

A Major Project Report  
On  
**“PERFORMANCE ANALYSIS BETWEEN OLSR AND DSR ROUTING  
PROTOCOLS USING NS3 SIMULATOR”**

Submitted in partial fulfilment of the requirements for the A80088 a

Major Project Report  
in  
**Computer Science and Engineering**

Submitted

By

**Mr. A. RAVI DUTT**

**H. T. No: 15261A05G7**

**Mr. S. PRUDHVIRAJ**

**H. T. No: 15261A05H3**

Under the Guidance of

**Ms. Vijayalaxmi C Handaragall**

**(Assistant Professor)**

**Ms. C. Sudha**

**(Assistant Professor)**



**Department of Computer Science and Engineering**  
**MAHATMA GANDHI INSTITUTE OF TECHNOLOGY**  
**(Affiliated to Jawaharlal Nehru Technological University Hyderabad)**  
**GANDIPET, HYDERABAD – 500 075. Telangana (INDIA)**

# MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(Affiliated to Jawaharlal Nehru Technological University Hyderabad)

GANDIPET, HYDERABAD – 500 075. Telangana (INDIA)

## CERTIFICATE



This is to certify that the project entitled “**PERFORMANCE ANALYSIS BETWEEN OLSR AND DSR ROUTING PROTOCOLS USING NS3 SIMULATOR**”, being submitted by **Mr. A. RAVI DUTT** bearing **Roll No: 15261A05G7** and **Mr. S. PRUDHVI RAJ** bearing **Roll No: 15261A05H3** in partial fulfilment of the requirements for the A80088 a Major Project Report in Computer Science and Engineering is a record of bonafide work carried out by them. The Results of investigations enclosed in this report have been verified and found satisfactory.

Project Guides

**Ms. Vijayalaxmi C Handaragall**

**(Assistant Professor, CSE)**

**Ms. C. Sudha**

**(Assistant Professor, CSE)**

**Dr. C R K Reddy**

Head of Department

Computer Science and Engineering

**External Examiner**

## **ACKNOWLEDGEMENT**

We would like to express our sincere thanks to **Dr. K. Jaya Sankar, Principal MGIT**, for providing the working facilities in college.

We wish to express our sincere thanks and gratitude to **Dr. C R K Reddy, Professor and HOD**, Department of CSE, MGIT, for all the timely support and valuable suggestions during the period of project.

We are extremely thankful to **Dr. M. Rama Bai, Professor, Dr. K. Sreekala, Assistant Professor** and **Mrs. J. Sreedevi, Assistant Professor**, Department of CSE, MGIT, major project coordinators for their encouragement and support throughout the project.

We are extremely thankful and indebted to our internal guides **Ms. Vijayalaxmi C Handaragall, Assistant Professor** and **Ms. C. Sudha, Assistant Professor**, Department of CSE, for their constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank all the faculty and staff of CSE Department who helped us directly or indirectly, for completing this project.

**A. RAVI DUTT**

**(15261A05G7)**

**S. PRUDHVIRAJ**

**(15261A05H3)**

## **ABSTRACT**

The main objective of this project is to show a difference between both reactive and proactive routing protocols of Mobile Ad-hoc Network (MANET). These protocols have a wide area of applications in Wireless Sensor Networks which are helpful to traverse in different networking systems. In Proactive routing protocols every node maintains one or more tables representing the entire topology of the network and traverse to the destination node whereas the reactive routing protocols are used to set up routes on demand to traverse to the desired destination node. The performance analysis for these protocols are compared between Optimized Link State Routing Protocol (OLSR) and Dynamic Source Routing Protocol (DSR).

The measurements for these protocols can be analysed through a varied set of parameters such as simulation time, number of sinks, packets received, End to End Delay, Throughput. Then this project gives a conclusion on which protocol is better suitable for different conditions and gives a conclusion when to use proactive and reactive routing protocols accordingly. To visualize all these operations, this project uses a tool to simulate the protocols which is known as Network Simulator 3 (NS3) which handles working of different networking protocols.

# LIST OF FIGURES

| <b>Figure No.</b> | <b>Figure Name</b>                                       | <b>Page No.</b> |
|-------------------|--|-----------------|
| 1.1               | OLSR Protocol Header and Message Format                  | 1               |
| 1.2               | DSR Route Request Option Format                          | 3               |
| 1.3               | DSR Route Reply Option Format                            | 3               |
| 1.4               | DSR Route Error Option Format                            | 4               |
| 3.1.1             | System Architecture of the application                   | 11              |
| 3.3.1.1           | State Chart Diagram                                      | 16              |
| 3.3.2.1           | Sequence Diagram   | 17              |
| 3.3.3.1           | Use Case Diagram   | 18              |
| 4.1.1             | Simulation of OLSR Protocol                              | 19              |
| 4.1.2             | Simulation data of OLSR Protocol using Trace Metrics     | 20              |
| 4.1.3             | Individual node performance evaluation for OLSR Protocol | 20              |
| 4.1.4             | Throughput evaluation of each node for OLSR Protocol     | 21              |
| 4.1.5             | Simulation of DSR Protocol                               | 21              |
| 4.1.6             | Simulation data of DSR Protocol using Trace Metrics      | 22              |
| 4.1.7             | Individual node performance evaluation for DSR Protocol  | 22              |
| 4.1.8             | Throughput evaluation of each node for DSR Protocol      | 23              |
| 4.1.9             | Simulation of OLSR Protocol with 20 nodes                | 23              |
| 4.1.10            | Generated CSV file for OLSR Protocol                     | 24              |
| 4.1.11            | Simulation of DSR Protocol with 20 nodes                 | 25              |
| 4.1.12            | Generated CSV file for DSR Protocol                      | 25              |
| 4.2.1.1           | NS3 Scratch folder for developing protocols              | 26              |
| 4.2.2.1           | Command for building the OLSR and DSR Protocols          | 26              |
| 4.2.3.1           | Simulation of routing protocols in python visualizer     | 27              |
| 4.2.4.1           | Launching the Trace Metrics Application                  | 27              |
| 4.2.4.2           | Locating the trace file in Trace Metrics                 | 28              |
| 4.2.5.1           | Chosen trace file details for execution                  | 28              |
| 4.2.5.2           | Executing the analysis of selected trace file            | 29              |

|         |   |    |
|---------|---|----|
| 4.2.6.1 | Output of generated trace file                                  | 29 |
| 4.2.6.2 | Individual node details generated from trace file               | 30 |
| 4.2.6.3 | Calculated throughput values of individual node                 | 30 |
| 4.2.7.1 | Simulation Time v/s Throughput with number of sinks as 1        | 31 |
| 4.2.7.2 | Simulation Time v/s Throughput with number of sinks as 5        | 31 |
| 4.2.7.3 | Simulation Time v/s Throughput with number of sinks as 10       | 32 |
| 4.2.7.4 | Node Ratio v/s Throughput                                       | 32 |
| 4.2.7.5 | Simulation Time v/s End to End Delay with number of sinks as 5  | 33 |
| 4.2.7.6 | Simulation Time v/s End to End Delay with number of sinks as 10 | 33 |
| 4.2.7.7 | Node Ratio v/s End to End Delay                                 | 34 |

## LIST OF TABLES

| Table No. | Table Name  | Page No. |
|-----------|---|----------|
| 2         | Summarization of different MANET Routing Protocols              | 9        |
| 4.1.1     | Simulating OLSR protocol with 5 nodes                           | 19       |
| 4.1.2     | Evaluating OLSR protocol using trace file                       | 19       |
| 4.1.3     | Simulating DSR protocol with 5 nodes                            | 21       |
| 4.1.4     | Evaluating DSR protocol using trace file                        | 22       |
| 4.1.5     | Measuring performance of OLSR protocol using generated CSV file | 23       |
| 4.1.6     | Measuring performance of DSR protocol using generated CSV file  | 24       |

# INDEX

| <b>Sr. No</b> | <b>Topic</b>  | <b>Page No.</b> |
|---------------|---|-----------------|
|               | Abstract  | i               |
|               | List of Figures   | ii              |
|               | List of Tables  | iv              |
| 1.            | Introduction  | 1               |
|               | 1.1 Problem Definition                                      | 5               |
|               | 1.2 Existing System   | 5               |
|               | 1.2.1 Disadvantages of Existing System                      | 6               |
|               | 1.3 Proposed System   | 6               |
|               | 1.3.1 Advantages of Proposed System                         | 6               |
|               | 1.4 Requirements Specification                              | 7               |
|               | 1.4.1 Software Requirements                                 | 7               |
|               | 1.4.2 Hardware Requirements                                 | 7               |
| 2.            | Literature Survey on MANET Protocols                        | 8               |
| 3.            | Methodology for evaluating performance between OLSR and DSR | 11              |
|               | Routing Protocols   |                 |
|               | 3.1 Architecture  | 11              |
|               | 3.2 Modules   | 12              |
|               | 3.2.1 Development of OLSR Routing Protocol                  | 12              |
|               | 3.2.2 Development of DSR Routing Protocol                   | 13              |
|               | 3.2.3 Performance evaluation of OLSR and DSR Routing        | 15              |
|               | Protocols   |                 |
|               | 3.3 UML Diagrams  | 16              |



|    |   |    |
|----|---|----|
|    | 3.3.1 State Chart Diagram                                       | 16 |
|    | 3.3.2 Sequence Diagram  | 17 |
|    | 3.3.3 Use Case Diagram  | 18 |
| 4. | Testing and Results   | 19 |
|    | 4.1 Test Cases  | 19 |
|    | 4.2 Output Screens  | 26 |
|    | 4.2.1 Launching the NS3 scratch folder for developing protocols | 26 |
|    | 4.2.2 Command for building OLSR and DSR protocols               | 27 |
|    | 4.2.3 Simulation in Python Visualizer                           | 27 |
|    | 4.2.4 Launching of Trace Metrics for evaluation of trace file   | 28 |
|    | 4.2.5 Executing the analysis of selected trace file             | 29 |
|    | 4.2.6 Output of the trace file generated after execution        | 29 |
|    | 4.2.7 Graphical representation of the obtained results          | 31 |
| 5. | Conclusion  | 35 |
|    | Bibliography  | 36 |
|    | Appendix  | 37 |

# 1. INTRODUCTION

## Optimized Link State Routing Protocol

The Optimized Link State Routing (OLSR) is a table-driven, proactive routing protocol developed for MANETs. It is an optimization of pure link state protocols in that it reduces the size of control packet as well as the number of control packets transmission required. OLSR reduces the control traffic overhead by using Multipoint Relays (MPR), [4] which is the key idea behind OLSR. An MPR is a node's one-hop neighbour which has been chosen to forward packets. Instead of pure flooding of the network, packets are just forwarded by a node's MPRs. This delimits the network overhead, thus being more efficient than pure link state routing protocols. OLSR is well suited to large and dense mobile networks.

Because of the use of MPRs, the larger and more the dense a network, the more optimized link state routing is achieved. MPRs helps providing the shortest path to a destination. The only requirement is that all MPRs declare the link information for their MPR selectors (i.e., the nodes who has chosen them as MPRs). The network topology information is maintained by periodically exchange link state information. [4] If more reactivity to topological changes is required, the time interval for exchanging of link state information can be reduced.

|                    |           |                         |
|--------------------|-----------|-------------------------|
| 0                  | 15        | 31                      |
| Packet Length      |           | Packet Sequence Number  |
| Message Type       | Vtime     | Message Size            |
| Originator Address |           |                         |
| TTL                | Hop Count | Message Sequence Number |
| Data               |           |                         |

Fig 1.1: OLSR Protocol Header and Message Format

In fig 1.1, it shows a visualization of 32-bit OLSR header and message formats that is built for showing the path of communication between the routers in that network. Below is the detailed explanation of every field involved in the OLSR Protocol header and message format.

**Packet Length:** Size of the packet in bytes (16 bits).

**Packet Sequence Number:** It specifies the sequence of data to be sent (16 bits).

**Message Type:** Specifies what is the type of message to be forwarded (8 bits, 0 to 255).

**Vtime:** Validity time indicates for how long time after reception a node must consider the information in the message as valid, unless a more recent update to the information is received (8 bits). [1]

**Message Size:** The length of the message header and data in bytes (16 bits).

**Originator Address:** Contains the main address of the node which originally generated this message (32 bits).

**Time to Live (TTL):** Contains the maximum number of hops a message will be transmitted. (8 bits), 0 to 255.

**Hop Count:** Contains the number of hops a message has attained.

### **Dynamic Source Routing**

Dynamic Source Routing (DSR) is a self-maintaining routing protocol for wireless networks. The protocol can also function with cellular telephone systems and mobile networks with up to about 200 nodes. A Dynamic Source Routing network can configure and organize itself independently of oversight by [2] human administrators.

In Dynamic Source Routing, each source determines the route to be used in transmitting its packets to selected destinations. There are two main components, called Route Discovery and Route Maintenance. Route discovery determines the optimum path for transmission between a given source and destination which are route request and route reply.

Route maintenance (route error) ensures that the transmission path remains optimum and loop-free as network conditions change, even if this requires changing the route during transmission. The route maintenance path also ensures the source routes if there are any network disconnections within the system to obtain a new route.

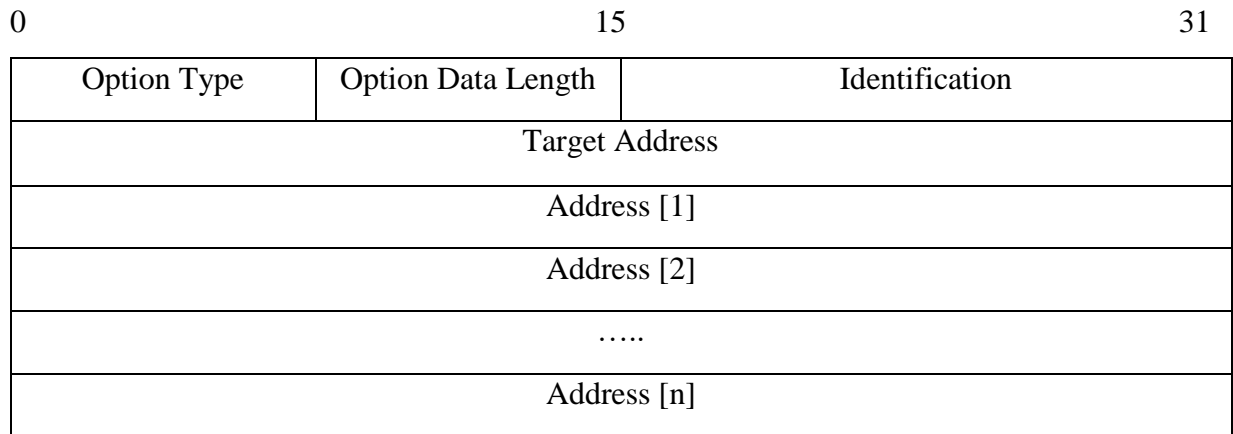


Fig 1.2: DSR Route Request Option Format

In fig 1.2, it shows a visualization of 32-bit DSR route request option format for source initiating communication. Below is [2] the detailed explanation of every field involved in the DSR Route Request Option format.

**Option Type:** Nodes not understanding this option will ignore it (8 bits).

**Option Data Length:** Length of the option, in octets (8 bits unsigned integer).

**Identification:** A unique value generated by the initiator of the route request (16 bits).

**Target Address:** The address of the node that is the target of the route request (32 bits).

**Address [1..n]:** Address[i] is the IPv4 address of the i-th node recorded in the route request option (32 bits).

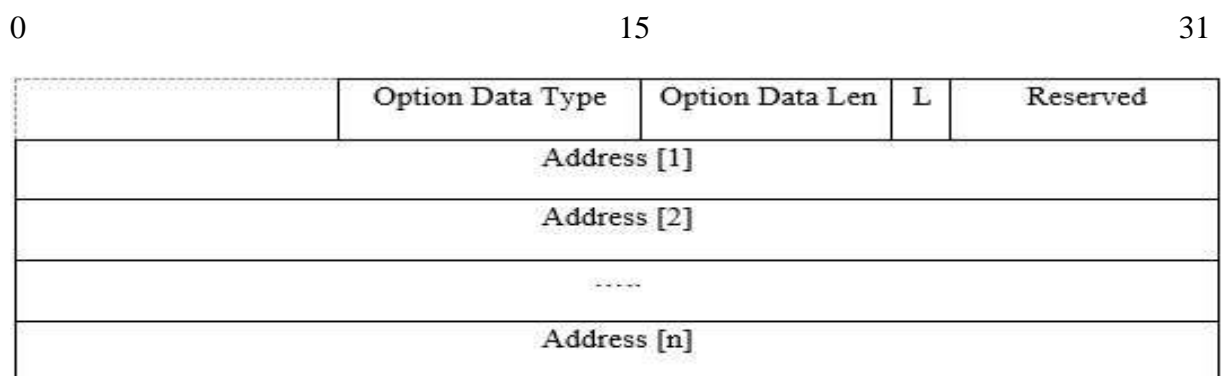


Fig 1.3: DSR Route Reply Option Format

In fig 1.3, it shows a visualization of 32-bit DSR route reply option format from destination to the source. [7] Below is the detailed explanation of every field involved in the DSR Route Reply option format.

