WIRELESS SENSOR NETWORKS LAB

MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY, JAIPUR

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

WSN LAB ASSIGNMENTS

SUBMITTED TO: DR. MEENAKSHI TRIPATHI
SUBMITTED BY: UJESH MAURYA
ID: 2015UCP1338
SEMESTER: 8 (2015-19)

ASSIGNMENT 1

Objective: To study about Emulators and Simulators and the differences between them with specific reference to Network Simulator NS-3 and then answer the following questions:

Question 1. What is NS-3?

Ans. NS-3 is a discrete-event network simulator targeted primarily for research and educational use. It has been developed to provide an open, extensible network simulation platform for networking research and education.

It provides models of how packet data networks work and perform, and provides a simulation engine for users to conduct simulation experiments. Available model set in NS-3 focus on modeling how Internet protocols and networks work, but it is not limited to Internet system; several users use NS-3 to model non Internet based systems.

Question 2. What is the difference between simulation and emulation? Is simulation or emulation done by NS-3?

Simulation process copies something from the real world into a virtual environment often to give an idea about how something works or might work. It simulates the basic behavior but doesn't necessarily abide to all the rules of the real environment that it simulates.

Emulation process, on the other hand duplicates the thing exactly as it exists in real life. The emulation is efficiently as it exists in real life. The emulation is efficiently a complete imitation of the real thing – it just operates in a virtual environment instead of the real world.

NS-3 performs Simulation. It is also called discrete event network simulator. It tries to simulate how a network will perform if implemented practically, what will be its limitations and what will be its efficiency.

Question 3. What Does NS-3 Provide?

Features of NS-3:-

- It provides models of how packet data networks work and perform.
- It provides simulation engine to conduct simulation experiments.
- It provides scripting to be done in C++ or Python.
- It provides pcap packet trace files which can be used to analyze traces.
- It provides features not available in NS-2 like a implementation code execution environment.
- It provides a lower base level of abstraction, allowing it to align better with how real systems are put together.
- It provides its usage on Linux, on macOS systems, although support exists for Windows frameworks that can build Linux code too such as Cygwin.

Conclusion: We have successfully understood the basics about Network Simulators and Emulators and also read about NS-3 and its features.

ASSIGNMENT 2

Objective: Design and configure a simple network model. Collect Statistics and Analyze network performance.

Tasks Performed: Report the following-

1. Network Topology
2. Scenario (Interaction Between Input and Output)
3. Steps for Topology Creation and Configuration
4. Network Parameters (Simulation Time, Frames Generated, MAC Protocol Used etc.)
5. Network Performance Metrics (Throughput, Frames dropped etc.)

Steps Involved:

1. Network Topology

Here is an example of a topology where two nodes are connected point to point and one from them is connected to other three nodes via LAN (CSMA).

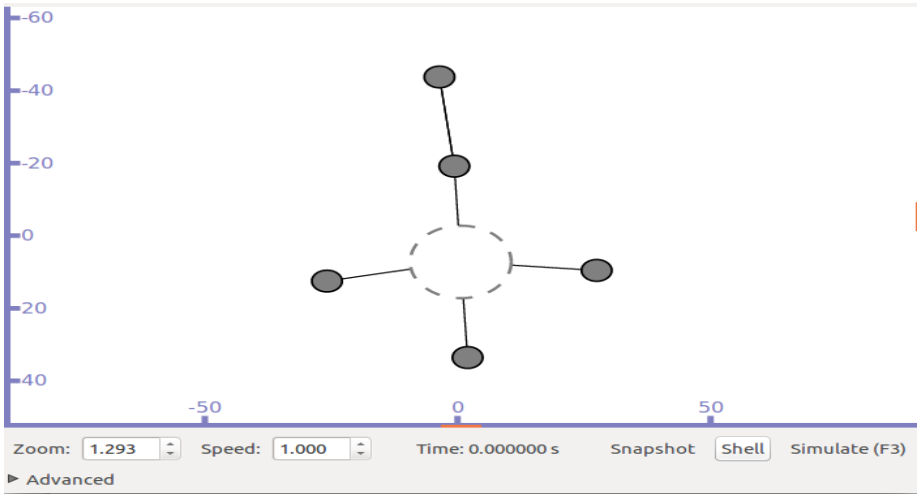

Figure 1 two nodes are connected point to point

2. Scenario

Devices are in a network where one device act as a client and the other as server. The client application is sending data packets to the server which are acknowledged by the server and he client receives acknowledgement response.

3. Steps for Topology Creation and Configuration

i. Import the necessary modules and header files
ii. Create a NodeContainer object and then use its Create method to create two point to point nodes.
iii. Create a PointToPointHelper object and set its DeviceAttribute("Channel Rate") and also set its ChannelAttribute("Delay")
iv. Create NetDeviceContainer object install the p2p nodes on it
v. Create another NodeContainer with one point to point node and three LAN nodes.
vi. Set device attributes for these nodes.
vii. Create an InternetStackHelper and install nodes on stack
viii. Create Ipv4AddressHelper and assign addresses to those nodes
ix. Using UdpEchoServerHelper create server app on one node
x. Using UdpEchoClientHelper install client on the other node
xi. Start and stop time for the client and server apps as required
xii. Call Simulator Run and Destroy methods

Detailed Simulation Parameters:

- Nodes - 5
- Data Rate - 5 Mbps
- Channel Delay - 2msec
- Base Address – 10.1.1.0,255.255.255.0 & 10.1.2.0,255.255.255.0
- Simulation Time - 10 sec
- Server Start Time - 1 sec
- Server Stop Time - 10 sec
- Client Start Time - 4 sec

- Client Stop Time - 10 sec
- Port Used - 9 , 49153
- Protocol Used - UDP
- Max Datagram Size - 65507 Bytes
- Max No. Of Packets - Variable
- Interval To Send Packets - Variable(in Seconds)
- Packet Size - Variable (In Bytes)
- The Setup is IPv4.

