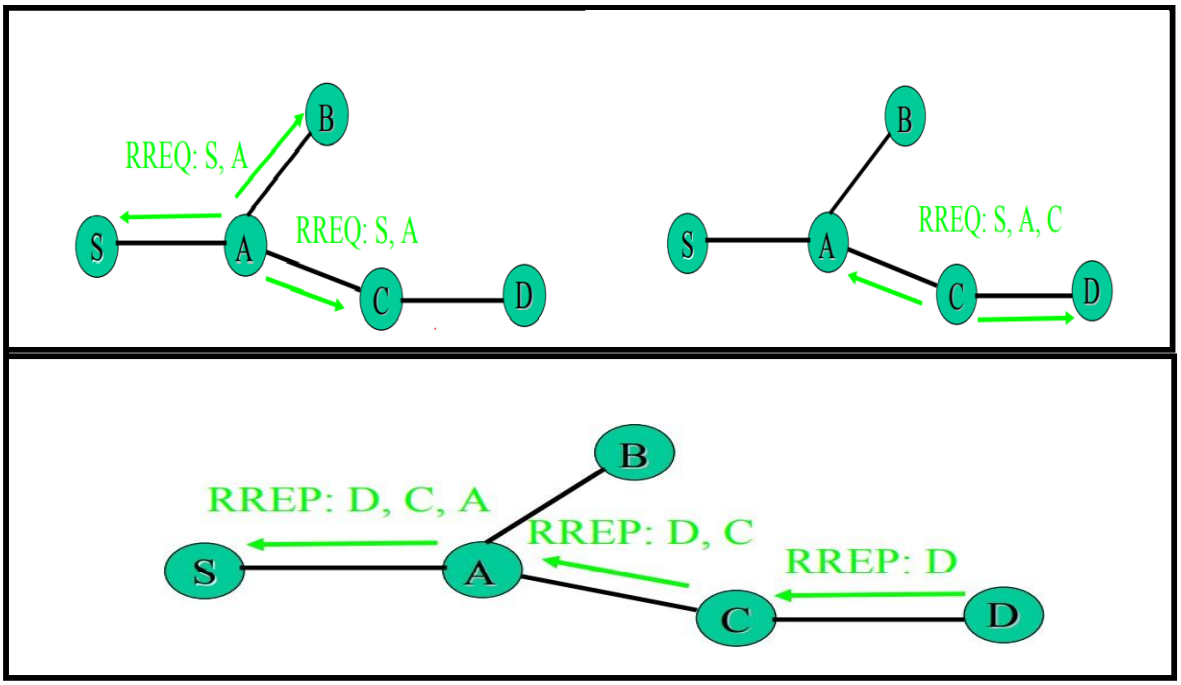

nam: testAODV.nam
File
Views
Analysis
testAODV.nam
150.045124
Step: 20.0ms
EDIT
TIME

PRACTICAL 8

Aim : Implementation of routing protocol in NS2 for DSR protocol.

Theory : What is DSR?

- The Dynamic Source Routing protocol (DSR) is a simple and efficient routing protocol designed specifically for use in multi-hop wireless ad hoc networks of mobile nodes.
- DSR allows the network to be completely self-organizing and self-configuring, without the need for any existing network infrastructure or administration which makes it a reactive protocol.
- Source routing is a routing technique in which the sender of a packet determines the complete sequence of nodes through which the packets are forwarded.
- The protocol is composed of the two main mechanisms of "Route Discovery" and "Route Maintenance".
- Discovery determines the optimum path for a transmission between a given source and destination.
- Route Maintenance ensures that the transmission path remains optimum and loop- free as network conditions change, even if this requires changing the route during a transmission.