**Option Data Type:** Nodes not understanding this option will ignore it (8 bits).

**Option Data Length:** Length of the option, in octets, excluding the option type and option data length fields (8 bits).

**Last Hop External (L):** Set to indicate that the last hop given by the route reply is an arbitrary path in a network external to the DSR network (1 bit).

**Reserved:** Must be sent as 0 and ignored on reception (7 bits).

**Address [1..n]:** The source route being returned by the route reply (32 bits).

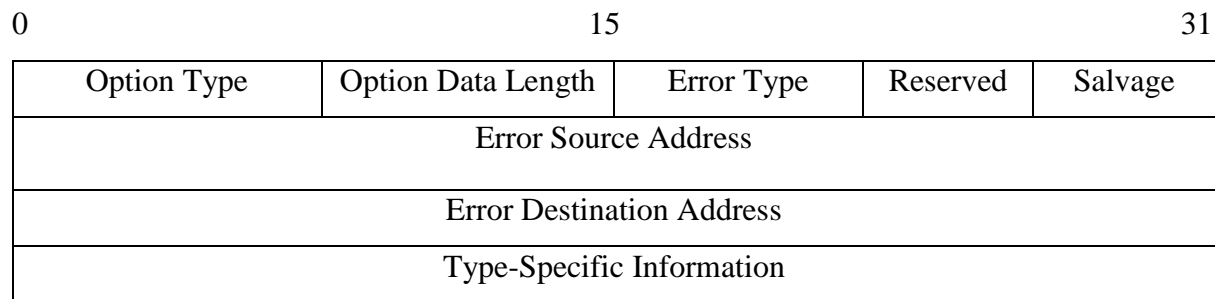


Fig 1.4: DSR Route Error Option Format

In fig 1.4, it shows a visualization of 32-bit DSR route error option format between the source and destination route. [7] Below is the detailed explanation of every field involved in the DSR Route Error option format.

**Option Type:** Nodes not understanding this option will ignore it (8 bits).

**Option Data Length:** Length of the option, in octets, excluding the option type and option data length fields (8 bits).

**Error Type:** The type of error encountered and their values range from 1 to 3 (8 bits).

**Reserved:** Must be sent as 0 and ignored on reception (4 bits).

**Salvage:** Copied from the salvage field in the DSR Source Route option of the packet triggering the Route Error (4 bits unsigned integer).

**Error Source Address:** The address of the node originating the route error (32 bits).

**Error Destination Address:** The address of the node to which the route error must be delivered (32 bits).

**Type Specific Information:** Information specific to the error type of this route error message (32 bits).

## **1.1 Problem Definition**

This project aims at measuring the performance analysis between OLSR and DSR protocols using the Network Simulator. The project mainly focuses on classifying which protocol amongst the both is efficient for providing wireless communication system across a network. In networks utilizing a proactive routing protocol, every node maintains one or more tables representing the entire topology of the network. These tables are updated regularly in order to maintain an up-to-date routing information from each node to every other node.

Unlike proactive routing protocols, reactive routing protocols does not make the nodes initiate a route discovery process until a route to a destination is required. This leads to higher latency than with proactive protocols, but lower overhead. So, for showing the differences between them this project tries to explain them by using proactive OLSR and reactive DSR protocols by analysing through different parameters such as throughput, end to end delay, total simulation time, number of sinks which helps in analysing which protocol is better suitable for ad hoc communication systems.

## **1.2 Existing System**

The wireless communication protocols resemble a larger network topology which helps in easier and faster communication between different nodes or routers. Among these protocols Mobile Ad Hoc Networks (MANET) [1] have a greater provision for enabling communication. These protocols are varied based on their proactive and reactive natures of communication and transmission ranges. They could be basically be distinguished by the routing table technique and on-demand routing protocol establishments.

There have been many systems established using these protocols such as Unmanned Aerial Vehicles (UAV) communications and problems to different attack vectors such as Black Hole Attack Vectors have also been resolved. [1] The MANET protocols used are Optimized Link State Routing (OLSR) and Dynamic Source Routing (DSR) Protocols which were developed using the static wireless communications by enabling performance analysis through Random Waypoint Model, and also investigate how well these selected protocols performs on Wireless Sensor Networks (WSN) in static environments. [3] The performance analysis of these protocols will focus on the impact of the network size and the number of nodes.

### **1.2.1 Disadvantages of Existing System**

The main disadvantage of the existing system is that, the protocols which were developed do not show any simplification of whether the proactive or reactive protocols which have the best performance amongst. This does not give a conclusion of which protocol to use in order to have an effective wireless communication. Also, static wireless networks are only limited to a restricted bandwidth area which cannot be used for large scale applications such as military and other science related areas.

### **1.3 Proposed System**

From the above existing system, to overcome its disadvantages this is a system developed to differentiate and compare the performances of both Proactive and Reactive Routing Protocols. So, in this system this project is going to differentiate [4] and analyse their performances using Optimized Link State Routing (Proactive) and Dynamic Source Routing (Reactive) protocols.

This project first implements these both protocols using the C++ Language and then simulate them using the Network Simulator (NS3). OLSR is a proactive link state routing protocol which uses hello and topology control (TC) messages to discover and then disseminate link state information throughout the mobile ad hoc network.

DSR is a reactive routing protocol which forms a route on-demand when a transmitting node requests one and it uses source routing [7] instead of relying on the routing table at each intermediate device. Once the protocols are being simulated, this project tries to analyse their performances by implementing a combined logic which distinguishes both the protocols with a set of parameters.

For each and every visualization of the protocol this project generates a Comma Separated Value (CSV) file on the background on the execution which stores different values such as Receiver Rate, Packets Received, Throughput. Based on the total simulation time as a final parameter this project can draw an analysis of which routing protocol is best for wireless communication.

### **1.3.1 Advantages of Proposed System**

The main advantage of the proposed system is that, this project could draw a final output of which protocol is best amongst both the proactive and reactive routing protocols for wireless communications. Based on the CSV files generated, this work analyse each and every parameter finally by their total simulation time and the throughput. Also, could result in deciding which amongst the both protocols can be used for large scale industrial communication systems.

## **1.4 Requirements Specification**

### **1.4.1 Software Requirements**

|                      |  |
|----------------------|--|
| Operating System     | : Ubuntu 16.04                                       |
| Programming Language | : C++  |
| Other Software       | : NS3, VMWare 12.0, Trace Metrics, Python Visualizer |

### **1.4.2 Hardware Requirements**

|           |                     |
|-----------|---------------------|
| RAM       | : 4GB               |
| Hard Disk | : 500 MB            |
| Processor | : Dual Core 2.6 GHz |



## **2. LITERATURE SURVEY ON MANET PROTOCOLS**

The research for analysing different wireless communication technologies has brought up the development of different routing protocols which help in the proper establishment of routes and the transmission of message flow system from the desired source to destination. This has reduced the wired setup communication system and also the [1] setup cost. This type of wireless communications is been enabled using the Mobile Ad Hoc Networks (MANET) which work on top of a link layer routing network.

The past developments in this area include the performance analysis of MANET routing protocols for UAV communications (Unnamed Aerial Vehicle) technology which are used in the military base camps for transferring message from one base station to another and also used in other civilian application such as in Amazon for delivering couriers from the desired package centre to the resident location. This type of communication system requires a low setup cost and also can travel at a faster rate across larger bandwidths. [2]

The MANET routing protocols were also analysed by considering a set of parameters such as the Quality of Service (QOS) for analysing which protocol could obtain the best route amongst Ad Hoc On Demand Distance Vector (AODV) Routing Protocol and Dynamic Source Routing (DSR) Protocols. [3] This analysis helped in finding the better node density which enables to obtain the higher performance among the protocols by parallely estimating the factors such as throughput and average end delay.

The Optimized Link State Routing (OLSR) Protocol is used in eliminating long delays in transmitting data packets, helps in establishing communication very frequently for dense set of nodes, enables route requests for new destinations very frequently and also allows higher broadcast ratio. This protocol has guided for large range of communication system and also has an improved capacity for storing the routing table information that is updated on a periodic basis. [4] These different methodologies for analysing the wireless sensor networks has improved the efficiency for wireless communications and increased rates of data transfer between the routers.

**Table 2:** Summarization of different MANET Routing Protocols

| S.No | Year | Author  | Title   | Techniques  | Advantages  | Disadvantages   |
|------|------|---|---|---|---|---|
| 1    | 2018 | Hassen Redwan Hussen, Sung-Chan Choi, Jaeho Kim, Jong-Hong Park | Performance Analysis of MANET Routing Protocols for UAV Communications    | Adhoc On-Demand Distance Vector, Dynamic Source Routing, Geographic Routing Protocols, Optimized Link State Routing, Optimised Link State Routing | Potential benefits for Drone communications, Civilian applications communication  | Does not state the data rates in WAVE and different application rates such as FTP and Video       |
| 2    | 2018 | Prabhat Kumar Sahu, Biswamohan Acharya, Niranjana Panda         | QOS based Performance Analysis of AODV and DSR Routing Protocols in MANET | Adhoc On-Demand Distance Vector, Dynamic Source Routing, Delay Averaging Technique Quality of Service (QOS), Measurement                          | Better node density enables the higher performance of DSR than AODV by analysing parallelly their throughput and average delay. | Could not analyse QOS parameters such as packet delivery ratio, normalize overheads, jitter, etc. |

|   |      |  |  |   |   |  |
|---|------|--|--|---|---|--|
| 3 | 2016 | P.Jacquet,<br>P. Muhlethaler,<br>T.Clausen,<br>A. Laouiti,<br>A. Qayyum,<br>L. Viennot | Optimized<br>Link State<br>Routing<br>Protocol for<br>Ad Hoc<br>Networks   | Neighbour<br>Sensing,<br>Multipoint<br>Relay<br>Selection,<br>Routing<br>Optimality,<br>Routing<br>Table<br>Calculation | Eliminates<br>long delays<br>in<br>transmitting<br>data packets,<br>Communication is very<br>frequent for<br>dense set of<br>nodes, Route<br>requests for<br>new<br>destinations<br>are very<br>frequent,<br>Higher<br>Broadcast<br>Ratio | OLSR cannot be<br>adopted to the<br>network which<br>is sparse as the<br>communication<br>speed becomes<br>slow. |
| 4 | 2015 | Shivani<br>Attri   | Performance<br>Analysis of<br>OLSR and<br>DSR Routing<br>Protocols for<br>Static<br>Wireless<br>Sensor<br>Networks | Measuring<br>performance<br>through<br>End to End<br>Delay,<br>Network<br>Load,<br>Throughput                           | Average<br>throughput<br>of OLSR is<br>much better<br>than DSR,<br>average end<br>to end delay<br>of DSR is<br>much higher<br>than OLSR   | DSR shows less<br>network<br>average load as<br>compared to<br>OLSR routing<br>protocol.                         |

### 3. METHODOLOGY FOR EVALUATING PERFORMANCE BETWEEN OLSR AND DSR ROUTING PROTOCOLS

#### 3.1 Architecture

The system architecture of this project can be visualized as a 3-tier architecture where the initial phase which is the data tier consists of the OLSR and DSR protocols as input which needs to be simulated. [8] The business tier is the application logic where the protocols are simulated and evaluated using the NS3 Simulator and then gets simulated using the Python Visualizer.

Once the protocols are being simulated the CSV files are being generated in the background for both the protocols and are thereby analysed based on the parameters generated. Once all the parameters are being analysed the effective protocol among the both is being concluded as the output for utilizing [4] them in the process of wireless communications in presentation tier.

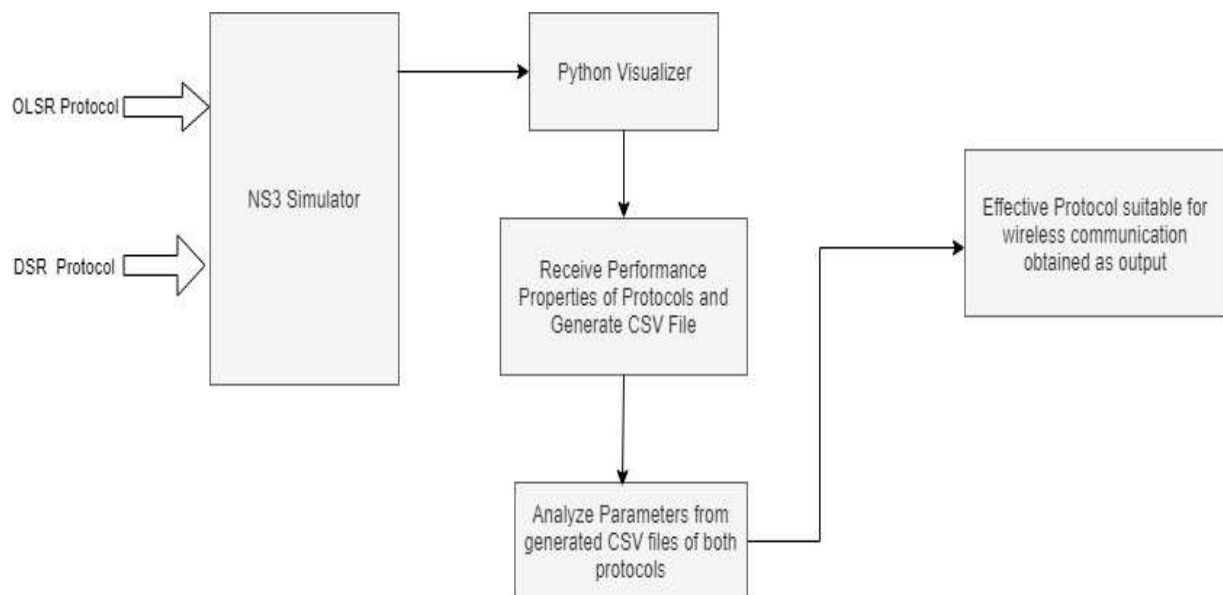


Fig 3.1.1: System Architecture of the application

In fig 3.1.1, the system architecture of this project is being explained using a 3-tier architecture model where the flow explains how the protocols help in analysing the better performance of the entire system by processing through several phases and also about the different stages of simulation and their results.

## 3.2 Modules

### 3.2.1 Development of OLSR Routing Protocol

The network topology information is maintained by periodically exchange link state information. If more reactivity to topological changes is required, the time interval for exchanging of link state information can be reduced. [4] Control messages OLSR uses three kinds of control messages: HELLO, Topology Information Control (TC), and Multiple Interface Declaration (MID). A Hello message is sent periodically to all of nodes neighbours. Hello messages contain information about a node neighbour, the nodes it has chosen as MPRs (i.e., the MPR Selector set), and a list of neighbours with whom bidirectional links have not yet been confirmed.

Every node periodically floods the network with a TC message using the multipoint relaying mechanism. This message contains the nodes MPR Selector set. A MID message is used for announcing that a node is running OLSR on more than one interface. The MID message is flooded throughout the network by the MPRs. [3] Multipoint Relays is defined as; A node N selects an arbitrary subset of its 1-hop symmetric neighbours to forward data traffic. This subset, referred to as an MPR set, covers all the nodes that are two hops away. The MPR set is calculated from information about the nodes symmetric one hop and two hop neighbours. This information is extracted from HELLO messages.

Similar to the MPR set, an MPR Selectors set is maintained at each node. An MPR Selector set is the set of neighbours that have chosen the node as their MPR. Upon receiving a packet, a node checks its MPR Selector set to see if the sender has chosen the n node as MPR. If so, the packet is forwarded, else the packet is [3] processed and discarded. Selection of Multipoint Relay Nodes is done by choosing MPR set so that a minimum of one-hop symmetric neighbours can able to reach all the symmetric two-hop neighbours.

In order to calculate the MPR set, the node must have link state information about all one-hop and two-hop neighbours. Again, this information is gathered from HELLO messages. Only nodes with [7] willingness different than WILL\_NEVER may be considered as MPR. Neighbour discovery is doing as links in an ad-hoc network can be either unidirectional or bidirectional, a protocol for determining the link status is needed. In OLSR, HELLO messages serve this purpose. HELLO messages are broadcast periodically for neighbour sensing. When

a node receives a [7] HELLO message in which its address is found, it registers the link to the source node as symmetric.

### **3.2.2 Development of DSR Routing Protocol**

#### **DSR Route Discovery**

When some source node originates a new packet addressed to some destination node, the source node places in the header of the packet a "source route" giving the sequence of hops that the packet is to follow [3] on its way to the destination. Normally, the sender will obtain a suitable source route by searching its "Route Cache" of routes previously learned; if no route is found in its cache, it will initiate the Route Discovery protocol to dynamically find a new route to this destination node. In this case, we call the source node the "initiator" and the destination node the "target" of the Route Discovery.

When initiating a Route Discovery, the sending node saves a copy of the original packet (that triggered the discovery) in a local buffer called the "Send Buffer". The Send Buffer contains a copy of each packet that cannot be transmitted by this node because it does not yet have a source route to the packet's destination. [4] Each packet in the Send Buffer is logically associated with the time that it was placed into the Send Buffer and is discarded after residing in the Send Buffer for some timeout period `SendBufferTimeout`; if necessary for preventing the Send Buffer from overflowing, a FIFO or other replacement strategy may also be used to evict packets even before they expire.