Results:

Output →

```
lenovo@lenovo-Lenovo-G580:~/ns3/ns-allinone-3.27/ns-3.27$ ./waf --run scratch/se
cond
Waf: Entering directory `/home/lenovo/ns3/ns-allinone-3.27/ns-3.27/build'
Waf: Leaving directory `/home/lenovo/ns3/ns-allinone-3.27/ns-3.27/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (2.084s)
At time 4s client sent 1024 bytes to 10.1.2.2 port 9
At time 4.01561s server received 1024 bytes from 10.1.1.1 port 49153
At time 4.01561s server sent 1024 bytes to 10.1.1.1 port 49153
At time 4.02721s client received 1024 bytes from 10.1.2.2 port 9
```

Figure 2 Log info

Network Performance Metrics→

Here 2 milli sec delay is for a packet = 0.002 sec but total delay is (2.00369 - 2) sec = 0.00369 sec

Travel/Propagation delay = 0.00169 sec

So, Throughput = (1024*8)/0.00369*1024*1024 = 2.117 Mbps

In this example no packets are dropped.

But, if either client or server stops before entire communication takes place, the packets are dropped or if time interval between packets is more , then frames are dropped.

Conclusion: We have successfully configured a simple network topology and analyzed it and also collected various statistics.

ASSIGNMENT 3

Objective→ Create 2 Wi-Fi Networks with separate access points, the access points being in a point to point connection. Send data packets between devices in these networks and also create trace file and analyze it via wireshark or tcpdump.

Tasks Performed→ Report the following-

1. Network Topology
2. Scenario (Interaction Between Input and Output)
3. Steps for Topology Creation and Configuration
4. Network Parameters

Steps Involved:→

1. Network Topology

Here is the topology where two devices/nodes act as Access Points and each of these access points has 3 more station nodes in its range.

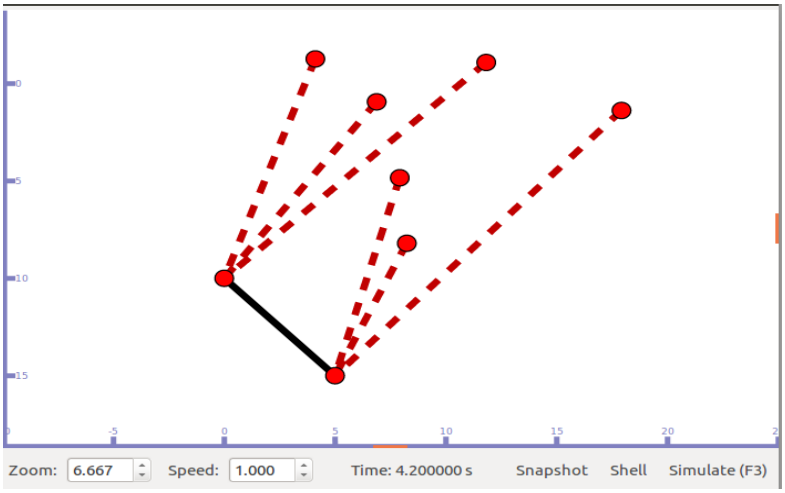

Figure 3 Network Topology

2. Scenario→

One station device in range of one access point act as server and another station device in range of other access point act as the client. The server and client nodes exchange data packets.

A trace file is also created of this transfer.

3. Steps for Topology Creation and Configuration→

In this, we create 2 separate Wi-Fi networks as follows→

i. Create 2 P2P nodes and define data rate and channel delay using PointToPoint Helper.
ii. Install the P2P nodes. These nodes acts as access points for our 2 Wi-Fi networks.
iii. Create Wi-Fi nodes for both networks.
iv. Set Wi-Fi Access point nodes from P2P nodes for both networks.
v. Create YansWifiChannelHelper and YansWiFiPhyHelper for both the networks.
vi. Create WifiMacHelper for both Wi-Fi networks, assign SSID to them and set MAC type.
vii. Install both mobile nodes as well as access points.
viii. Set mobility of the Wi-Fi networks, min-max coordinates and bounds.
ix. Create InternetStackHelper stack and install all nodes.
x. Create an IPv4 AddressHelper to assign address to each of the trace network nodes simultaneously creating variables for P2P and Wi-Fi interfaces.

Now we send data packet from client in one Wi-Fi network to server on another wifi network→

i. Create UDPEchoServerHelper and Install any one Wi-Fi node as server.
ii. Create UDPEchoClientHelper object and assign address of any node in another Wi-Fi network.
iii. Set address of server in client object to connect to server.
iv. Set attributes from client device like MaxPacket, Interval , Packet Size etc.

To Enable Tracing→

i. EnablePacpall for point to point nodes.
ii. EnablePcap for physical YansChannelHelper object

iii. After simulating this model, we will get seperate Pacp files for every node for which Pcap is enabled.