Steps for execution:

Step 1 : Create file by using the below command.
gedit DSR.tcl //DSR is file name

Step 2 : Code with explanation
 # A 10-node example for ad-hoc simulation with DSR # Define options

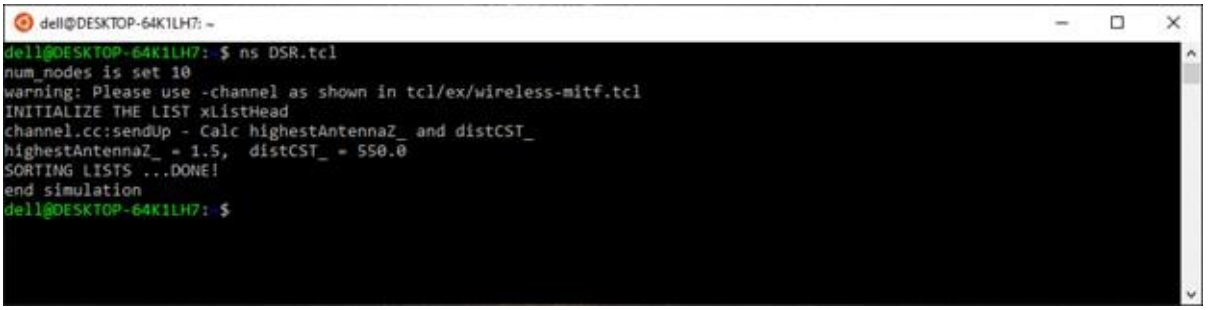
```
set val(chan)   Channel/WirelessChannel       ;# channel type
set val(prop)  Propagation/TwoRayGround ;# radio-propagation model
set val(netif)  Phy/WirelessPhy                  ;# network interface
type set val(mac)                          Mac/802_11         ;# MAC
type
set val(ifq)            CMUPriQueue                    ;# interface queue type
set val(ll)             LL                             ;# link layer type
set val(ant)            Antenna/OmniAntenna            ;# antenna model
set             50                             ;# max packet in ifq
val(ifqlen)
set val(nn)             10                             ;# number of mobile nodes
set val(rp)             DSR                            ;# routing protocol
set val(x)              500                            ;# X dimension of
                                                       topography
set val(y)              400                            ;# Y dimension of
set val(stop)  50                                      topography
                                                    ;# time of simulation end
set ns [new Simulator]

set tracefd [open simple-dsdv.tr
w] set windowVsTime2 [open
win.tr w] set namtrace [open
simwrls.nam w]
$ns trace-all $tracefd
$ns use-newtrace
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object

set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)
#
# Create nn mobilenodes [$val(nn)] and attach them to the channel.
#
# configure the nodes
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-channelType $val(chan) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace ON

for {set i 0} {$i < $val(nn) } { incr i } {
set node_($i) [$ns node]
```

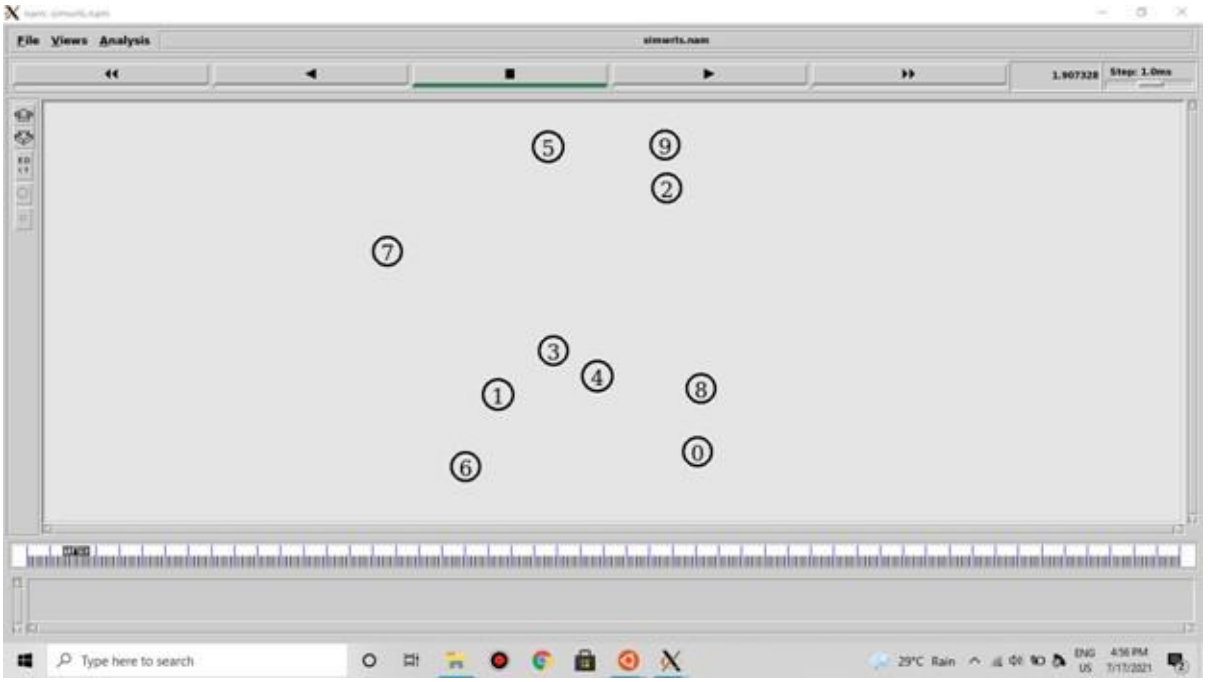
```
}
for {set i 0} {$i < $val(nn)} {incr i} {
$node_($i) set X_ [expr rand()*500]
$node_($i) set Y_ [expr rand()*400]
$node_($i) set Z_ 0
}
# Generation of movements
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 110.0 "$node_(0) setdest 480.0 300.0 5.0"

# Set a TCP connection between node_(0) and node_(1)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(9) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 2.0 "$ftp start"

for {set i 0} {$i<$val(nn)} {incr i} {
$ns initial_node_pos $node_($i) 30
}
# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
$ns at $val(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150.01 "puts \"end simulation\" ; $ns
halt" proc stop {} {
global ns tracefd namtrace
$ns flush-trace
close $tracefd
close
$namtrace
exec nam simwrls.nam &
}
$ns run
```