While a packet remains in the Send Buffer, the node should occasionally initiate a new Route Discovery for the packet's destination address. However, the node must limit the rate at which such new Route Discoveries for the same address are initiated, since it is possible that the destination node is not currently reachable. In particular, due to the limited wireless transmission range and the movement of the nodes in the network, [9] the network may at times become partitioned, meaning that there is currently no sequence of nodes through which a packet could be forwarded to reach the destination. Depending on the movement pattern and the density of nodes in the network, such network partitions may be rare or common.

If a new Route Discovery was initiated for each packet sent by a node in such a partitioned network, a large number of unproductive Route Request packets would be propagated

throughout the subset of the ad hoc network reachable from this node. [9] In order to reduce the overhead from such Route Discoveries, a node should use an exponential back-off algorithm to limit the rate at which it initiates new Route Discoveries for the same target, doubling the timeout between each successive discovery initiated for the same target.

If the node attempts to send additional data packets to this same destination node more frequently than this limit, the subsequent packets should be buffered in the Send Buffer until a Route Reply is received giving a route to this destination, [4] but the node must not initiate a new Route Discovery until the minimum allowable interval between new Route Discoveries for this target has been reached. This limitation on the maximum rate of Route Discoveries for the same target is similar to the mechanism required by Internet nodes [5] to limit the rate at which ARP Requests are sent for any single target IP address

### **DSR Route Maintenance**

When originating or forwarding a packet using a source route, each node transmitting the packet is responsible for confirming that data can flow over the link from that node to the next hop. An acknowledgement can provide confirmation that a link is capable of carrying data, [3] and in wireless networks, acknowledgements are often provided at no cost, either as an existing standard part of the MAC protocol in use, or by a "passive acknowledgement".

If a built-in acknowledgement mechanism is not available, the node transmitting the packet can explicitly request that a DSR-specific software acknowledgement be returned by the next node along the route; this software acknowledgement will [3] normally be transmitted directly to the sending node, but if the link between these two nodes is unidirectional (Section 4.6), this software acknowledgement could travel over a different, multi-hop path.

After an acknowledgement has been received from some neighbour, a node may choose not to require acknowledgements from that neighbour for a brief period of time, unless the network interface connecting a node to that neighbour always [10] receives an acknowledgement in response to unicast traffic.

When a software acknowledgement is used, the acknowledgement request may be retransmitted up to a maximum number of times. A retransmission of the acknowledgement

request can be sent as a separate packet, piggybacked on a retransmission of the original data packet, or piggybacked on any packet with the same next-hop destination that does not also contain a software acknowledgement.[8]

After the acknowledgement request has been retransmitted the maximum number of times, if no acknowledgement has been received, then the sender treats the link to this next-hop destination as currently "broken". [9] It should remove this link from its Route Cache and should return a "Route Error" to each node that has sent a packet routed over that link since an acknowledgement was last received.

### **3.3.3 Performance Evaluation between OLSR and DSR Routing Protocols**

This project helps to obtain a final conclusion of which protocol among the OLSR and DSR routing protocols is the most suitable for wireless communications. [6] Also, by this the project can also give a conclusion which amongst the proactive or reactive routing protocols are suitable in wireless communications and which are capable of being used in large scale applications. [8]

In order to obtain the final estimate, this project uses a set of parameters and understands the resultant information obtained in the background which is in the Comma Separated Value (.csv) format. This gives a detailed explanation of the set of parameters considered for evaluating their performances at each simulation second and bring out the best performance suitable for these wireless communications. The set of parameters used in this project are throughput, packets received, the number of sinks, end to end delay and the total simulation time taken. [5]

Based on the CSV file generated in the background for the route established the project helps in estimating which amongst the both protocols have the better efficiency. [9] Once the efficient protocol is being obtained this project can bring a conclusion whether proactive or reactive routing protocol which is the best suitable for wireless communication system.



### 3.3 UML Diagrams

#### 3.3.1 State Chart Diagram

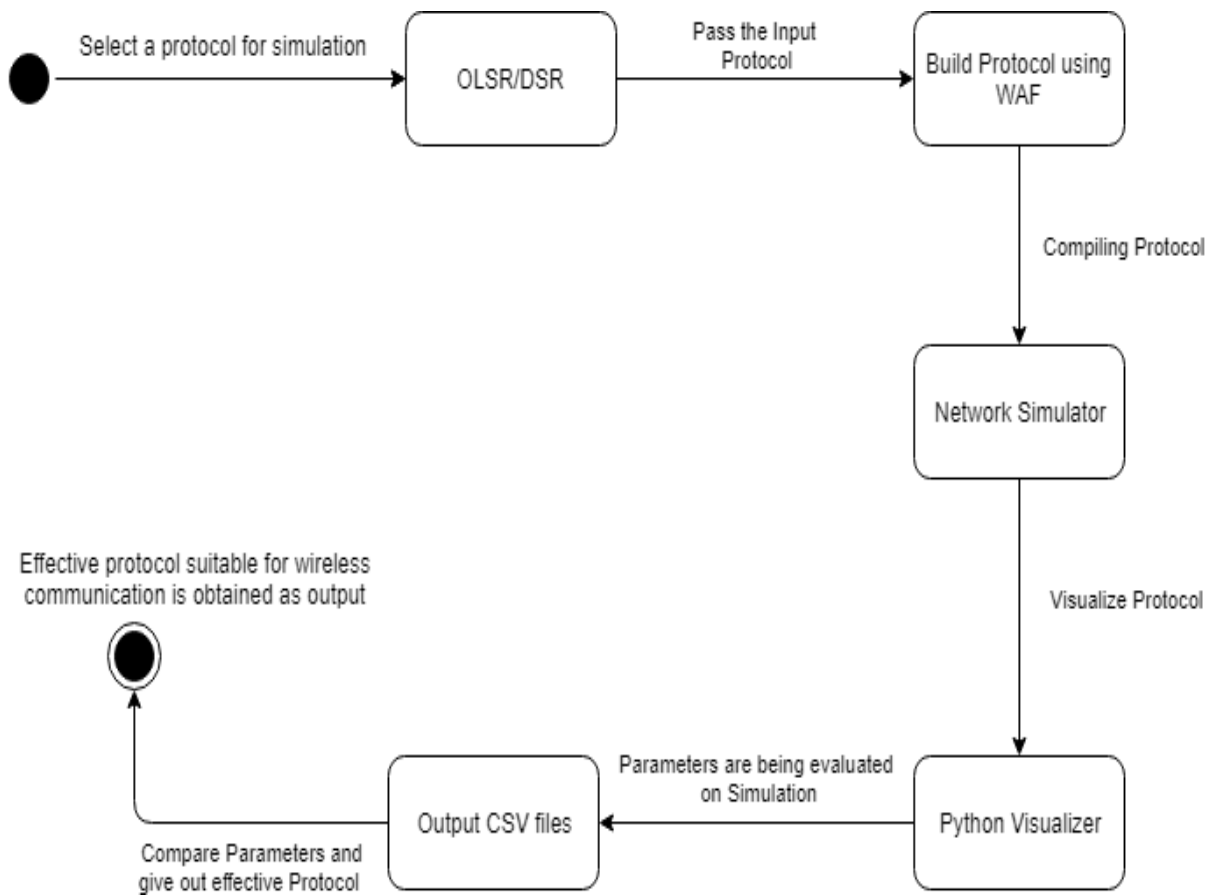


Fig 3.3.1.1: State Chart Diagram

In fig 3.3.1.1, it shows a detailed explanation of state chart diagram where the start point is the selection of the protocol and then passing it to the further states for simulating and analysing their parameters and thereby obtaining the required CSV files [7].

Once the CSV files are obtained the parameters are compared from both the protocols and then the effective protocol is decided as the best protocol suitable for wireless communication is obtained as output at the final state.

### 3.3.2 Sequence Diagram

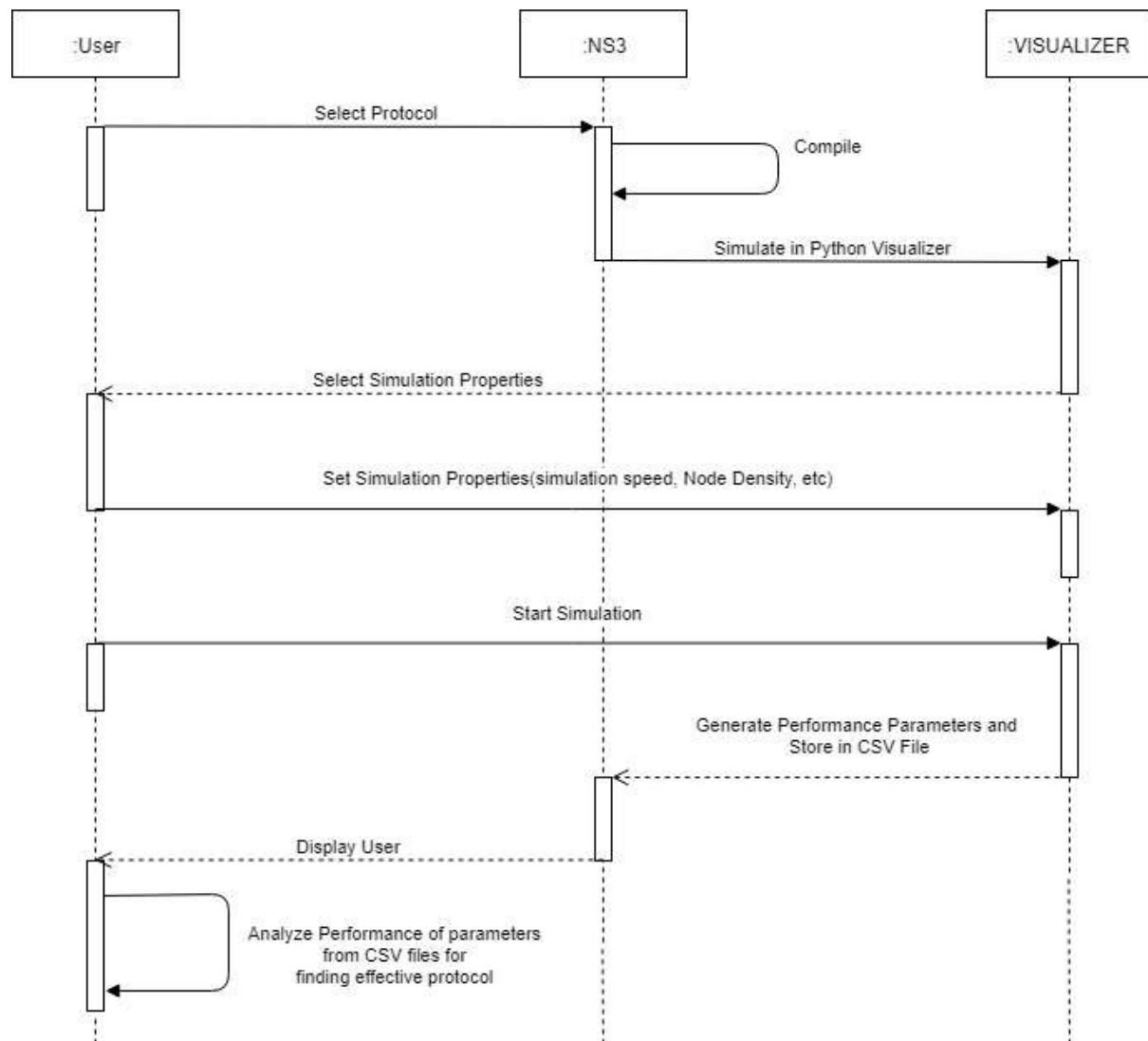


Fig 3.3.2.1: Sequence Diagram

In fig 3.3.2.1, it shows the detailed explanation of sequence diagram where the objects are User, NS3 and the Visualizer which helps in establishing the sequence of flow of messages. The user selects a protocol and compiles it in the NS3 platform and then tries to visualizes it and then analyses the set of parameters from the CSV files [9] obtained and obtains which protocol is the best suitable for wireless communication amongst OLSR and DSR.

### 3.3.3 Use Case Diagram

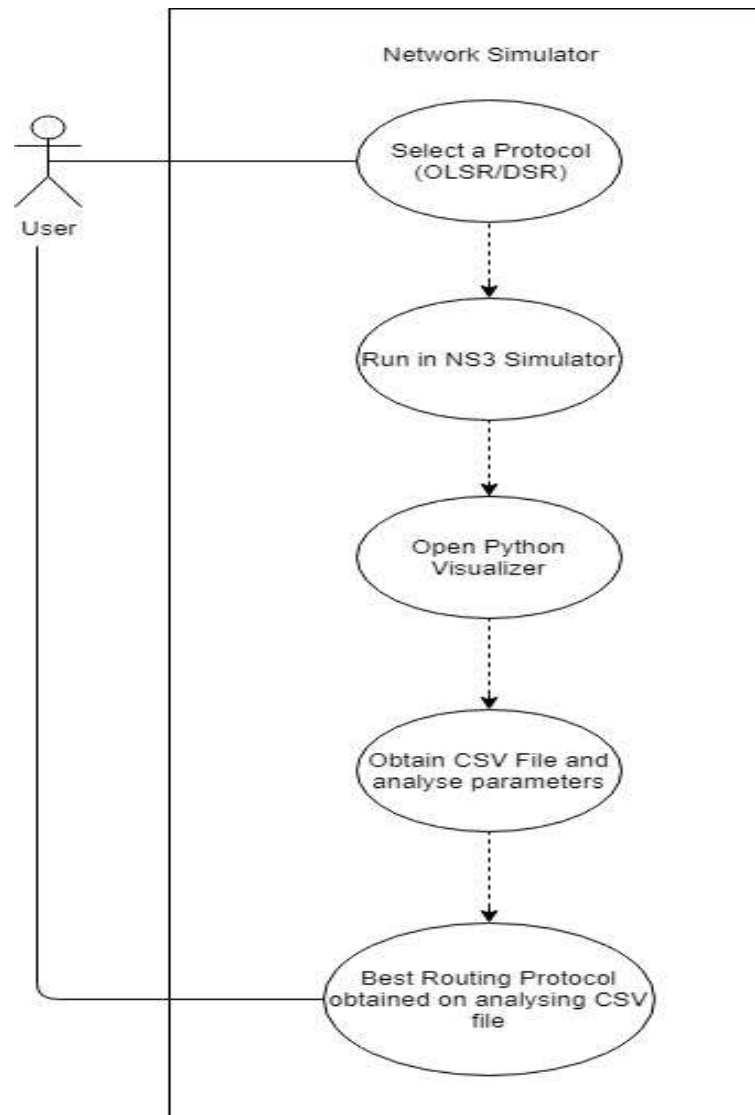


Fig 3.3.3.1: Use Case Diagram

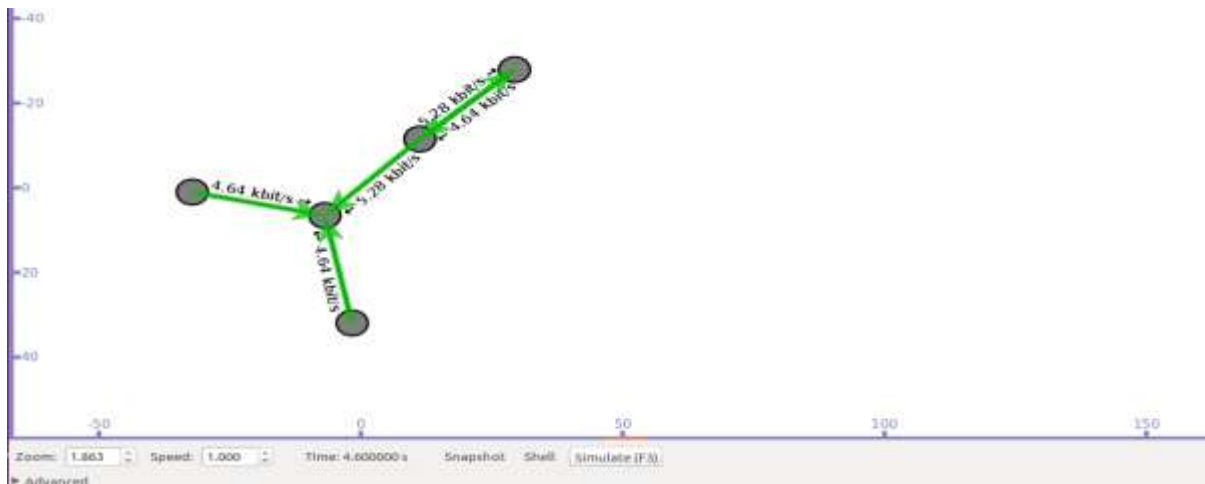
In fig 3.3.3.1, it shows a detailed explanation of use case diagram, where the user acting as an goes through several use cases for analysing the effective protocol that can be used in wireless communication by comparing the obtained CSV files [9] after simulation within the total simulation time taken and then finally this obtained output is being sent to the user.

## 4. TESTING AND RESULTS

### 4.1 Test Cases

**Table 4.1.1:** Simulating OLSR protocol with 5 nodes

| S. No | Test Case Description                                  | Expected Output  | Actual Output        |
|-------|--|--|----------------------|
| 1     | Simulation of OLSR Protocol with number of nodes as 5. | It must simulate the protocol and display the simulation rate in the system. | Working as expected. |



**Fig 4.1.1:** Simulation of OLSR Protocol

In fig 4.1.1, it shows the simulation of Optimized Link State Routing Protocol which consists a total number of 5 nodes with different data rates that are been transmitted between the neighbouring nodes for establishing a communication path.