Detailed Simulation Parameters→

- Nodes – 2 APs with 3 STA nodes for each
- Access Points - 2
- Data Rate - 5 Mbps
- Channel Delay - 2msec
- Base Address - 10.1.1.0,255.255.255.0
- Simulation Time - as user wishes (variable)
- Server Start Time - 1 sec
- Server Stop Time - 10 sec
- Client Start Time - 2 sec
- Client Stop Time - 10 sec
- Port Used - 9 , 49153
- Protocol Used - UDP
- Max Datagram Size - 65507 Bytes
- Max No. Of Packets - Variable
- Interval To Send Packets - Variable(in Seconds)
- Packet Size - Variable (In Bytes)

Results→

Output

```
lenovo@lenovo-Lenovo-G580:~/ns3/ns-allinone-3.27/ns-3.27$ ./waf --run scratch/th
ird
Waf: Entering directory `/home/lenovo/ns3/ns-allinone-3.27/ns-3.27/build'
Waf: Leaving directory `/home/lenovo/ns3/ns-allinone-3.27/ns-3.27/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (2.164s)
At time 2s client sent 1024 bytes to 10.1.2.3 port 9
At time 2.018s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.018s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.04002s client received 1024 bytes from 10.1.2.3 port 9
lenovo@lenovo-Lenovo-G580:~/ns3/ns-allinone-3.27/ns-3.27$
```

Figure 4 Output Logs

Conclusion→ We have successfully configured the required network topology and analyzed it and also collected various statistics and a saved a pcap file for packet tracing.

ASSIGNMENT 4

Objective → Create a scenario for hidden & exposed terminal problem. Find out the packet drop in both of the scenarios. Also propose a solution (RTS/CTS mechanism)

Tasks Performed → Report the following-

1. Network Topology
2. Scenario (Interaction Between Input and Output) (with and without RTS/CTS mechanism)
3. Steps for Topology Creation and Configuration
4. Show difference in throughput and packet drop in both the cases
5. Find out throughput less than the without RTS/CTS mechanism
6. Find situation where throughput will be higher even with RTC/CTS
7. Calculate overhead caused by RTS/CTS packets
8. Network Parameters

Steps Involved:

1. **Network Topology**

Here is the topology where three devices/nodes act as Access Points and each of these access points has 3 more station nodes in its range

AP1 o----------o AP2 o----------o AP3

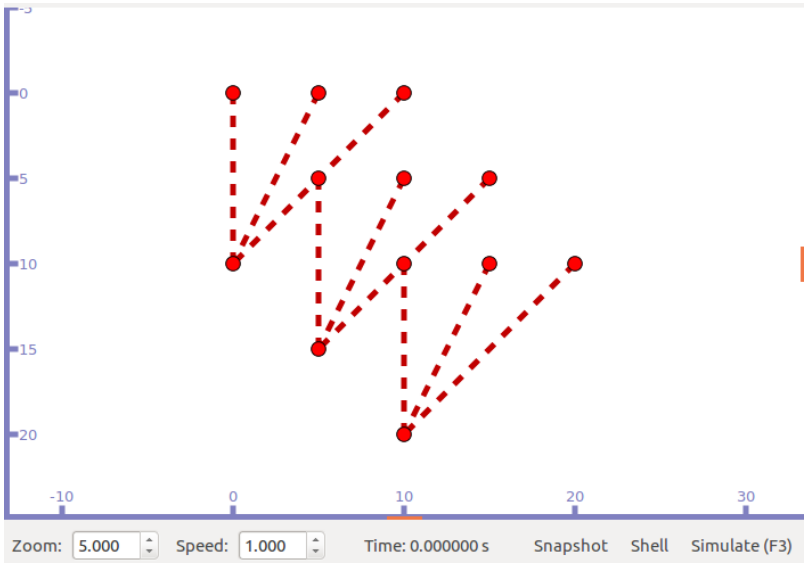

Figure 5 Wifi Network

2. **Scenario1 (Hidden Terminal)**

Hidden Terminal problem occurs when a node is not in range of the sender of data but is in the range of the receiver. When two such unaware nodes start to transmit data to the common receiver data collision occurs. To avoid this problem one can use RTS/CTS control packets.

So we create a scenario where in the above displayed topology, a node (client) from wifi network1 and another node (client) from wifi network3 send data simultaneously to a node (server) in wifi network2.

3. **Steps to create topology and send data**

In this, we create 3 separate wifi networks as follows:

i. Create 3 nodes that will act as access points (using NodeContainer).
ii. Create MobilityHelper and set PositionAllocator and mobility model as required (Here gridposition Allocator, Constant position mobility model).
iii. Create 3 WifiStaNodes with 3 nodes in each.
iv. Set previously created 3 nodes as AccessPoints Nodes.
v. Create YansWiFiChannelHelper , YansWifiPhyHelper for all 3 seperate networks and set channel to phy helper.
vi. Set SSID, RemoteStationManager , Mactype , Probing etc for 3 networks.
vii. Install WiFi Phy,Mac and StaNodes to StaDevices net device container.

viii. Create 3 ApWiFimac and install on access point devices Net Devices Container.

ix. Set previously created mobility helpers to StaNodes and ApNodes.

x. Now create 2 seprate Adhoc networks with AP1-AP2 & AP2-AP3.