Step 3 : Run the DSR.tcl file with the below command.

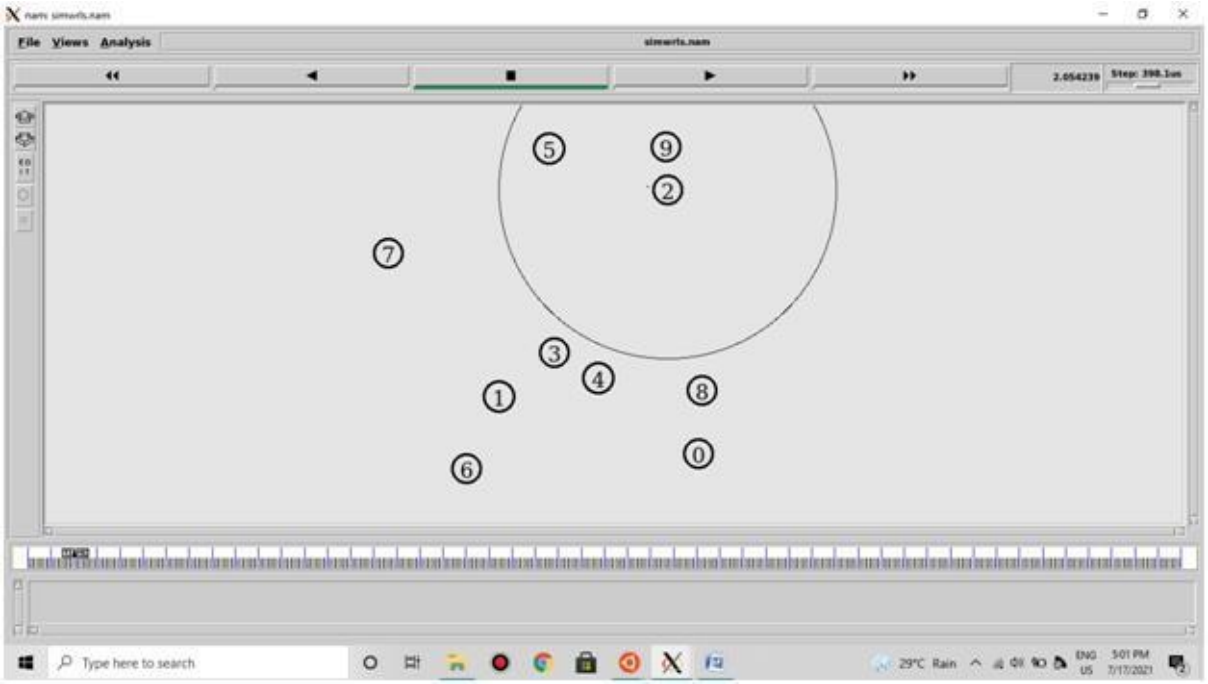
ns DSR.tcl

Output:

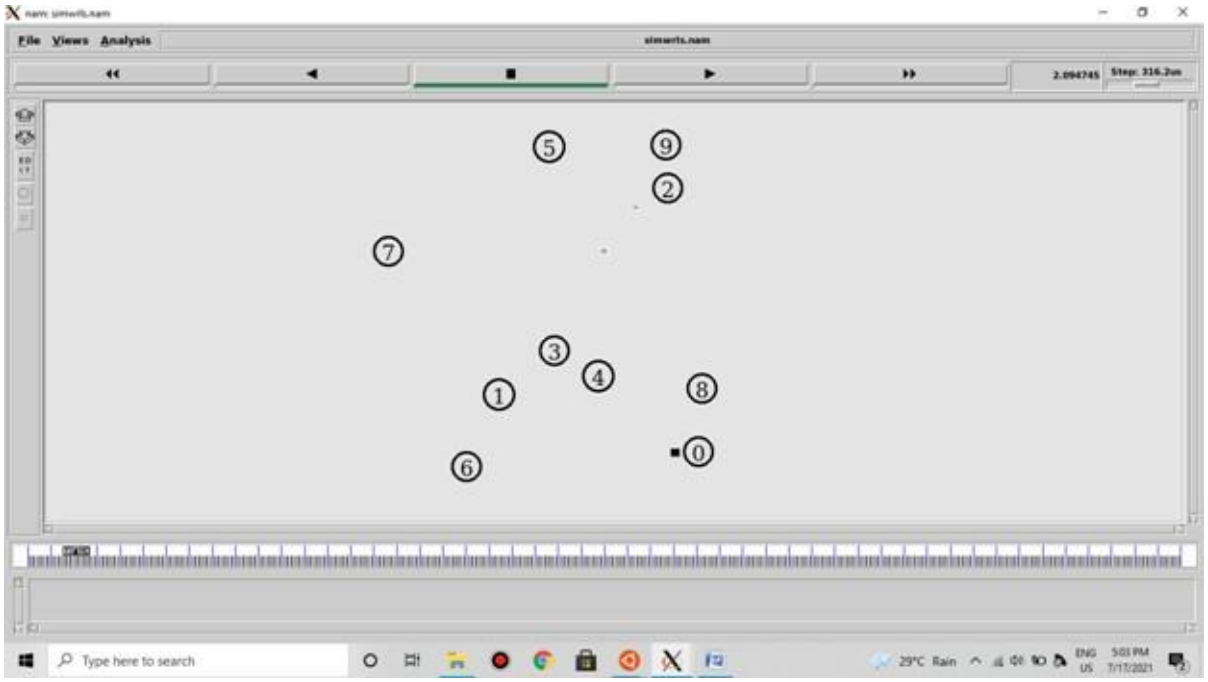




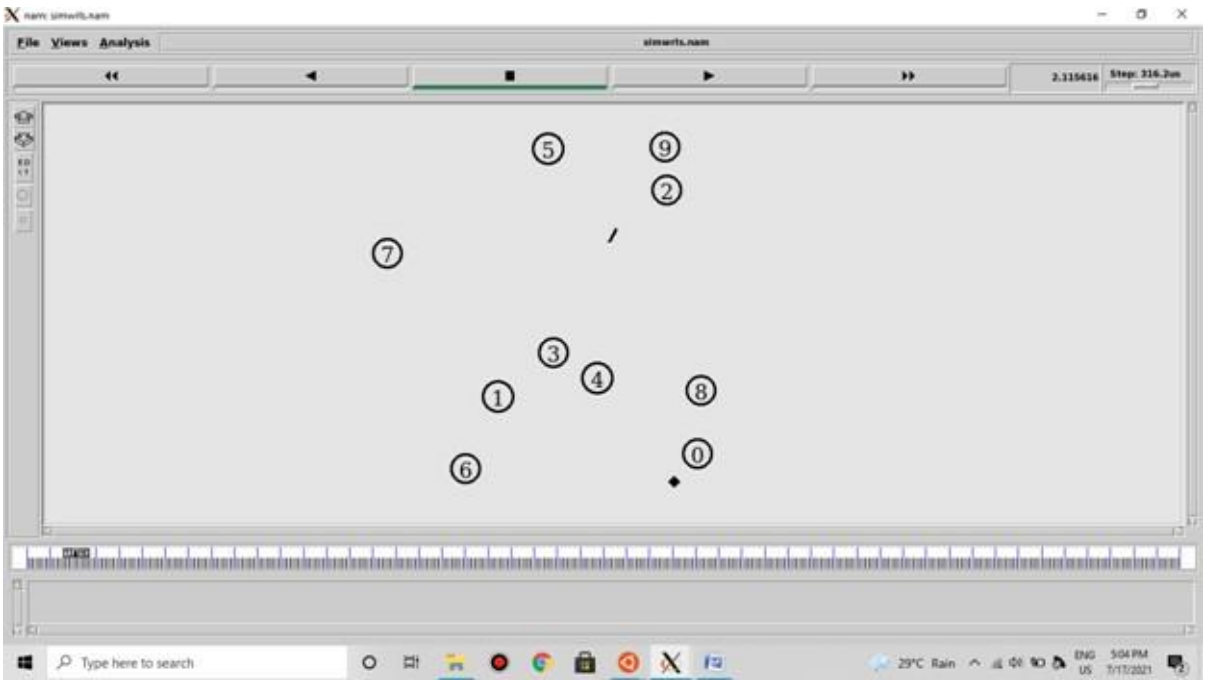
A. Initialized nodes



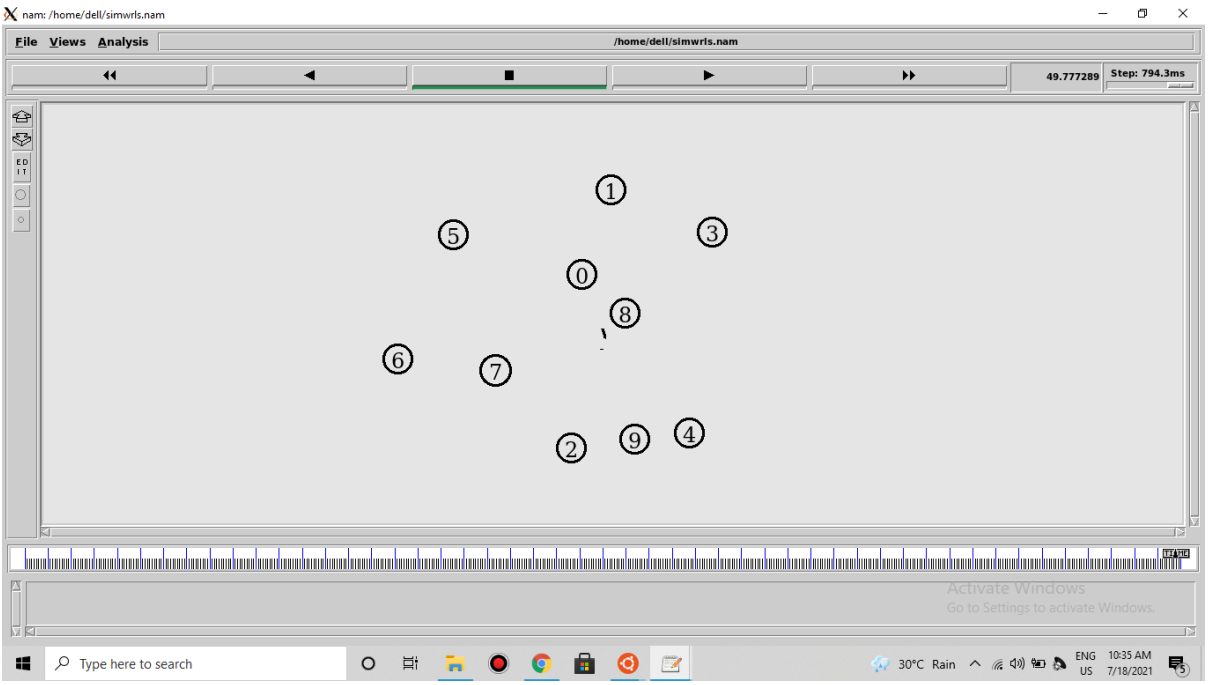
B. Route discovery



C.Dropping of packets



D. Transfer of packets



E. Position changed of nodes

PRACTICAL 9

Aim: Implementation of routing protocol in NS2 for DSDV protocol.