**Table 4.1.2:** Evaluating OLSR protocol using trace file

| S. No | Test Case Description                                    | Expected Output  | Actual Output        |
|-------|--|--|----------------------|
| 2     | Evaluation of OLSR Protocol by analysing the trace file. | It must display all the parametric evaluation that take place at each node and the entire simulation data. | Working as expected. |

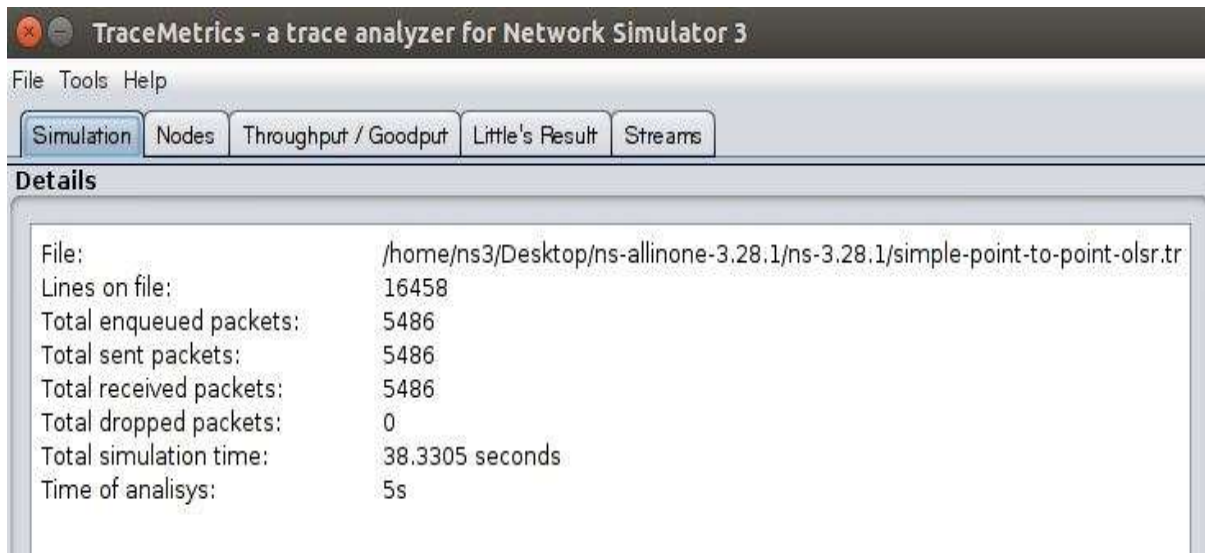


Fig 4.1.2: Simulated data of OLSR Protocol using Trace Metrics

In fig 4.1.2, it explains about the parameters that are evaluated for the above simulated OLSR protocol which help in analysing the performance of the wireless communication that it undergoes.

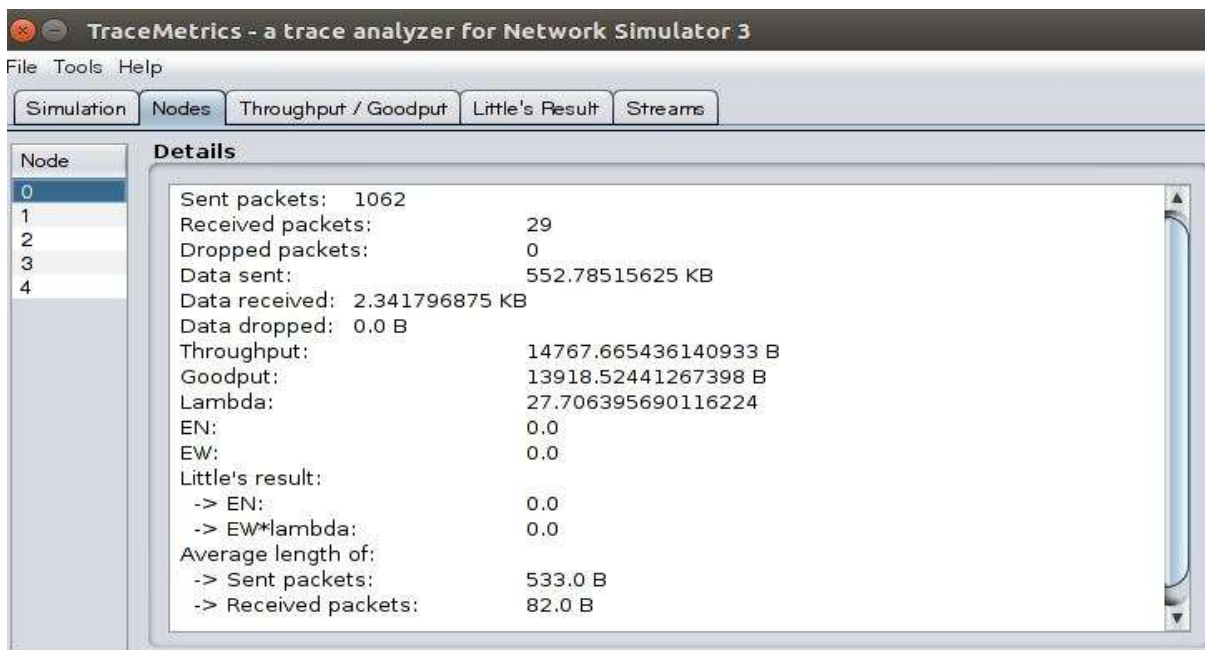



Fig 4.1.3: Individual node performance evaluation for OLSR Protocol

In fig 4.1.3, it explains about the individual node performance in OLSR Protocol when it is in the process of establishing a communication path between the neighbouring nodes and also about the parameters that are involved in the transmission.



The screenshot shows the TraceMetrics application window with the 'Throughput / Goodput' tab selected. It displays a table with three columns: Node, Throughput, and Goodput. The data is as follows:

| Node | Throughput         | Goodput            |
|------|--------------------|--------------------|
| 0    | 14767.665436140933 | 13918.52441267398  |
| 1    | 28.541240004696    | 0.0                |
| 2    | 30221.416365557452 | 28371.349186679014 |
| 3    | 30139.862511576943 | 28371.349186679014 |
| 4    | 30.054395324871837 | 0.0                |

Fig 4.1.4: Throughput evaluation of each node for OLSR Protocol

In fig 4.1.4, it shows the calculated throughput [10] and goodput of the individual node when simulated and also it helps in finding the best path through which the faster communication can be done.

**Table 4.1.3:** Simulation of DSR protocol with 5 nodes

| S. No | Test Case Description                                 | Expected Output  | Actual Output        |
|-------|---|--|----------------------|
| 3     | Simulation of DSR Protocol with number of nodes as 5. | It must simulate the protocol and display the simulation rate in the system. | Working as expected. |

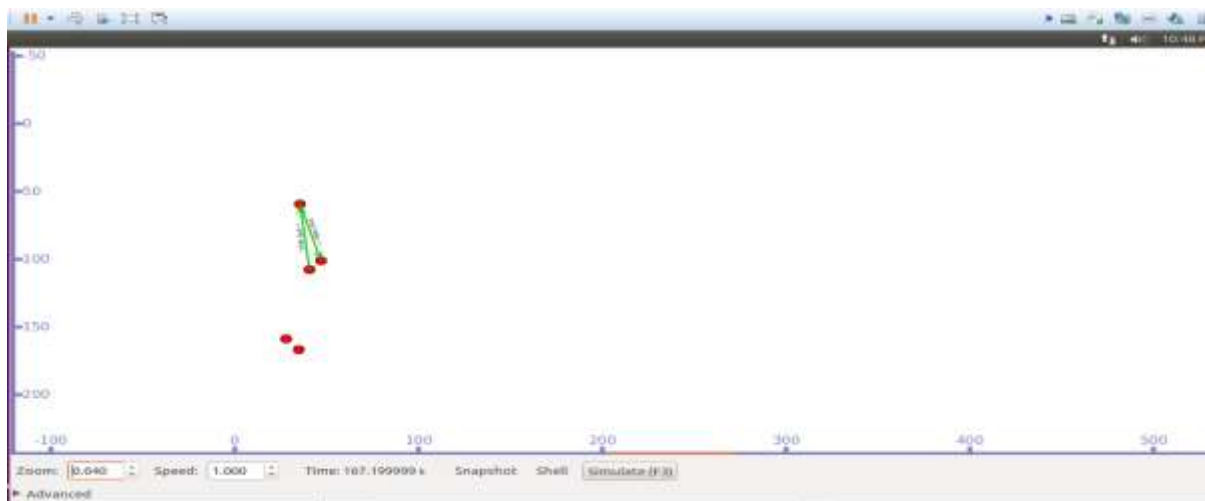
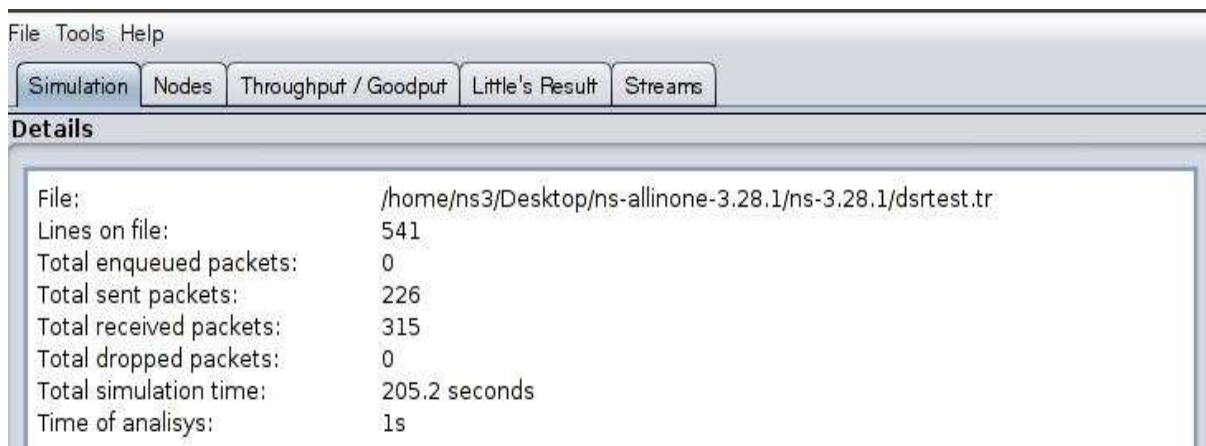


Fig 4.1.5 Simulation of DSR Protocol

In fig 4.1.5, it shows the simulation of Dynamic Source Routing Protocol which consists a total number of 5 nodes with different data rates that are been transmitted between the neighbouring nodes for establishing a communication path.

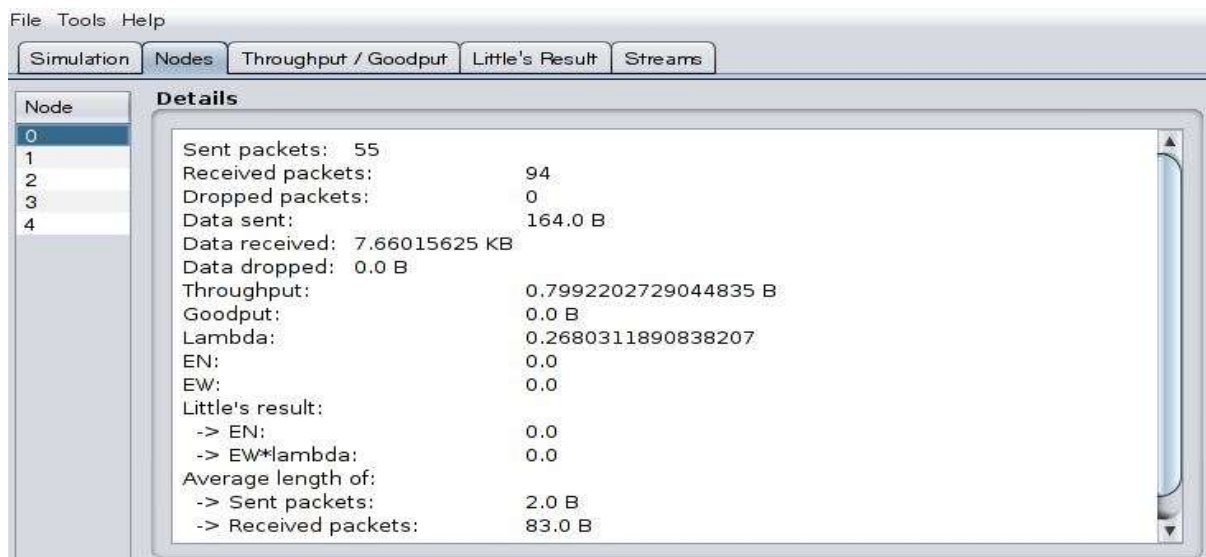
**Table 4.1.4:** Evaluating DSR protocol using trace file

| S. No | Test Case Description                                   | Expected Output  | Actual Output        |
|-------|---|--|----------------------|
| 4     | Evaluation of DSR Protocol by analysing the trace file. | It must display all the parametric evaluation that take place at each node and the entire simulation data. | Working as expected. |



**Fig 4.1.6** Simulated data of DSR Protocol using Trace Metrics

In fig 4.1.6, it explains about the parameters that are [5] evaluated for the above simulated DSR protocol which help in analysing the performance of the wireless communication that it undergoes.



**Fig 4.1.7:** Individual node performance evaluation for DSR Protocol

In fig 4.1.7, it explains about the individual node performance in DSR Protocol when it is in the process of establishing a communication path between the neighbouring nodes and also about the parameters that are involved in the transmission.



| Node | Throughput         | Goodput            |
|------|--------------------|--------------------|
| 0    | 0.7992202729044835 | 0.0                |
| 1    | 0.8576998050682262 | 0.0                |
| 2    | 5.828460038986355  | 2.8070175438598494 |
| 3    | 0.5263157894736843 | 0.0                |
| 4    | 61.812865497076025 | 31.500974658869396 |

Fig 4.1.8: Throughput evaluation of each node for DSR Protocol

In fig 4.1.8, it shows the calculated throughput and [5] goodput of the individual node when simulated and also it helps in finding the best path through which the faster communication can be done.

**Table 4.1.5:** Measuring performance of OLSR protocol using generated CSV file

| S. No | Test Case Description   | Expected Output   | Actual Output        |
|-------|---|---|----------------------|
| 5     | Performance analysis of OLSR Protocol by understanding its simulation and CSV files where number of nodes is 20 and total simulation time is 200. | Simulating and generating the CSV file for OLSR Protocol. | Working as expected. |

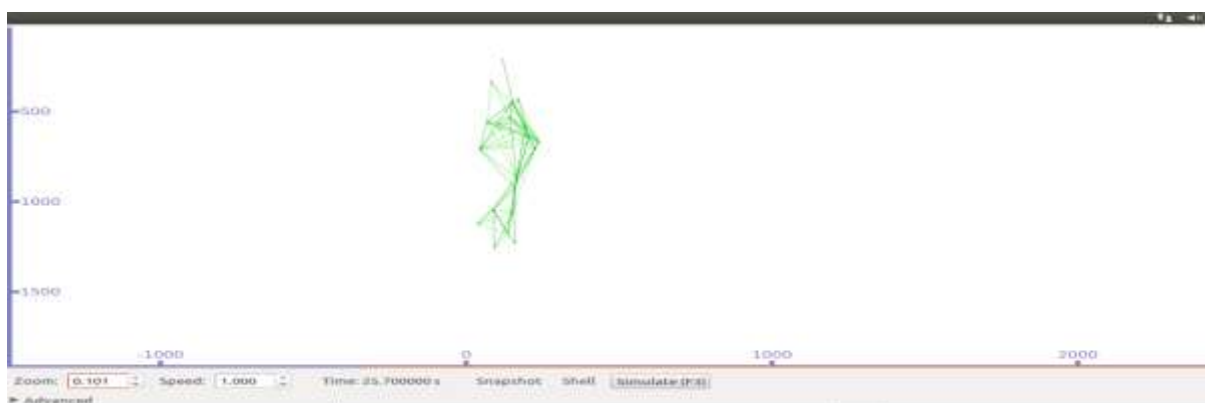


Fig 4.1.9: Simulation of OLSR Protocol with 20 nodes



In fig 4.1.9, it resonates the simulation of OLSR routing protocol with 20 number of nodes with a transmission power of 7.5W and number of sinks as 10, [8] which shows the communication path from the source to destination needed for wireless communication.

|    | A                 | B                | C               | D                | E             | F          |
|----|-------------------|------------------|-----------------|------------------|---------------|------------|
| 1  | Simulation Second | Packets Received | Number Of Sinks | Routing Protocol | End2End Delay | Throughput |
| 2  | 0                 | 0                | 10              | OLSR             | 7.5           | 0          |
| 3  | 1                 | 0                | 10              | OLSR             | 7.5           | 0          |
| 4  | 2                 | 0                | 10              | OLSR             | 7.5           | 0          |
| 5  | 3                 | 0                | 10              | OLSR             | 7.5           | 0          |
| 6  | 4                 | 0                | 10              | OLSR             | 7.5           | 0          |
| 7  | 5                 | 0                | 10              | OLSR             | 7.5           | 0          |
| 8  | 6                 | 0                | 10              | OLSR             | 7.5           | 0          |
| 9  | 7                 | 0                | 10              | OLSR             | 7.5           | 0          |
| 10 | 8                 | 0                | 10              | OLSR             | 7.5           | 0          |
| 11 | 9                 | 0                | 10              | OLSR             | 7.5           | 0          |
| 12 | 10                | 0                | 10              | OLSR             | 7.5           | 0          |
| 13 | 11                | 14               | 10              | OLSR             | 21.5          | 7.168      |
| 14 | 12                | 24               | 10              | OLSR             | 31.5          | 12.288     |
| 15 | 13                | 24               | 10              | OLSR             | 31.5          | 12.288     |
| 16 | 14                | 24               | 10              | OLSR             | 31.5          | 12.288     |
| 17 | 15                | 21               | 10              | OLSR             | 28.5          | 10.752     |
| 18 | 16                | 30               | 10              | OLSR             | 37.5          | 15.36      |
| 19 | 17                | 32               | 10              | OLSR             | 39.5          | 16.384     |
| 20 | 18                | 27               | 10              | OLSR             | 34.5          | 13.824     |
| 21 | 19                | 28               | 10              | OLSR             | 35.5          | 14.336     |
| 22 | 20                | 30               | 10              | OLSR             | 37.5          | 15.36      |
| 23 | 21                | 34               | 10              | OLSR             | 41.5          | 17.408     |
| 24 | 22                | 34               | 10              | OLSR             | 41.5          | 17.408     |
| 25 | 23                | 36               | 10              | OLSR             | 43.5          | 18.432     |
| 26 | 24                | 35               | 10              | OLSR             | 42.5          | 17.92      |
| 27 | 25                | 32               | 10              | OLSR             | 39.5          | 16.384     |

Fig 4.1.10: Generated CSV file for OLSR Protocol

In fig 4.1.10, it shows the output CSV file of OLSR Protocol where the parameters such as Packets Received, Number of Sinks, End to End Delay and Throughput are evaluated for finding the protocols performance and comparing the different parameters for identifying the suitable range of values for enabling wireless communication.

**Table 4.1.6:** Measuring performance of DSR protocol using generated CSV file

| S. No | Test Case Description  | Expected Output  | Actual Output        |
|-------|--|--|----------------------|
| 6     | Performance analysis of DSR Protocol by understanding its simulation and CSV files where number of nodes is 20 and total simulation time is 200. | Simulating and generating the CSV file for DSR Protocol. | Working as expected. |

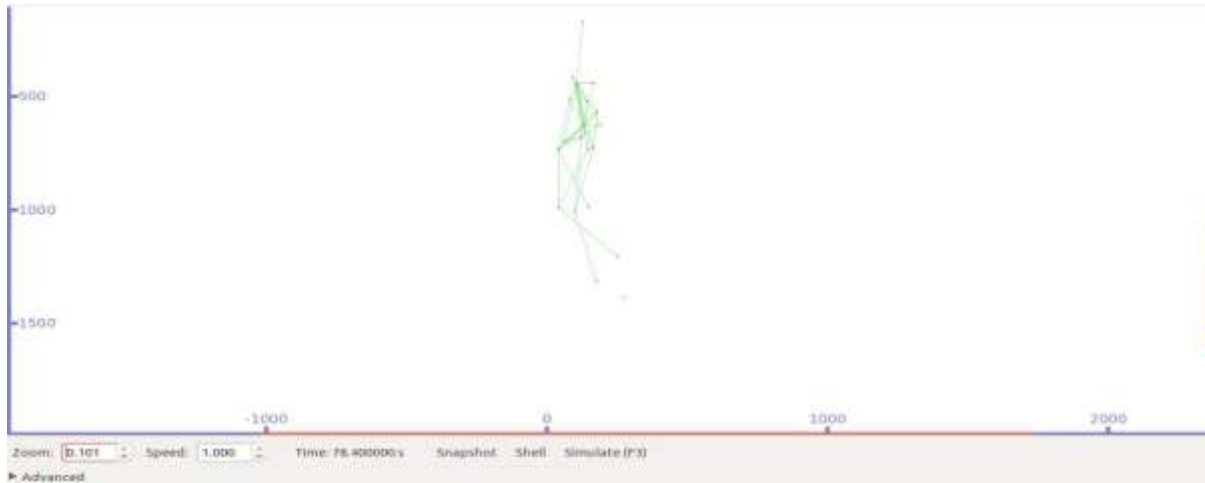


Fig 4.1.11: Simulation of DSR Protocol with 20 nodes

In fig 4.1.11, it resonates the simulation of DSR routing protocol with 20 number of nodes with a transmission power of 7.5W and number of sinks as 10, which shows the communication path from the source to destination needed for wireless communication.

|    | A                 | B                | C               | D                | E             | F          |
|----|-------------------|------------------|-----------------|------------------|---------------|------------|
| 1  | Simulation Second | Packets Received | Number Of Sinks | Routing Protocol | End2End Delay | Throughput |
| 2  | 0                 | 0                | 10              | DSR              | 7.5           | 0          |
| 3  | 1                 | 0                | 10              | DSR              | 7.5           | 0          |
| 4  | 2                 | 0                | 10              | DSR              | 7.5           | 0          |
| 5  | 3                 | 0                | 10              | DSR              | 7.5           | 0          |
| 6  | 4                 | 0                | 10              | DSR              | 7.5           | 0          |
| 7  | 5                 | 0                | 10              | DSR              | 7.5           | 0          |
| 8  | 6                 | 0                | 10              | DSR              | 7.5           | 0          |
| 9  | 7                 | 0                | 10              | DSR              | 7.5           | 0          |
| 10 | 8                 | 0                | 10              | DSR              | 7.5           | 0          |
| 11 | 9                 | 0                | 10              | DSR              | 7.5           | 0          |
| 12 | 10                | 0                | 10              | DSR              | 7.5           | 0          |
| 13 | 11                | 14               | 10              | DSR              | 21.5          | 7.168      |
| 14 | 12                | 36               | 10              | DSR              | 43.5          | 18.432     |
| 15 | 13                | 45               | 10              | DSR              | 52.5          | 23.04      |
| 16 | 14                | 41               | 10              | DSR              | 48.5          | 20.992     |
| 17 | 15                | 40               | 10              | DSR              | 47.5          | 20.48      |
| 18 | 16                | 40               | 10              | DSR              | 47.5          | 20.48      |
| 19 | 17                | 39               | 10              | DSR              | 46.5          | 19.968     |
| 20 | 18                | 39               | 10              | DSR              | 46.5          | 19.968     |
| 21 | 19                | 40               | 10              | DSR              | 47.5          | 20.48      |
| 22 | 20                | 40               | 10              | DSR              | 47.5          | 20.48      |
| 23 | 21                | 39               | 10              | DSR              | 46.5          | 19.968     |
| 24 | 22                | 40               | 10              | DSR              | 47.5          | 20.48      |
| 25 | 23                | 40               | 10              | DSR              | 47.5          | 20.48      |
| 26 | 24                | 41               | 10              | DSR              | 48.5          | 20.992     |
| 27 | 25                | 40               | 10              | DSR              | 47.5          | 20.48      |

Fig 4.1.12: Generated CSV file for DSR Protocol

In fig 4.1.12, it shows the output CSV file of DSR Protocol where the parameters such as Packets Received, Number of Sinks, End to End Delay and Throughput are evaluated for finding the protocols performance and comparing the different parameters for identifying the suitable range of values for enabling wireless communication.

## 4.2 Output Screens

### 4.2.1 Launching the NS3 scratch folder for developing protocols

```
ns3@ubuntu:~$ cd Desktop
ns3@ubuntu:~/Desktop$ cd ns-allinone-3.28.1
ns3@ubuntu:~/Desktop/ns-allinone-3.28.1$ cd ns-3.28.1
ns3@ubuntu:~/Desktop/ns-allinone-3.28.1/ns-3.28.1$ ls
AUTHORS      different.pcap      DLTxPhyStats.txt  odc1.csv          src                ULMacStats.txt     utils            waf.bat
bindings     DLMacStats.txt      doc                odc.tr            test.py            ULPdcpStats.txt    utils.py         waf-tools
build        DLPdcpStats.txt     examples          README            testpy-output      ULRLcStats.txt     utils.pyc        wscript
CHANGES.html DLRlcStats.txt      LICENSE           RELEASE_NOTES     testpy.supply      ULSinrStats.txt    VERSION          wutils.py
contrib      DLRsrpSinrStats.txt Makefile           scratch           ULInterferenceStats.txt ULTxPhyStats.txt  waf              wutils.pyc
ns3@ubuntu:~/Desktop/ns-allinone-3.28.1/ns-3.28.1$ cd scratch
ns3@ubuntu:~/Desktop/ns-allinone-3.28.1/ns-3.28.1/scratch$ ls
dsr.cc  odc.cc  olsr.cc  scratch-simulator.cc  subdir
ns3@ubuntu:~/Desktop/ns-allinone-3.28.1/ns-3.28.1/scratch$
```

Fig 4.2.1.1: NS3 Scratch folder for developing protocols

In fig 4.2.1.1, it explains the steps needed to move to the scratch folder for developing the network protocols. Any network operations needed to be performed are done in the “ns-3.28.1/scratch” workspace where the testing of protocols is being done.

### 4.2.2 Command for building OLSR and DSR protocols

```
ns3@ubuntu:~/Desktop/ns-allinone-3.28.1/ns-3.28.1$ ./waf --run scratch/olsr --vis
Waf: Entering directory '/home/ns3/Desktop/ns-allinone-3.28.1/ns-3.28.1/build'
Waf: Leaving directory '/home/ns3/Desktop/ns-allinone-3.28.1/ns-3.28.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (23.976s)
Could not load icon applets-screenshooter due to missing gnomedesktop Python module
Could not load icon gnome-terminal due to missing gnomedesktop Python module
scanning topology: 5 nodes...
scanning topology: calling graphviz layout
scanning topology: all done.

ns3@ubuntu:~/Desktop/ns-allinone-3.28.1/ns-3.28.1$ ./waf --run scratch/dsr --vis
Waf: Entering directory '/home/ns3/Desktop/ns-allinone-3.28.1/ns-3.28.1/build'
Waf: Leaving directory '/home/ns3/Desktop/ns-allinone-3.28.1/ns-3.28.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (3.302s)
Could not load icon applets-screenshooter due to missing gnomedesktop Python module
Could not load icon gnome-terminal due to missing gnomedesktop Python module
scanning topology: 5 nodes...
scanning topology: calling graphviz layout
scanning topology: all done.
```

Fig 4.2.2.1: Command for building the OLSR and DSR Protocols

In fig 4.2.2.1, it shows the command line compilation and execution steps for building the OLSR and DSR protocols using ‘waf’ which scans the entire topology of the protocol and then shows the simulation of the protocol.

### 4.2.3 Simulation in Python Visualizer

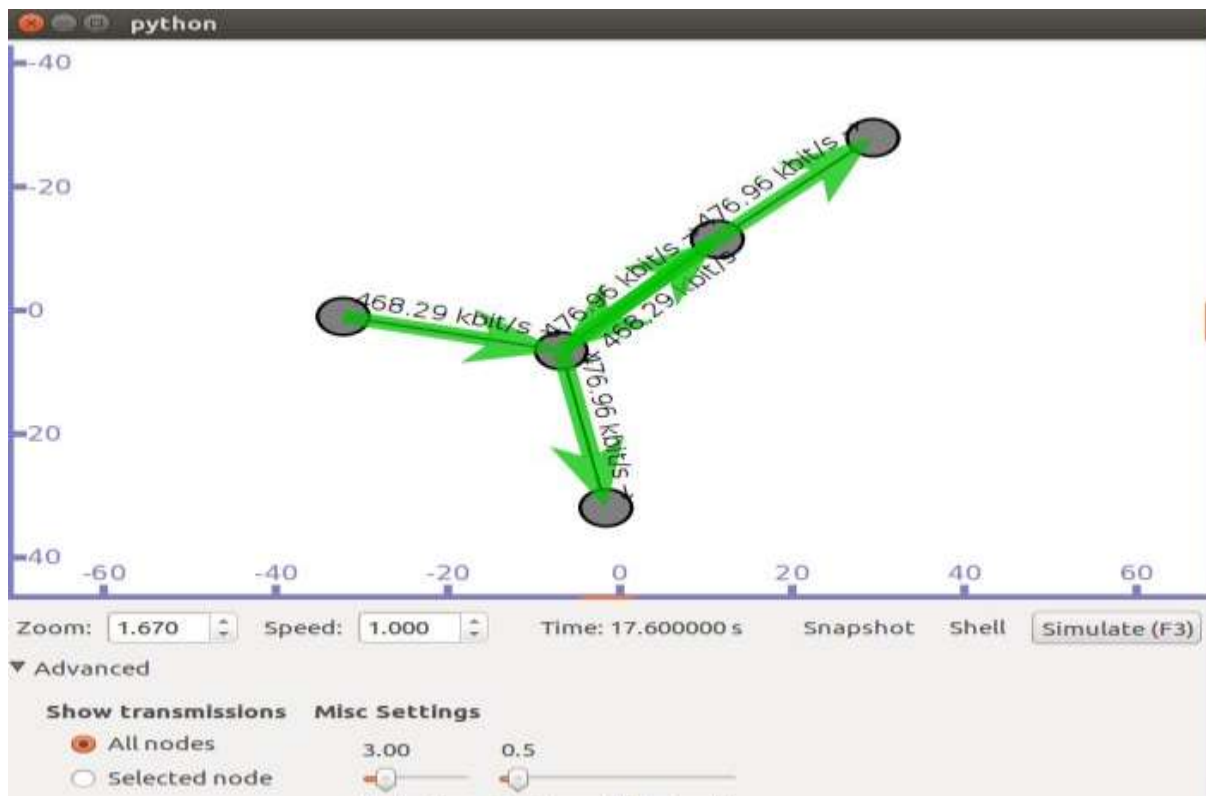


Fig 4.2.3.1: Simulation of routing protocols in python visualizer

In fig 4.2.3.1, it represents the simulation of the protocol once it is being compiled and run. The Python Visualizer is an application for simulating the network communication path between multiple nodes. In this application we can select [10] individual node for knowing its communication path and increase the node speed and also adjust the dimension of the nodes. It also enables a feature of storing the snapshots of the communication lines that take place within the stipulated total simulation time and then understanding the trace file.

### 4.2.4 Launching of Trace Metrics for evaluation of trace file

```
ns3@ubuntu: ~/TraceMetrics/tracemetrics-1.4.0
ns3@ubuntu:~$ cd TraceMetrics
ns3@ubuntu:~/TraceMetrics$ ls
tracemetrics-1.4.0  tracemetrics-1.4.0.zip
ns3@ubuntu:~/TraceMetrics$ cd tracemetrics-1.4.0
ns3@ubuntu:~/TraceMetrics/tracemetrics-1.4.0$ ls
lib  README.TXT  tracemetrics.jar
ns3@ubuntu:~/TraceMetrics/tracemetrics-1.4.0$ java -jar "tracemetrics.jar"
```

Fig 4.2.4.1: Launching the Trace Metrics Application



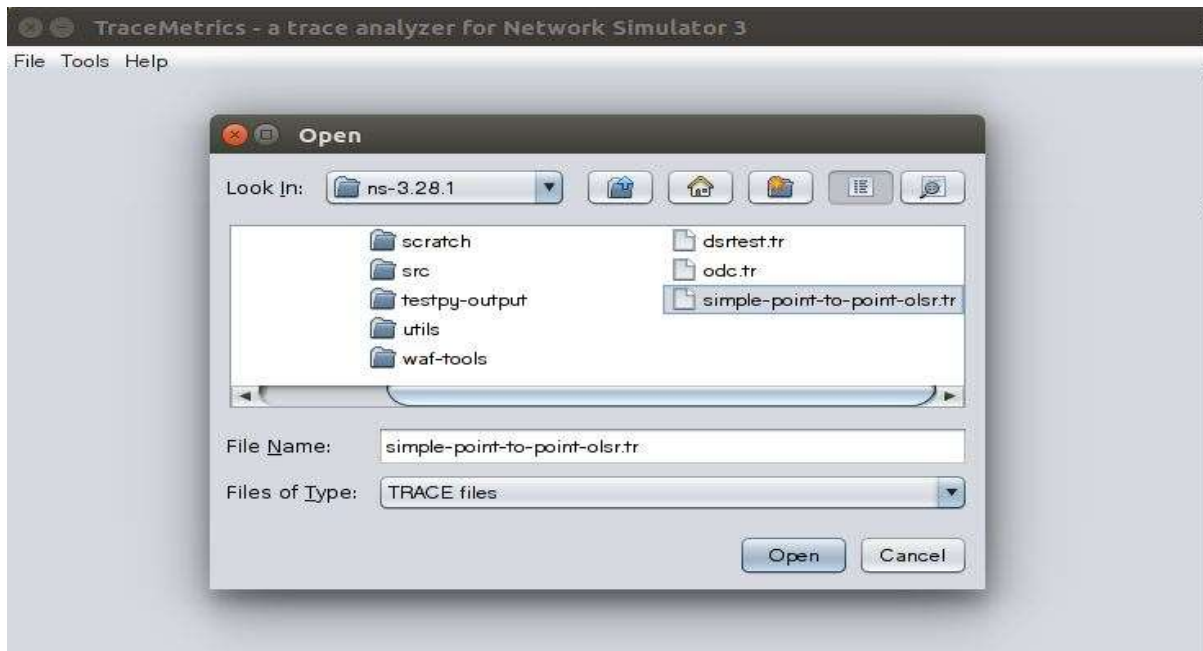


Fig 4.2.4.2: Locating the trace file in Trace Metrics

In fig 4.2.4.1 and 4.2.4.2, it explains how to launch and locate the trace metrics application. The trace metrics is a java application that is used for evaluating the trace files in network simulator. It is of type “jar” file which is used along with the “java” run command for launching the interface of Trace Metrics [8] and then we can locate the file in the “Desktop/ns-allinone-3.28.1/ns-3.28.1” folder where the trace file can be found out.

#### 4.2.5 Executing the analysis of selected trace file

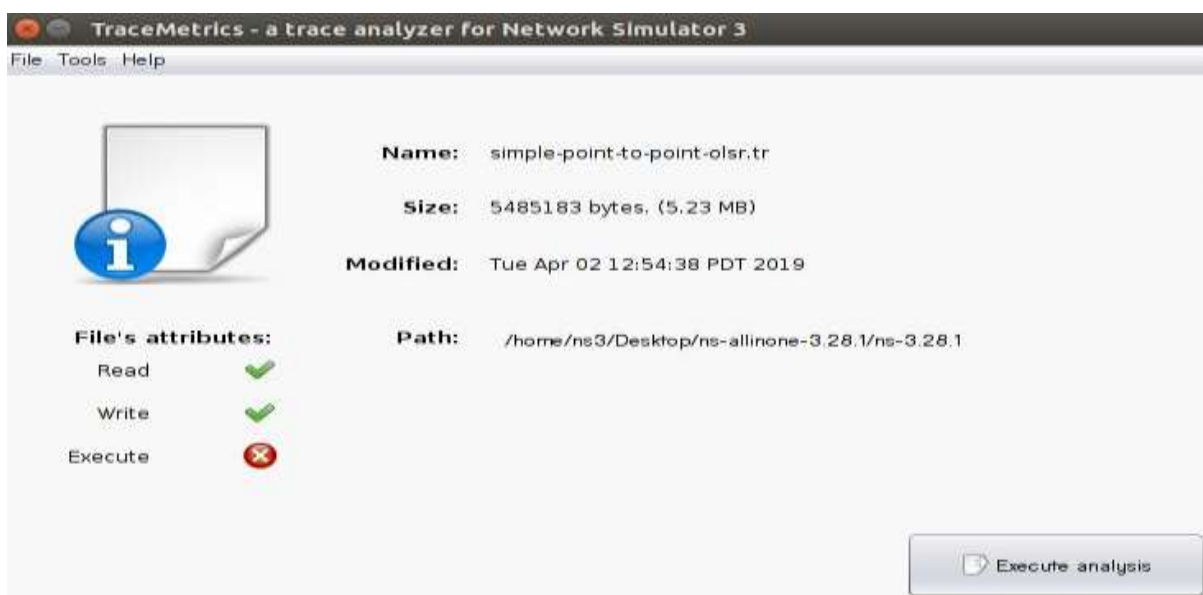


Fig 4.2.5.1: Chosen trace file details for execution



Fig 4.2.5.2: Executing the analysis of selected trace file

In fig 4.2.5.1 and 4.2.5.2, the execution of selected trace file is being done by clicking on the Execute Analysis button which loads the file for analysis within given range of time.

#### 4.2.6 Output of the trace file generated after execution

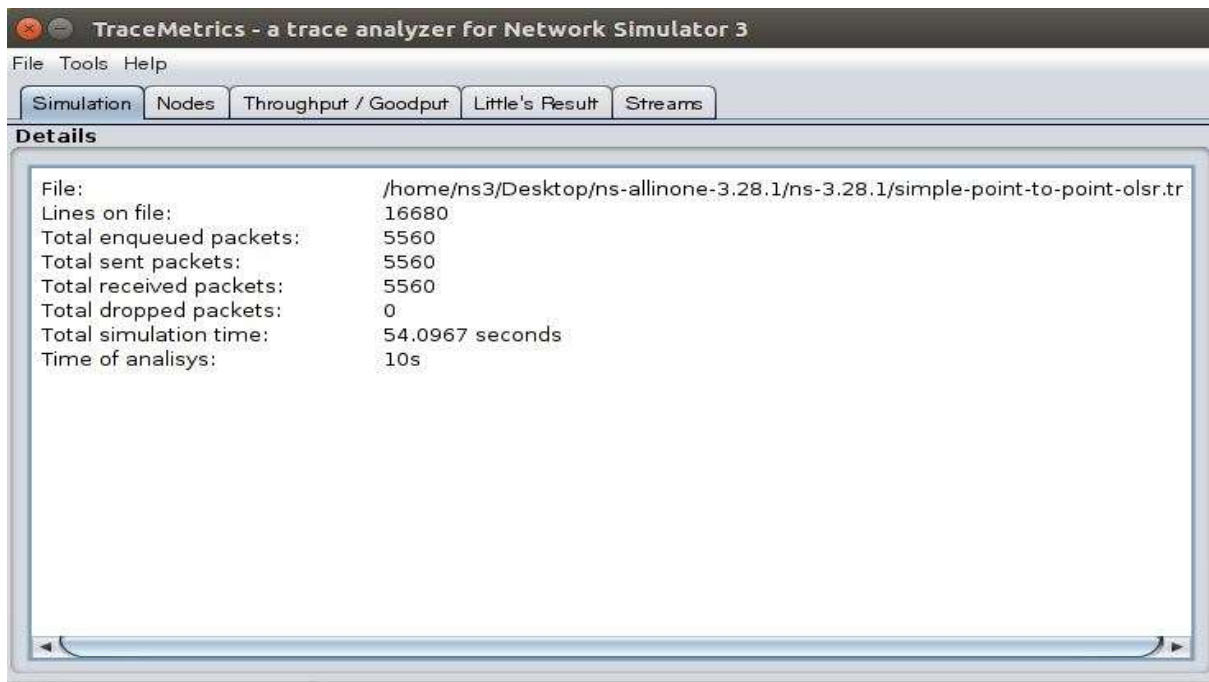


Fig 4.2.6.1: Output of generated trace file

In fig 4.2.6.1, it shows the output of generated trace file where it shows the calculated values of total enqueued packets during simulation, total sent packets, total received packets and the dropped packets within the total simulation time.

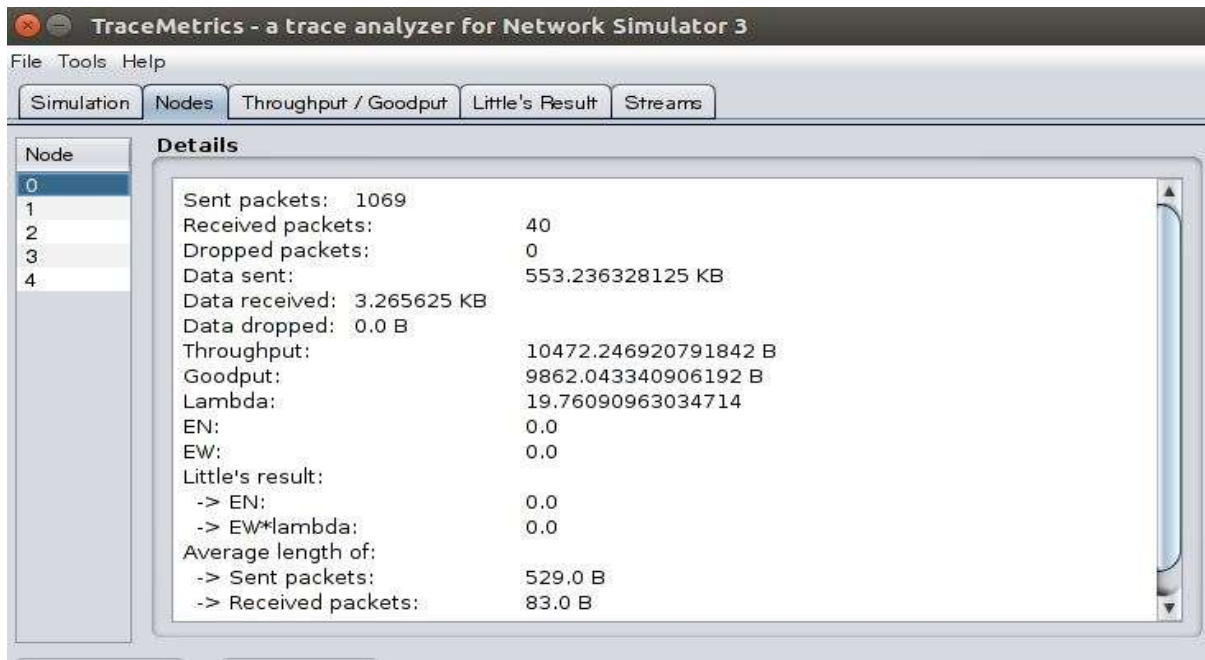


Fig 4.2.6.2: Individual node details generated from trace file

In fig 4.2.6.2, it shows the detailed calculated values for individual node when it communicates with the neighbouring nodes and displays the number of sent packets, received packets, dropped packets, data sent and received, data dropped and throughput, [7] etc for giving an understanding of how faster and efficient the communication takes place.

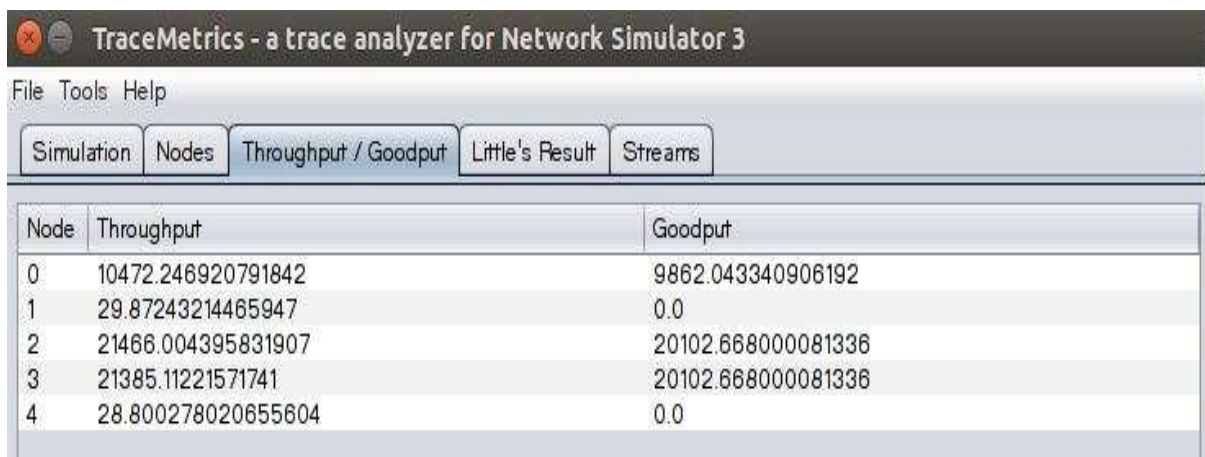


Fig 4.2.6.3: Calculated throughput values of individual node

In fig 4.2.6.3, it shows the throughput values of individual nodes when the total simulation time is being completed and helps in this project for generalizing of which protocol works better under the produced throughput rate.

#### 4.2.7 Graphical representation of the obtained results

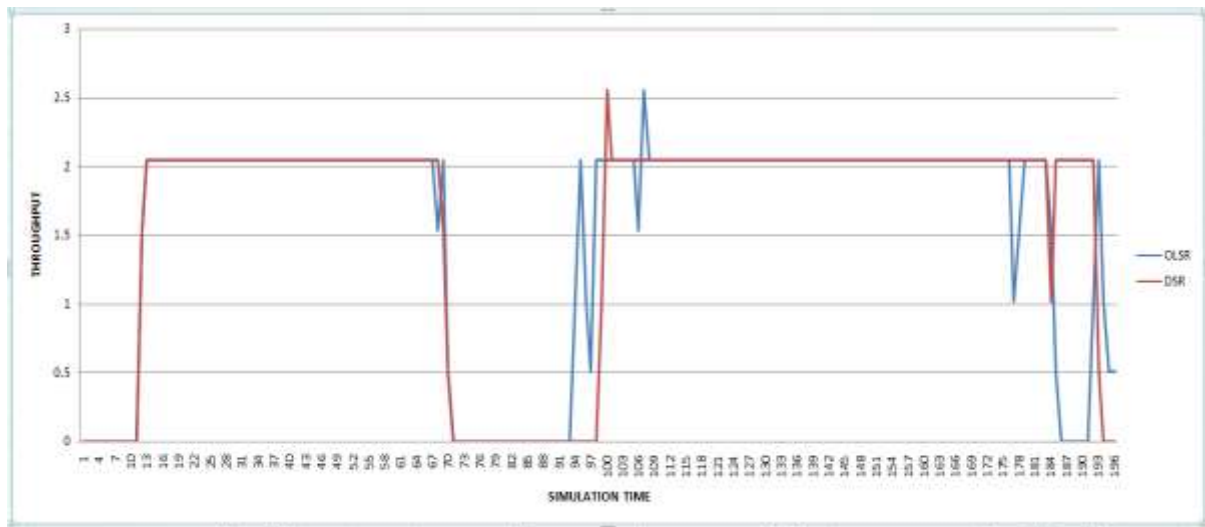


Fig 4.2.7.1: Simulation Time v/s Throughput with number of sinks as 1

In fig 4.2.7.1, it shows the graph plotted between the simulation time and throughput with number of sinks as 1. In this case OLSR performs better with a good overall throughput and produces the best result case.

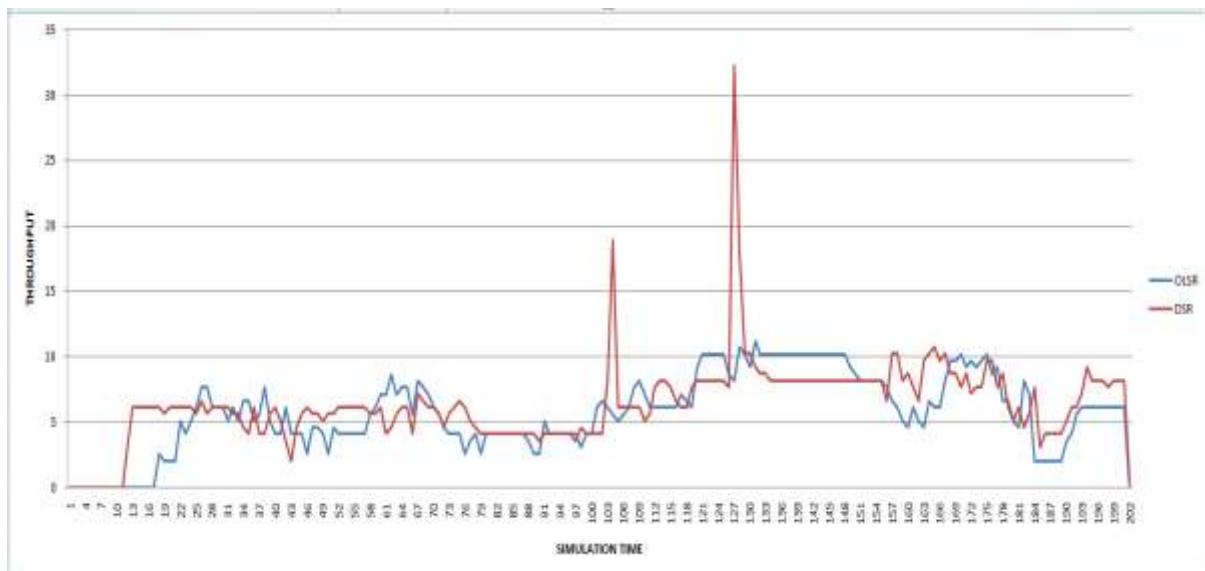


Fig 4.2.7.2: Simulation Time v/s Throughput with number of sinks as 5

In fig 4.2.7.2, it shows the graph plotted between the simulation time and throughput with number of sinks is 5. In this case both the protocols function with the same and also show the same throughput rate by the end of total simulation time.





Fig 4.2.7.3: Simulation Time v/s Throughput with number of sinks as 10

In fig 4.2.7.3, it shows the graph plotted between the simulation time and throughput with number of sinks as 10. In this case both the protocols function with the same and also show the same throughput rate by the end of total simulation time.

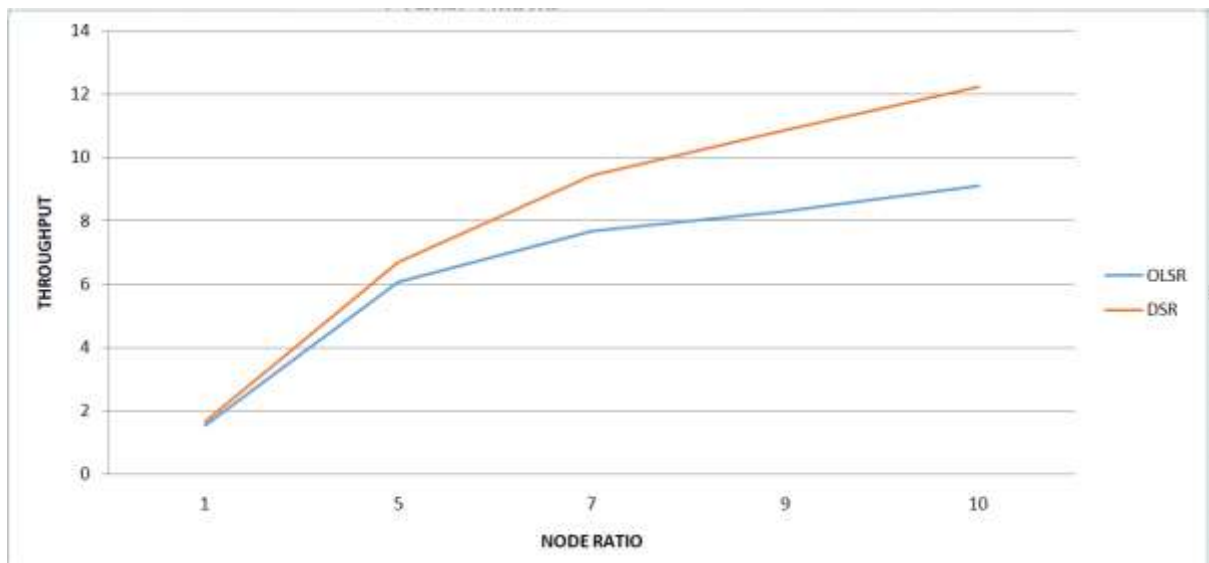


Fig 4.2.7.4: Node Ratio v/s Throughput

In fig 4.2.7.4, it shows the graph plotted between node ratio and throughput which explains the scenario where when the ratio of number of nodes and nodes of sinks increases the throughput of the DSR protocol is high and the number of packets received in that particular unit time is higher.

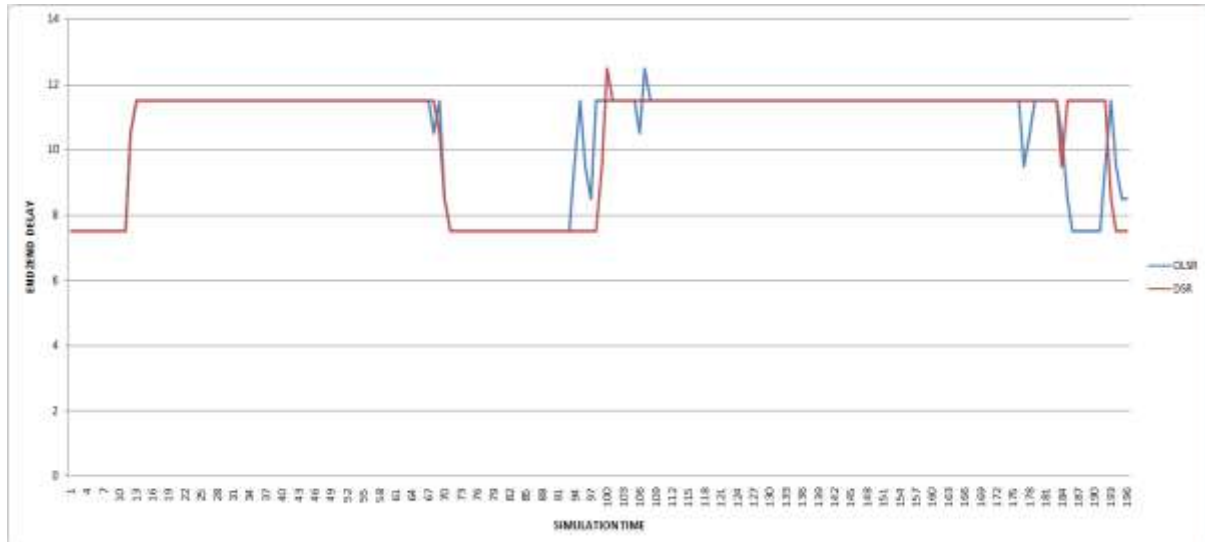


Fig 4.2.7.5: Simulation Time v/s End to End Delay with number of sinks as 5

In fig 4.2.7.5, it shows the graph between simulation time and end to end delay with number of sinks as 5. This resultant output explains that the end to end delay is reduced between source and destination using OLSR protocol.

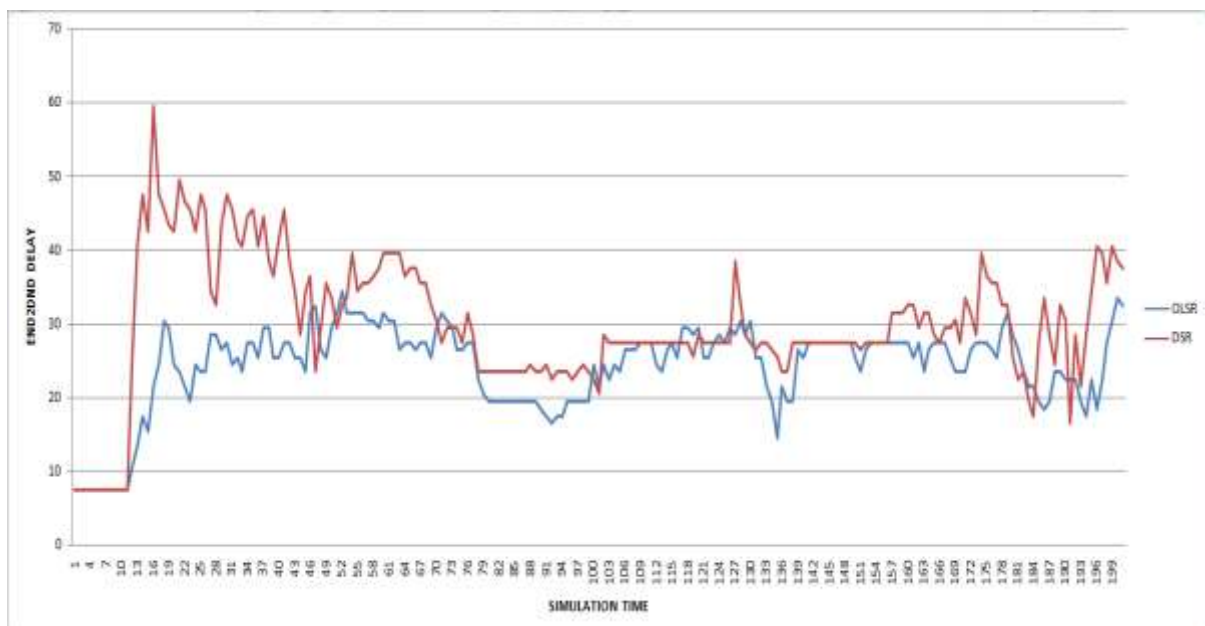


Fig 4.2.7.6: Simulation Time v/s End to End Delay with number of sinks as 10

In fig 4.2.7.6, it shows the graph between simulation time and end to end delay with number of sinks as 10. This resultant output explains that the end to end delay is reduced between source and destination using OLSR protocol.

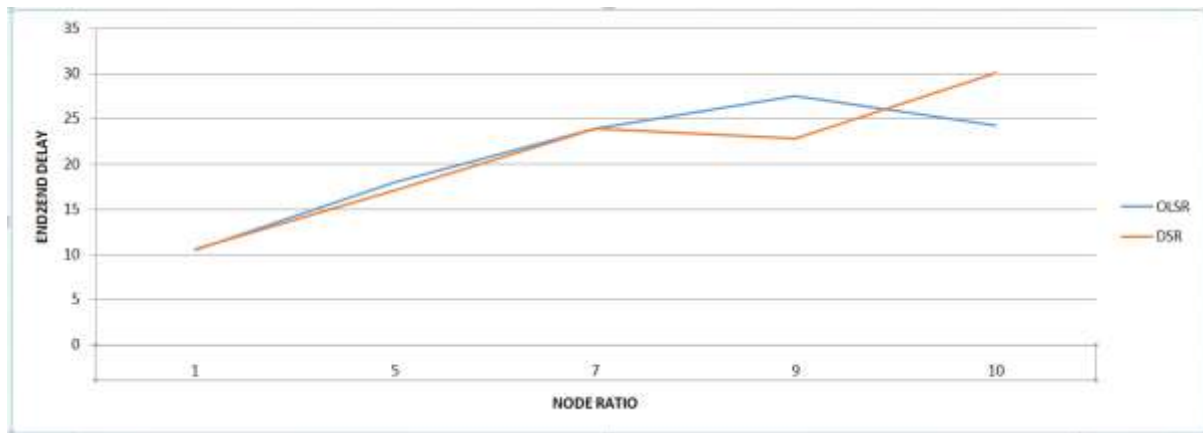


Fig 4.2.7.7 Node Ratio v/s End to End Delay

In fig 4.2.7.7, it shows the graph plotted between node ratio and end to end delay which explains the scenario where when the ratio of number of nodes and nodes of sinks increases the end to end delay for OLSR protocol decreases and due to this the protocol exhibits only a minute delay rate with the other neighbouring nodes.

## 5. CONCLUSION

This proposed project intends to depict the results obtained in measuring the performance between the OLSR and DSR routing protocols, which helps in analysing whether proactive or reactive protocols which is the most suitable depending on the different parameters considered. On considering different set of parameters generated using the CSV file, this project visualizes a set of resultant graphs. Based on the graphs obtained the concluded output is that, the OLSR (proactive) protocol works with a better efficiency for enabling wireless communication when the number of sinks increases based on the simulation time, the end to end delay between the system decreases and also faster transmission can take place.

In the other case using DSR (reactive) protocol as the node ratio increases the throughput generated within the unit of time also increases. On considering different scenarios in this project an analysis is obtained for opting which protocol on what parameter basis for obtaining effective wireless communication paths. But there are other such parameters and protocols which can help in providing even more efficient communication paths.

### **Future Scope**

From above, to add still more functionalities and parameters to the system there are added on proactive and reactive routing protocols which helps in enhancing the performance of wireless communications. Some of these protocols which can be analysed are Ad Hoc On Demand Distance Vector, Temporally Ordered Routing Algorithm, Destination Sequence Distance Vector, Wireless Routing Protocol etc. These set of routing protocols can also be utilized for performing parameter analysis and also can bring an ability to understand its real time applications in enhancing wireless communications.

Some essential parameters that can help in improving the communication flow are area coverage, transmission, maximum and minimum node powers, packet size, delay, jitter. So, from all these combined this project can conclude that there is a great scope for even more future enhancements in the wireless communication by using different MANET protocols.

## BIBLIOGRAPHY

- [1] Hassen Redwan Hussen, Sung-Chan Choi, Jaeho Kim, Jong-Hong Park, Performance Analysis of MANET Routing Protocols for UAV Communications, IOT Platform Research Center, Korea Electronics Technology Institute (KETI), ICUFN, IEEE 2018.
- [2] Prabhat Kumar Sahu, Biswamohan Acharya, Niranjana Panda, QOS based Performance Analysis of AODV and DSR Routing Protocols in MANET, 2018 2<sup>nd</sup> International Conference on Data Science and Business Analytics, IEEE 2018, Siksha 'O' Anusandhan Deemed to be University, Bhubaneswar, India.
- [3] P.Jacquet, P. Muhlethaler, T.Clausen, A. Laouiti, A. Qayyum, L. Viennot, Optimized Link State Routing Protocol for Ad Hoc Networks, Hipercom Project, IEEE 2016, INRIA Rocquencourt, France.
- [4] Shivani Attri, Performance Analysis of OLSR and DSR Routing Protocols for Static Wireless Sensor Networks (WSN), IJARCET Volume 4, Issue 4, April 2015, Maharshi Dayanand Univeristy, Delhi, India.
- [5] Sudha Singh, S C Dutta, Dharmendra K Singh, A study on Recent Research Trends in MANET, IJRRCS Volume 3, No. 3, June 2013, National Institute of Technology, Patna, India.
- [6] Alex Hinds, Michael Ngulube, Shaoying Zhu, A Review of Routing Protocols for Mobile Ad-Hoc Networks (MANET), IJNET, Volume 3, No. 1, February 2013.
- [7] Routing issues in MANET's – Prasad B.V.V.S
- [8] Ad Hoc Wireless Networks: Architectures and Protocols – C Siva Ram Murthy
- [9] Optimization of Routing Protocols in MANET using GA – Singh Sankalp Bahadur.
- [10] A Discrete-Event Network Simulator – NSNAM Directory Reference

## APPENDIX

```
/* Development of Optimized Link State Routing Protocol (OLSR) */

#include <iostream>

#include <fstream>

#include <string>

#include <cassert>

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/applications-module.h"

#include "ns3/olsr-helper.h"

#include "ns3/ipv4-static-routing-helper.h"

#include "ns3/ipv4-list-routing-helper.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Optimized Link State Routing");

int main (int argc, char *argv[])

{

    #if 0

        LogComponentEnable ("SimpleGlobalRoutingExample", LOG_LEVEL_INFO);

    #endif

    // Set up some default values for the simulation. Use the

    Config::SetDefault ("ns3::OnOffApplication::PacketSize", UIntegerValue (210));

    Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue ("448kb/s"));

    CommandLine cmd;
```

```

cmd.Parse (argc, argv);

NS_LOG_INFO ("Create nodes.");

NodeContainer c;

c.Create (5);

NodeContainer n02 = NodeContainer (c.Get (0), c.Get (2));

NodeContainer n12 = NodeContainer (c.Get (1), c.Get (2));

NodeContainer n32 = NodeContainer (c.Get (3), c.Get (2));

NodeContainer n34 = NodeContainer (c.Get (3), c.Get (4));

// Enable OLSR

NS_LOG_INFO ("Enabling OLSR Routing.");

OlsrHelper olsr;

Ipv4StaticRoutingHelper staticRouting;

Ipv4ListRoutingHelper list;

list.Add (staticRouting, 0);

list.Add (olsr, 10);

InternetStackHelper internet;

internet.SetRoutingHelper (list); // has effect on the next Install ()

internet.Install (c);

// We create the channels first without any IP addressing information

NS_LOG_INFO ("Create channels.");

PointToPointHelper p2p;

p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer nd02 = p2p.Install (n02);

NetDeviceContainer nd12 = p2p.Install (n12);

p2p.SetDeviceAttribute ("DataRate", StringValue ("1500kbps"));

```

```

p2p.SetChannelAttribute ("Delay", StringValue ("10ms"));

NetDeviceContainer nd32 = p2p.Install (n32);

NetDeviceContainer nd34 = p2p.Install (n34);

// Later, we add IP addresses.

NS_LOG_INFO ("Assign IP Addresses.");

Ipv4AddressHelper ipv4;

ipv4.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer i02 = ipv4.Assign (nd02);

ipv4.SetBase ("10.1.2.0", "255.255.255.0");

Ipv4InterfaceContainer i12 = ipv4.Assign (nd12);

ipv4.SetBase ("10.1.3.0", "255.255.255.0");

Ipv4InterfaceContainer i32 = ipv4.Assign (nd32);

ipv4.SetBase ("10.1.4.0", "255.255.255.0");

Ipv4InterfaceContainer i34 = ipv4.Assign (nd34);

// Create the OnOff application to send UDP datagrams of size
// 210 bytes at a rate of 448 Kb/s from n0 to n4

NS_LOG_INFO ("Create Applications.");

uint16_t port = 9; // Discard port (RFC 863)

OnOffHelper onoff1 ("ns3::UdpSocketFactory",

                   InetSocketAddress (i34.GetAddress (1), port));

onoff1.SetConstantRate (DataRate ("448kb/s"));

ApplicationContainer onOffApp1 = onoff1.Install (c.Get (0));

onOffApp1.Start (Seconds (10.0));

onOffApp1.Stop (Seconds (20.0));

// Create a similar flow from n3 to n1, starting at time 1.1 seconds

OnOffHelper onoff2 ("ns3::UdpSocketFactory",

```



```

        InetSocketAddress (i12.GetAddress (0), port));

onoff2.SetConstantRate (DataRate ("448kb/s"));

ApplicationContainer onOffApp2 = onoff2.Install (c.Get (3));

onOffApp2.Start (Seconds (10.1));

onOffApp2.Stop (Seconds (20.0));

// Create packet sinks to receive these packets

PacketSinkHelper sink ("ns3::UdpSocketFactory",

        InetSocketAddress (Ipv4Address::GetAny (), port));

NodeContainer sinks = NodeContainer (c.Get (4), c.Get (1));

ApplicationContainer sinkApps = sink.Install (sinks);

sinkApps.Start (Seconds (0.0));

sinkApps.Stop (Seconds (21.0));

AsciiTraceHelper ascii;

p2p.EnableAsciiAll (ascii.CreateFileStream ("simple-point-to-point-olsr.tr"));

p2p.EnablePcapAll ("simple-point-to-point-olsr");

Simulator::Stop (Seconds (30));

NS_LOG_INFO ("Run Simulation.");

Simulator::Run ();

Simulator::Destroy ();

NS_LOG_INFO ("Done.");

return 0;

}

```

**/\* Development of Dynamic Source Routing Protocol (DSR) \*/**

```
#include "ns3/core-module.h"
```

```
#include "ns3/network-module.h"
```

```

#include "ns3/applications-module.h"

#include "ns3/mobility-module.h"

#include "ns3/config-store-module.h"

#include "ns3/wifi-module.h"

#include "ns3/internet-module.h"

#include "ns3/dsr-module.h"

#include <sstream>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("DsrTest");

int main (int argc, char *argv[])

{

    #if 0

    LogComponentEnable ("Ipv4L3Protocol", LOG_LEVEL_ALL);

    LogComponentEnable ("UdpL4Protocol", LOG_LEVEL_ALL);

    LogComponentEnable ("UdpSocketImpl", LOG_LEVEL_ALL);

    LogComponentEnable ("NetDevice", LOG_LEVEL_ALL);

    LogComponentEnable ("Ipv4EndPointDemux", LOG_LEVEL_ALL);

    #endif

    #if 0

    LogComponentEnable ("DsrOptions", LOG_LEVEL_ALL);

    LogComponentEnable ("DsrHelper", LOG_LEVEL_ALL);

    LogComponentEnable ("DsrRouting", LOG_LEVEL_ALL);

    LogComponentEnable ("DsrOptionHeader", LOG_LEVEL_ALL);

    LogComponentEnable ("DsrFsHeader", LOG_LEVEL_ALL);

    LogComponentEnable ("DsrGraReplyTable", LOG_LEVEL_ALL);

    LogComponentEnable ("DsrSendBuffer", LOG_LEVEL_ALL);

```

```

LogComponentEnable ("DsrRouteCache", LOG_LEVEL_ALL);
LogComponentEnable ("DsrMaintainBuffer", LOG_LEVEL_ALL);
LogComponentEnable ("DsrRreqTable", LOG_LEVEL_ALL);
LogComponentEnable ("DsrErrorBuffer", LOG_LEVEL_ALL);
LogComponentEnable ("DsrNetworkQueue", LOG_LEVEL_ALL);
#endif

NS_LOG_INFO ("creating the nodes");

// General parameters
uint32_t nWifis = 10;
uint32_t nSinks = 1;
double TotalTime = 100.0;
double dataTime = 60.0;
double ppers = 1;
uint32_t packetSize = 64;
double dataStart = 100.0; // start sending data at 100s

//mobility parameters
double pauseTime = 0.0;
double nodeSpeed = 20.0;
double txpDistance = 100.0;
std::string rate = "0.512kbps";
std::string dataMode ("DsssRate11Mbps");
std::string phyMode ("DsssRate11Mbps");

//Allow users to override the default parameters and set it to new ones from CommandLine.
CommandLine cmd;

cmd.AddValue ("nWifis", "Number of wifi nodes", nWifis);
cmd.AddValue ("nSinks", "Number of SINK traffic nodes", nSinks);

```

```

cmd.AddValue ("rate", "CBR traffic rate(in kbps), Default:8", rate);

cmd.AddValue ("nodeSpeed", "Node speed in RandomWayPoint model, Default:20",
              nodeSpeed);

cmd.AddValue ("packetSize", "The packet size", packetSize);

cmd.AddValue ("txpDistance", "Specify node's transmit range, Default:300", txpDistance);

cmd.AddValue ("pauseTime", "pauseTime for mobility model, Default: 0", pauseTime);

cmd.Parse (argc, argv);

SeedManager::SetSeed (10);

SeedManager::SetRun (1);

NodeContainer adhocNodes;

adhocNodes.Create (nWifis);

NetDeviceContainer allDevices;

NS_LOG_INFO ("setting the default phy and channel parameters");

Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode", StringValue
(phyMode));

Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue
("2200"));

// disable fragmentation for frames below 2200 bytes

Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold",
                    StringValue ("2200"));

NS_LOG_INFO ("setting the default phy and channel parameters ");

WifiHelper wifi;

wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();

YansWifiChannelHelper wifiChannel;

wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");

wifiChannel.AddPropagationLoss ("ns3::RangePropagationLossModel", "MaxRange",

```

```

        DoubleValue (txpDistance));

wifiPhy.SetChannel (wifiChannel.Create ());

// Add a mac and disable rate control

WifiMacHelper wifiMac;

wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode",

        StringValue (dataMode), "ControlMode",StringValue (phyMode));

wifiMac.SetType ("ns3::AdhocWifiMac");

allDevices = wifi.Install (wifiPhy, wifiMac, adhocNodes);

NS_LOG_INFO ("Configure Tracing.");

AsciiTraceHelper ascii;

Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream ("dsrtest.tr");

wifiPhy.EnableAsciiAll (stream);

MobilityHelper adhocMobility;

ObjectFactory pos;

pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");

pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=100.0]"));

pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=500.0]"));

Ptr<PositionAllocator> taPositionAlloc = pos.Create ()->GetObject<PositionAllocator> ();

std::ostringstream speedUniformRandomVariableStream;

speedUniformRandomVariableStream << "ns3::UniformRandomVariable[Min=0.0|Max="

        << nodeSpeed<< "]";

std::ostringstream pauseConstantRandomVariableStream;

pauseConstantRandomVariableStream << "ns3::ConstantRandomVariable[Constant="

        << pauseTime<< "]";

adhocMobility.SetMobilityModel ("ns3::RandomWaypointMobilityModel",

```

```

        "Speed", StringValue (speedUniformRandomVariableStream.str ()),
        "Pause", StringValue (pauseConstantRandomVariableStream.str ()),
        "PositionAllocator", PointerValue (taPositionAlloc)
    );

    adhocMobility.Install (adhocNodes);

    InternetStackHelper internet;

    DsrMainHelper dsrMain;

    DsrHelper dsr;

    internet.Install (adhocNodes);

    dsrMain.Install (dsr, adhocNodes);

    NS_LOG_INFO ("assigning ip address");

    Ipv4AddressHelper address;

    address.SetBase ("10.1.1.0", "255.255.255.0");

    Ipv4InterfaceContainer allInterfaces;

    allInterfaces = address.Assign (allDevices);

    uint16_t port = 9;

    double randomStartTime = (1 / ppers) / nSinks;

    //distributed btw 1s evenly as we are sending 4pkt/s

    for (uint32_t i = 0; i < nSinks; ++i)
    {
        PacketSinkHelper sink ("ns3::UdpSocketFactory", InetSocketAddress
                               (Ipv4Address::GetAny (), port));

        ApplicationContainer apps_sink = sink.Install (adhocNodes.Get (i));

        apps_sink.Start (Seconds (0.0));

        apps_sink.Stop (Seconds (TotalTime));

        OnOffHelper onoff1 ("ns3::UdpSocketFactory", Address (InetSocketAddress

```

```

        (allInterfaces.GetAddress (i), port)));

onoff1.SetAttribute ("OnTime", StringValue
    ("ns3::ConstantRandomVariable[Constant=1.0]"));

onoff1.SetAttribute ("OffTime", StringValue
    ("ns3::ConstantRandomVariable[Constant=0.0]"));

onoff1.SetAttribute ("PacketSize", UIntegerValue (packetSize));

onoff1.SetAttribute ("DataRate", DataRateValue (DataRate (rate)));

ApplicationContainer apps1 = onoff1.Install (adhocNodes.Get (i + nWifis - nSinks));

apps1.Start (Seconds (dataStart + i * randomStartTime));

apps1.Stop (Seconds (dataTime + i * randomStartTime));

}

NS_LOG_INFO ("Run Simulation.");

Simulator::Stop (Seconds (TotalTime));

Simulator::Run ();

Simulator::Destroy ();

}

```

**/\* Performance Comparison between OLSR and DSR Routing Protocols \*/**

```

#include <fstream>

#include <iostream>

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/mobility-module.h"

#include "ns3/wifi-module.h"

#include "ns3/olsr-module.h"

#include "ns3/dsr-module.h"

```

```

#include "ns3/applications-module.h"

using namespace ns3;

using namespace dsr;

NS_LOG_COMPONENT_DEFINE ("olsr-dsr-compare");

class RoutingExperiment
{
public:
    RoutingExperiment ();

    void Run (int nSinks, double txp, std::string CSVfileName);

    std::string CommandSetup (int argc, char **argv);

private:
    Ptr<Socket> SetupPacketReceive (Ipv4Address addr, Ptr<Node> node);

    void ReceivePacket (Ptr<Socket> socket);

    void CheckThroughput ();

    uint32_t port;

    uint32_t bytesTotal;

    uint32_t packetsReceived;

    std::string m_CSVfileName;

    int m_nSinks;

    std::string m_protocolName;

    double m_txp;

    bool m_traceMobility;

    uint32_t m_protocol;

};

RoutingExperiment::RoutingExperiment ()

: port (9),

```



```

bytesTotal (0),
packetsReceived (0),
m_CSVfileName ("odc1.csv"),
m_traceMobility (false),
m_protocol (2) //1->OLSR,2->DSR
{
}

static inline std::string
PrintReceivedPacket (Ptr<Socket> socket, Ptr<Packet> packet, Address senderAddress)
{
    std::ostringstream oss;

    oss << Simulator::Now ().GetSeconds () << " " << socket->GetNode ()->GetId ();
    if (InetSocketAddress::IsMatchingType (senderAddress))
    {
        InetSocketAddress addr = InetSocketAddress::ConvertFrom (senderAddress);
        oss << " received one packet from " << addr.GetIpv4 ();
    }
    else
    {
        oss << " received one packet!";
    }
    return oss.str ();
}

void RoutingExperiment::ReceivePacket (Ptr<Socket> socket)
{
    Ptr<Packet> packet;

```

```

Address senderAddress;

while ((packet = socket->RecvFrom (senderAddress)))
{
    bytesTotal += packet->GetSize ();
    packetsReceived += 1;
    NS_LOG_UNCOND (PrintReceivedPacket (socket, packet, senderAddress));
}
}

void RoutingExperiment::CheckThroughput ()
{
    double kbs = (bytesTotal * 8.0) / 1000;
    bytesTotal = 0;
    std::ofstream out (m_CSVfileName.c_str (), std::ios::app);
    if(m_txp>=7.5){
        out << (Simulator::Now ()).GetSeconds () << "," << packetsReceived << "," << m_nSinks
        << "," << m_protocolName << "," << ( m_txp + packetsReceived)<< "," << kbs << ""
        << std::endl;
        out.close ();
    }
    packetsReceived = 0;
    Simulator::Schedule (Seconds (1.0), &RoutingExperiment::CheckThroughput, this);
}

Ptr<Socket>
RoutingExperiment::SetupPacketReceive (Ipv4Address addr, Ptr<Node> node)
{
    TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");

```

```

Ptr<Socket> sink = Socket::CreateSocket (node, tid);

InetSocketAddress local = InetSocketAddress (addr, port);

sink->Bind (local);

sink->SetRecvCallback (MakeCallback (&RoutingExperiment::ReceivePacket, this));

return sink;
}

std::string
RoutingExperiment::CommandSetup (int argc, char **argv)
{
    CommandLine cmd;

    cmd.AddValue ("CSVfileName", "The name of the CSV output file name",
m_CSVfileName);

    cmd.AddValue ("traceMobility", "Enable mobility tracing", m_traceMobility);

    cmd.AddValue ("protocol", "1=OLSR;2=DSR", m_protocol);

    cmd.Parse (argc, argv);

    return m_CSVfileName;
}

int main (int argc, char *argv[])
{
    RoutingExperiment experiment;

    std::string CSVfileName = experiment.CommandSetup (argc,argv);

    //blank out the last output file and write the column headers

    std::ofstream out (CSVfileName.c_str ());

    out << "Simulation Second," << "Packets Received," <<

    "Number Of Sinks," << "Routing Protocol," << "End2End Delay," << "Throughput" <<

    std::endl;

```

```

out.close ();

int nSinks = 10;

double txp = 7.5;

experiment.Run (nSinks, txp, CSVfileName);
}

void RoutingExperiment::Run (int nSinks, double txp, std::string CSVfileName)
{
    Packet::EnablePrinting ();

    m_nSinks = nSinks;

    m_txp = txp;

    m_CSVfileName = CSVfileName;

    int nWifis = 20;

    double TotalTime = 200.0;

    std::string rate ("2048bps");

    std::string phyMode ("DsssRate11Mbps");

    std::string tr_name ("manet-routing-compare");

    int nodeSpeed = 20; //in m/s

    int nodePause = 0; //in s

    m_protocolName = "protocol";

    Config::SetDefault ("ns3::OnOffApplication::PacketSize",StringValue ("64"));

    Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue (rate));

    //Set Non-unicastMode rate to unicast mode

    Config::SetDefault("ns3::WifiRemoteStationManager::NonUnicastMode",StringValue

                        (phyMode));

    NodeContainer adhocNodes;

    adhocNodes.Create (nWifis);

```

```

// setting up wifi phy and channel using helpers

WifiHelper wifi;

wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();

YansWifiChannelHelper wifiChannel;

wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");

wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");

wifiPhy.SetChannel (wifiChannel.Create ());

// Add a mac and disable rate control

WifiMacHelper wifiMac;

wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",

    "DataMode",StringValue (phyMode),"ControlMode",StringValue (phyMode));

wifiPhy.Set ("TxPowerStart",DoubleValue (txp));

wifiPhy.Set ("TxPowerEnd", DoubleValue (txp));

wifiMac.SetType ("ns3::AdhocWifiMac");

NetDeviceContainer adhocDevices = wifi.Install (wifiPhy, wifiMac, adhocNodes);

MobilityHelper mobilityAdhoc;

int64_t streamIndex = 0; // used to get consistent mobility across scenarios

ObjectFactory pos;

pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");

pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=300.0]"));

pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=1500.0]"));

Ptr<PositionAllocator> taPositionAlloc = pos.Create ()->GetObject<PositionAllocator> ();

streamIndex += taPositionAlloc->AssignStreams (streamIndex);

std::stringstream ssSpeed;

ssSpeed << "ns3::UniformRandomVariable[Min=0.0|Max=" << nodeSpeed << "]";

```

```

std::stringstream ssPause;

ssPause << "ns3::ConstantRandomVariable[Constant=" << nodePause << "]\n";

mobilityAdhoc.SetMobilityModel ("ns3::RandomWaypointMobilityModel",

                                "Speed", StringValue (ssSpeed.str ()),

                                "Pause", StringValue (ssPause.str ()),

                                "PositionAllocator", PointerValue (taPositionAlloc));

mobilityAdhoc.SetPositionAllocator (taPositionAlloc);

mobilityAdhoc.Install (adhocNodes);

streamIndex += mobilityAdhoc.AssignStreams (adhocNodes, streamIndex);

NS_UNUSED (streamIndex); // From this point, streamIndex is unused

OlsrHelper olsr;

DsrHelper dsr;

DsrMainHelper dsrMain;

Ipv4ListRoutingHelper list;

InternetStackHelper internet;

switch (m_protocol)

{

case 1:

    list.Add (olsr, 100);

    m_protocolName = "OLSR";

    break;

case 2:

    m_protocolName = "DSR";

    break;

default:

    NS_FATAL_ERROR ("No such protocol:" << m_protocol);

```

```

    }
if (m_protocol < 2)
{
    internet.SetRoutingHelper (list);
    internet.Install (adhocNodes);
}
else if (m_protocol == 2)
{
    internet.Install (adhocNodes);
    dsrMain.Install (dsr, adhocNodes);
}
NS_LOG_INFO ("assigning ip address");
Ipv4AddressHelper addressAdhoc;
addressAdhoc.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer adhocInterfaces;
adhocInterfaces = addressAdhoc.Assign (adhocDevices);
OnOffHelper onoff1 ("ns3::UdpSocketFactory", Address ());
onoff1.SetAttribute ("OnTime", StringValue
    ("ns3::ConstantRandomVariable[Constant=1.0]"));
onoff1.SetAttribute ("OffTime", StringValue
    ("ns3::ConstantRandomVariable[Constant=0.0]"));
for (int i = 0; i < nSinks; i++)
{
    Ptr<Socket> sink = SetupPacketReceive (adhocInterfaces.GetAddress (i),
        adhocNodes.Get (i));
    AddressValue remoteAddress (InetSocketAddress (adhocInterfaces.GetAddress (i), port));

```

```

onoff1.SetAttribute ("Remote", remoteAddress);

Ptr<UniformRandomVariable> var = CreateObject<UniformRandomVariable> ();

ApplicationContainer temp = onoff1.Install (adhocNodes.Get (i + nSinks));

temp.Start (Seconds (var->GetValue (10.0,11.0)));

temp.Stop (Seconds (TotalTime));

}

std::stringstream ss;

ss << nWifis;

std::string nodes = ss.str ();

std::stringstream ss2;

ss2 << nodeSpeed;

std::string sNodeSpeed = ss2.str ();

std::stringstream ss3;

ss3 << nodePause;

std::string sNodePause = ss3.str ();

std::stringstream ss4;

ss4 << rate;

std::string sRate = ss4.str ();

AsciiTraceHelper ascii;

Ptr<OutputStreamWrapper> osw = ascii.CreateFileStream ("odc.tr");

wifiPhy.EnableAsciiAll (osw);

NS_LOG_INFO ("Run Simulation."); CheckThroughput ();

Simulator::Stop (Seconds (TotalTime));

Simulator::Run (); Simulator::Destroy ();

}

```