xi. Create Seprate YansWiFiChannel , WiFiHelper , YansphysicalHelper and WiFimacHelper for these 2 adhoc networks.

xii. Set standard (WiFi-Phy-Standard-802.11b).Set RemoteStationManager and mobility model to these 2 new networks.

xiii. Create InternetStackHelper stack and install all these nodes.

xiv. Set address & assign them to nodes for each of the network created.

xv. Now create an OnOffHelper on ST1 of AP2 & install client OnOffHelper Apps on ST1 of AP1 & ST1 of AP2 both.

xvi. Set different attributes like Data Rate , Time , Packet Size etc.

xvii. Create a FlowMonitor to get the FlowStates.

xviii. Extract & display the stats at the end.

xix. Enable the log components for the on-off Helper.

xx. Set simulator::Run() & simulator::Destroy() ;.

Adding RTS/CTS Mechanism

i. RTS/CTS can be enabled on setting RTSCTSThreshold to be greater than the packet size

ii. To enable Add :
Config :: setdefault ("ns3::WiFiRemoteStationManager::RTSCTSThreshold",CTSThrvalue)

Detailed Simulation Parameters→

- RTSCTSThr Value - 2200/10
- Probing Used - Active Probing
- ConstantPosition Mobility model used for ApNodes.
- RandomWalk2dMobility model used for StaNodes.
- Propogation Loss - Fixed for each pair of Nodes & doesnot depend on their actual positions.
- For Data Mode DSSS rate - 2Mbps
- for Control Mode DSSS rate - 1 Mbps
- Packet Size - 1400 bytes

Results :

Output→ Comparison in RTS/CTS enabled & disabled

```
RtsCts is disabled

Data going from IP Address 10.1.2.1 to IP Address 10.1.3.3

  Tx Packets: 2410
  Tx Bytes:   3441480
  Rx Packets: 1086
  Rx Bytes:   1550808
  Throughput: 1.3785 Mbps

Data going from IP Address 10.1.4.1 to IP Address 10.1.3.3

  Tx Packets: 2410
  Tx Bytes:   3441480
  Rx Packets: 1091
  Rx Bytes:   1557948
  Throughput: 1.38484 Mbps


RtsCts is enabled

Data going from IP Address 10.1.2.1 to IP Address 10.1.3.3

  Tx Packets: 2410
  Tx Bytes:   3441480
  Rx Packets: 1138
  Rx Bytes:   1625064
  Throughput: 1.4445 Mbps

Data going from IP Address 10.1.4.1 to IP Address 10.1.3.3

  Tx Packets: 2410
  Tx Bytes:   3441480
  Rx Packets: 1135
  Rx Bytes:   1620780
  Throughput: 1.44069 Mbps
```

Figure 6

4. Scenario #2 (Exposed Terminal)

Exposed Terminal problem occurs when a node is communicating with a node and due to this some other node is unable to communicate with some different node because the former node is in range of one of the nodes which are communicating. The node becomes exposed to its neighbour node. To avoid this problem one can use RTS/CTS control packets.

So we create a scenario where = , a node (client) from wifi network2 send data to a node (server) in wifi network1, and a node (client) from wifi network3 wants to send data to a node (server) in wifi network4.

Scenario can be generated following the similar steps as that for hidden terminal problem.

AP1----------AP2------------AP3------------AP4

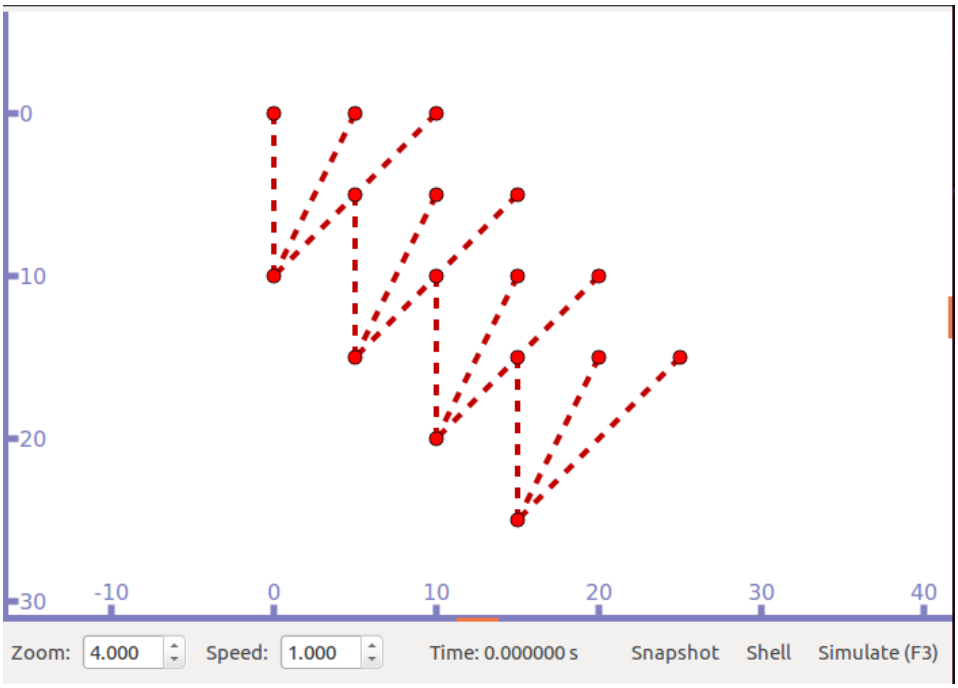

Figure 7

Using similar simulation parameters as that for hidden terminal scenario.

Results→

```
RtsCts is disabled

Data going from IP Address 10.1.3.1 to IP Address 10.1.2.3

  Tx Packets: 2410
  Tx Bytes:   3441480
  Rx Packets: 1089
  Rx Bytes:   1555092
  Throughput: 1.3823 Mbps

Data going from IP Address 10.1.4.1 to IP Address 10.1.7.3

  Tx Packets: 2410
  Tx Bytes:   3441480
  Rx Packets: 1092
  Rx Bytes:   1559376
  Throughput: 1.38611 Mbps


RtsCts is enabled

Data going from IP Address 10.1.3.1 to IP Address 10.1.2.3

  Tx Packets: 2410
  Tx Bytes:   3441480
  Rx Packets: 1132
  Rx Bytes:   1616496
  Throughput: 1.43689 Mbps

Data going from IP Address 10.1.4.1 to IP Address 10.1.7.3

  Tx Packets: 2410
  Tx Bytes:   3441480
  Rx Packets: 1132
  Rx Bytes:   1616496
  Throughput: 1.43689 Mbps
```

Figure 8

Observations→

- Throughput is less without RTS/CTS mechanism. This is so because with RTS/CTS there are excessive collisions as the stations remain hidden and cannot sense if the channel is free or not.
- Throughput can be lower even when RTS/CTS is applied if there are successive continuous collisions among these RTS/CTS control packets.
- The overhead caused by RTS/CTS packets amounts to the No. of RTS , CTS packets exchanged and the size of these packets.
- Minimum Overhead = (RTS+CTS) * No. of Packets to be transferred assuming no RTS , CTS collisions

Conclusion→ We have successfully configured the required network topology and analyzed it and also collected various statistics and a saved a pcap file for packet tracing.

ASSIGNMENT 5

Objective→ Create a Wireless Sensor Network of 10 Nodes using IEEE 802.15.4 protocol where 9 out of these nodes are sending hello message (ping) to the 10 nodes

Tasks Performed→ Report the following-

1. Network Topology
2. Scenario (Interaction Between Input and Output)
3. Steps for Topology Creation and Configuration
4. Network Parameters

Steps Involved→

1. Network Topology

Here is the topology where ten devices/nodes are in IEEE 802.15.4 Wireless Network

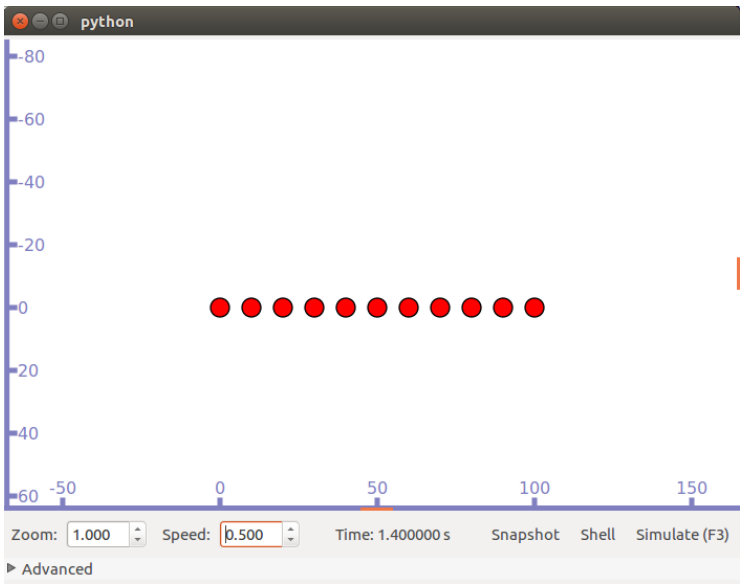

Figure 9

2. Scenario

One station n0 acts as the server and all the other nodes will act as clients and will send hello message to node n0

3. Steps for Topology Creation and Configuration

In this, we create 2 separate wifi networks as follows :

i. Use NodeContainer object to create 10 nodes
ii. Set mobility model for the nodes
iii. Create LrWpan Helper object and Install nodes to it
iv. Associate a PAN for the network
v. Create Internet Stack Helper , set base address and install nodes on this stack
vi. Create 9 Ping6Helper and 9 Application Containers for each of these nodes
vii. Set local node and remote node for each of these nodes
viii. Also set packet size , max packet count and packet interval
ix. Set time to start and stop these apps
x. Call Simulator Run and Destroy

Detailed Simulation Parameters→

- Nodes – 10
- Base Address – Ipv6 2001:1::
- Simulation Time - as user wishes (variable)
- Max No. Of Packets - 5
- Interval To Send Packets - 1(in Seconds)
- Packet Size - 10 (In Bytes)
- The Setup is IPv6

Results→

Output→

```
Created 11 devices
There are 11 nodes
```

Figure 10

```
Ping6Application:ScheduleTransmit(0x11bbc50, +0.0ns)
Ping6Application:Send()
Sent 18 bytes to 2001:1::ff:fe00:1
Ping6Application:Send()
Sent 18 bytes to 2001:1::ff:fe00:1
Ping6Application:Send()
Sent 18 bytes to 2001:1::ff:fe00:1
Ping6Application:Send()
Sent 18 bytes to 2001:1::ff:fe00:1
Ping6Application:Send()
Sent 18 bytes to 2001:1::ff:fe00:1
Ping6Application:Send()
Sent 18 bytes to 2001:1::ff:fe00:1
Ping6Application:Send()
Sent 18 bytes to 2001:1::ff:fe00:1
Ping6Application:Send()
Sent 18 bytes to 2001:1::ff:fe00:1
Ping6Application:Send()
Sent 18 bytes to 2001:1::ff:fe00:1
Ping6Application:Send()
Sent 18 bytes to 2001:1::ff:fe00:1
Ping6Application:HandleRead(0x11bb530, 0x110c2b0)
Ping6Application:HandleRead(0x11bae10, 0x110a9d0)
Ping6Application:HandleRead(0x11bb2d0, 0x110c160)
Ping6Application:HandleRead(0x11bb790, 0x1107d50)
Ping6Application:HandleRead(0x11bb9f0, 0x1107ea0)
Ping6Application:HandleRead(0x11babf0, 0x11c1d80)
Ping6Application:HandleRead(0x11bbc50, 0x1111f60)
Ping6Application:HandleRead(0x11bb530, 0x110c2b0)
Received Echo Reply size  = 10 bytes from 2001:1::ff:fe00:1 id =  48879 seq = 0 Hop Count = 0
Ping6Application:HandleRead(0x11bae10, 0x110a9d0)
Received Echo Reply size  = 10 bytes from 2001:1::ff:fe00:1 id =  48879 seq = 0 Hop Count = 0
Ping6Application:HandleRead(0x11bb2d0, 0x110c160)
Received Echo Reply size  = 10 bytes from 2001:1::ff:fe00:1 id =  48879 seq = 0 Hop Count = 0
Ping6Application:HandleRead(0x11bb790, 0x1107d50)
Received Echo Reply size  = 10 bytes from 2001:1::ff:fe00:1 id =  48879 seq = 0 Hop Count = 0
Ping6Application:HandleRead(0x11bb9f0, 0x1107ea0)
Received Echo Reply size  = 10 bytes from 2001:1::ff:fe00:1 id =  48879 seq = 0 Hop Count = 0
Ping6Application:HandleRead(0x11babf0, 0x11c1d80)
Received Echo Reply size  = 10 bytes from 2001:1::ff:fe00:1 id =  48879 seq = 0 Hop Count = 0
Ping6Application:HandleRead(0x11bbc50, 0x1111f60)
Received Echo Reply size  = 10 bytes from 2001:1::ff:fe00:1 id =  48879 seq = 0 Hop Count = 0
Ping6Application:HandleRead(0x11ba9d0, 0x11c1c70)
```

```
Ping6Application:HandleRead(0x11ba9d0, 0x11c1c70)
Ping6Application:HandleRead(0x11ba7e0, 0x11b9870)
Ping6Application:HandleRead(0x11ba7e0, 0x11b9870)
Received Echo Reply size  = 10 bytes from 2001:1::ff:fe00:1 id =  48879 seq = 0 Hop Count = 0
Ping6Application:HandleRead(0x11ba9d0, 0x11c1c70)
Received Echo Reply size  = 10 bytes from 2001:1::ff:fe00:1 id =  48879 seq = 0 Hop Count = 0
Ping6Application:HandleRead(0x11bb070, 0x110ab00)
Ping6Application:HandleRead(0x11bb070, 0x110ab00)
Received Echo Reply size  = 10 bytes from 2001:1::ff:fe00:1 id =  48879 seq = 0 Hop Count = 0
```

Figure 11

ASSIGNMENT 6

Objective→ Use AODV and DSDV for routing in networks and analyze the network and protocol performance.

AODV

Tasks Performed→ Report the following-

1. Network Topology
2. Scenario (Interaction Between Input and Output)
3. Steps for Topology Creation and Configuration
4. Network Parameters

Steps Involved→

1. Network Topology

Here is the topology where 4 nodes are connected

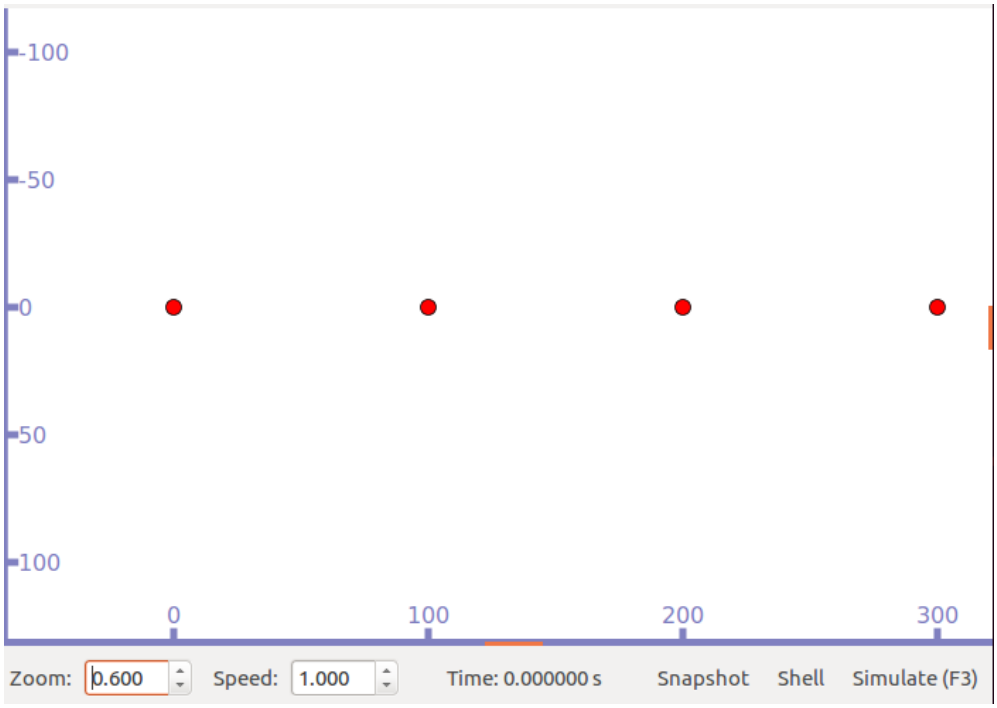

Figure 12

2. Scenario

Four nodes are connected in 1 dimension grid topology and leftmost node wants to send data to rightmost node for which it needs to first find a route.

3. Steps for Topology Creation and Configuration

In this, we create network as follows :

i. Use NodeContainer object to create 4 nodes placed 50m apart.
ii. Set mobility model for the nodes
iii. Create YansWifiChannelHelper and YansWiFiPhyHelper for the network.
iv. Create WifiMacHelper for network , assign SSID and set MAC type.
v. Install wifi onto nodes.
vi. Create InternetStackHelper stack, set aodv routing helper and install all nodes.
vii. Create an IPv4 AddressHelper to assign address to each of the nodes.
viii. Create ping application with server at rightmost node and a client at leftmost node.
ix. Also set parameters like packet size , max packet count and packet interval.
x. Set time to start and stop the application.
xi. Call Simulator Run and Destroy.

Detailed Simulation Parameters→

- MAC Protocol: 802.11
- Network simulation area: 300m
- Bandwidth: 6Mbps
- Source Node: Node 1 (10.0.0.1)
- Destination Node: Node 4 (10.0.0.4)

- Data Packet Size: 64 bytes
- Number of packets sent in each session: 10
- Simulation length: 10 seconds

Results→

Route Discovery time : 1.217525 s

Packet Loss Performance: 0%

End to end Delay (Round trip times): min/avg/max = 2/150.7/1239 ms

Throughput: 3kbps

```
Creating 4 nodes 100 m apart.
Starting simulation for 10 s ...
PING  10.0.0.4 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=0 ttl=62 time=1239 ms
64 bytes from 10.0.0.4: icmp_seq=1 ttl=62 time=241 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=62 time=3 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=62 time=3 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=62 time=6 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=62 time=3 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=62 time=3 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=62 time=2 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=62 time=3 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=62 time=4 ms
--- 10.0.0.4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9999ms
rtt min/avg/max/mdev = 2/150.7/1239/389.6 ms
```

Figure 13

```
Node: 0; Time: +4.0s, Local time: +4.0s, AODV Routing table

AODV Routing table
Destination Gateway   Interface Flag  Expire    Hops
10.0.0.2  10.0.0.2  10.0.0.1  UP  2.21            1
10.0.0.4  10.0.0.2  10.0.0.1  UP  2.00            3
10.255.255.255  10.255.255.255  10.0.0.1  UP  9223372032.85   1
127.0.0.1 127.0.0.1 127.0.0.1 UP  9223372032.85   1


Node: 1; Time: +4.00s, Local time: +4.00s, AODV Routing table

AODV Routing table
Destination Gateway   Interface Flag  Expire    Hops
10.0.0.1  10.0.0.1  10.0.0.2  UP  2.20            1
10.0.0.3  10.0.0.3  10.0.0.2  UP  2.02            1
10.0.0.4  10.0.0.3  10.0.0.2  UP  2.00            2
10.255.255.255  10.255.255.255  10.0.0.2  UP  9223372032.85   1
127.0.0.1 127.0.0.1 127.0.0.1 UP  9223372032.85   1
```

Figure 14

```
Node: 2; Time: +4.00s, Local time: +4.00s, AODV Routing table

AODV Routing table
Destination Gateway   Interface Flag  Expire    Hops
10.0.0.1  10.0.0.2  10.0.0.3  UP  2.65            2
10.0.0.2  10.0.0.2  10.0.0.3  UP  2.21            1
10.0.0.4  10.0.0.4  10.0.0.3  UP  2.02            1
10.255.255.255  10.255.255.255  10.0.0.3  UP  9223372032.85   1
127.0.0.1 127.0.0.1 127.0.0.1 UP  9223372032.85   1


Node: 3; Time: +4.00s, Local time: +4.00s, AODV Routing table

AODV Routing table
Destination Gateway   Interface Flag  Expire    Hops
10.0.0.1  10.0.0.3  10.0.0.4  UP  2.66            3
10.0.0.3  10.0.0.3  10.0.0.4  UP  2.02            1
10.255.255.255  10.255.255.255  10.0.0.4  UP  9223372032.85   1
127.0.0.1 127.0.0.1 127.0.0.1 UP  9223372032.85   1
```

Figure 15

DSDV

Network Topology, scenario and topology creation and configuration is same as that of AODV.

Detailed Simulation Parameters→

- MAC Protocol: 802.11
- Bandwidth: 11Mbps
- Source Node: Node 1 (10.0.0.1)
- Destination Node: Node 4 (10.0.0.4)
- Data Packet Size: 1000 bytes
- Number of packets sent in each session: 10
- Simulation length: 20 seconds

Results→

Packet Delivery Ratio : 1

Routing Load: 9

Average End to end Packet Delivery Time: 737.47 ms

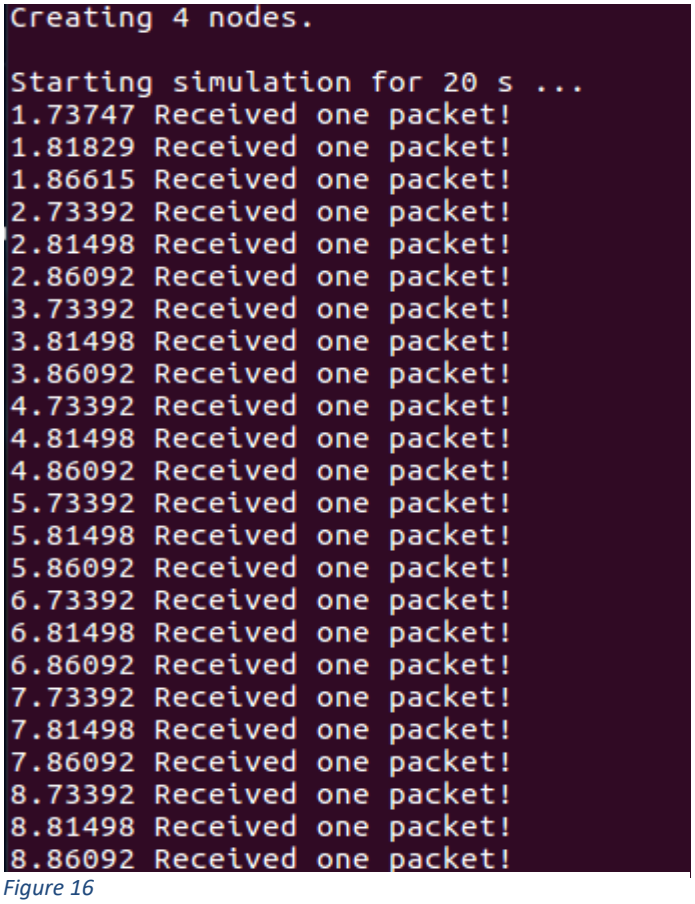

Figure 16

```
Node: 0, Time: +2.0s, Local time: +2.0s, DSDV Routing table

DSDV Routing table
Destination   Gateway   Interface   HopCount    SeqNum    LifeTime    SettlingTime
10.1.1.2    10.1.1.2    10.1.1.1    1           2           1.998s    2.000s
10.1.1.3    10.1.1.3    10.1.1.1    1           2           1.996s    2.000s
10.1.1.4    10.1.1.4    10.1.1.1    1           2           1.997s    2.000s
10.1.1.255    10.1.1.255    10.1.1.1    0           2           -9223372034.855s    0.000s
127.0.0.1   127.0.0.1   127.0.0.1   0           0           -9223372034.855s    0.000s


Node: 1, Time: +2.000s, Local time: +2.000s, DSDV Routing table

DSDV Routing table
Destination   Gateway   Interface   HopCount    SeqNum    LifeTime    SettlingTime
10.1.1.1    10.1.1.1    10.1.1.2    1           2           1.996s    2.000s
10.1.1.3    10.1.1.3    10.1.1.2    1           2           1.996s    2.000s
10.1.1.4    10.1.1.4    10.1.1.2    1           2           1.997s    2.000s
10.1.1.255    10.1.1.255    10.1.1.2    0           2           -9223372034.855s    0.000s
127.0.0.1   127.0.0.1   127.0.0.1   0           0           -9223372034.855s    0.000s
```

Figure 17

```
Node: 2, Time: +2.000s, Local time: +2.000s, DSDV Routing table

DSDV Routing table
Destination   Gateway   Interface   HopCount    SeqNum    LifeTime    SettlingTime
10.1.1.1    10.1.1.1    10.1.1.3    1           2           1.996s    2.000s
10.1.1.2    10.1.1.2    10.1.1.3    1           2           1.998s    2.000s
10.1.1.4    10.1.1.4    10.1.1.3    1           2           1.997s    2.000s
10.1.1.255    10.1.1.255    10.1.1.3    0           2           -9223372034.855s    0.000s
127.0.0.1   127.0.0.1   127.0.0.1   0           0           -9223372034.855s    0.000s


Node: 3, Time: +2.000s, Local time: +2.000s, DSDV Routing table

DSDV Routing table
Destination   Gateway   Interface   HopCount    SeqNum    LifeTime    SettlingTime
10.1.1.1    10.1.1.1    10.1.1.4    1           2           1.996s    2.000s
10.1.1.2    10.1.1.2    10.1.1.4    1           2           1.998s    2.000s
10.1.1.3    10.1.1.3    10.1.1.4    1           2           1.996s    2.000s
10.1.1.255    10.1.1.255    10.1.1.4    0           2           -9223372034.855s    0.000s
127.0.0.1   127.0.0.1   127.0.0.1   0           0           -9223372034.855s    0.000s
```

Figure 18

Conclusion → We have successfully analyzed network performance for AODV and DSDV routing protocols.