Theory:

- Proactive routing protocol maintains constant and updated routing information for each pair of networking nodes by propagating route updates proactively at fixed interval of time.
- The periodic and event-driven messages are responsible for route establishment and route maintenance.
- The Destination-Sequenced Distance Vector (DSDV) protocol is the commonly used proactive routing protocol in mobile ad hoc network (MANET).
- In DSDV, each node maintains a routing table with one route entry for each destination in which the shortest path is recorded.
- It uses a destination sequence number to avoid routing loops.
- The DSDV.tcl shows a node configuration for a wireless mobile node that runs DSDV as its adhoc routing protocol.
- Prior to the establishment of communication between the source and receiver node, the routing protocol should be mentioned to find the route between them.
- Data Transmission is established between nodes using UDP agent and CBR traffic. Routing process follows DSDV routing protocol.

Steps for execution:

Step 1: Create file by using the below command.
gedit DSDV.tcl //DSDV is file name

Step 2: Code with explanation

- **# A 9-node example for ad-hoc simulation with ASDV**
- **# Setting the parameters/Define the parameter options**
- set val(chan) Channel/WirelessChannel ;# channel type
- set val(prop) Propagation/TwoRayGround ;# radio-propagation model
- set val(netif) Phy/WirelessPhy ;# network interface type
- set val(mac) Mac/802_11 ;# MAC type(for wireless)
- set val(ifq) Queue/DropTail ;# interface queue type
- set val(ll) LL ;# link layer type
- set val(ant) Antenna/OmniAntenna ;# antenna model
- set val(ifqlen) 50 ;# max packet in ifq
- set val(nn) 9 ;# number of mobile nodes
- set val(rp) DSDV ;# routing protocol
- set val(x) 500 ;# X dimension of topography

- set val(y) 400 ;# Y dimension of topography
- set val(stop) 150 ;# time of simulation

```
set ns [new Simulator]
# Open up trace & nam
file set tracefd [open
DSDV.tr w]
set windowVsTime2 [open win.tr
w] set namtrace [open
DSDV.nam w]
$ns trace-all $tracefd
$ns use-newtrace
$ns namtrace-all-wireless $namtrace $val(x) $val(y)
# set up topography object
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
# general operational descriptor- storing the hop details in the
network create-god $val(nn)
#
# Create nn mobilenodes [$val(nn)] and attach them to the
channel. #
# configure the nodes
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-channelType $val(chan) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace ON

# Node Creation by incrementing i create the 9 nodes,ie here nn=9
for {set i 0} {$i < $val(nn) } { incr
i } { set node_($i) [$ns node]
        }
# Provide initial location of mobile nodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
```

```
$node_(1) set Z_ 0.0
$node_(2) set X_ 150.0
$node_(2) set Y_ 240.0
$node_(2) set Z_ 0.0

# Generation of movements/Defining the mobility
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
# For mobility 250.0= movement x value i.e X coordinate, 250.0=movement y value
i.e Y coordinate, 3.0=speed in m/s
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 110.0 "$node_(0) setdest 480.0 300.0 5.0"

# Set a TCP connection between node_(0) and node_(1)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(8) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"
# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr
i } { # 30 defines the node size
for nam
$ns initial_node_pos $node_($i) 30
}
# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
$ns at $val(stop) "$node_($i) reset";
}
# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150.01 "puts \"end simulation\" ; $ns
halt" proc stop {} {
global ns tracefd namtrace
$ns flush-trace
close $tracefd
close
$namtrace
exec nam DSDV.nam &
}
$ns run
```

Step 3: Run the DSDV.tcl file with the below command.

ns DSDV.tcl

Output: