**The Computer Communication Lab (236340)**

**Winter 2009-2010**

OLSRv2 protocol simulation and analysis

### Final Report

# Submitted by:

###### Assaf Israel

###### Eli Nazarov

###### Asi Bross

### Table of contents:

[1 Introduction 3](#_Toc255069073)

[1.1 OLSR introduction 3](#_Toc255069074)

[1.2 Neighborhood Discovery Protocol (NHDP) 3](#_Toc255069075)

[1.3 Optimized Link State Routing Ver. 2 (OLSRv2)](#_Toc255069076)  4

[2 Project Description 6](#_Toc255069077)

[2.1 Purpose 6](#_Toc255069078)

[2.2 Main Goals 6](#_Toc255069079)

[2.2.1 MPR Selection Analysis 6](#_Toc255069080)

[2.2.2 Data Send Mode Analysis 6](#_Toc255069081)

[2.2.3 Topology Analysis 6](#_Toc255069082)

[2.3 Analysis and Conclusions 6](#_Toc255069083)

[2.4 Output 10](#_Toc255069084)

[2.4.1 GUI 10](#_Toc255069085)

[2.4.2 SQL 11](#_Toc255069086)

[2.4.2.1 SQL Table Structure 11](#_Toc255069087)

[2.4.2.2 Event Types 11](#_Toc255069088)

[3 Topology Layout 13](#_Toc255069089)

[4 Implementation 14](#_Toc255069090)

[4.1 Class Diagram 14](#_Toc255069091)

[4.2 GUI 14](#_Toc255069092)

[4.2.1 Input Parameters 14](#_Toc255069093)

[4.2.1.1 Simulation parameters 14](#_Toc255069094)

[4.2.1.2 Layout parameters 15](#_Toc255069095)

[4.2.1.3 Protocol parameters 15](#_Toc255069096)

[4.2.2 World Topology 16](#_Toc255069097)

[4.3 Log 16](#_Toc255069098)

[4.4 Event Generator 16](#_Toc255069099)

[4.5 Dispatcher 17](#_Toc255069100)

[4.6 Topology Manager 17](#_Toc255069101)

[4.7 Protocol 18](#_Toc255069102)

[4.7.1 Protocol Layer 18](#_Toc255069103)

[4.7.2 NHDP Layer 18](#_Toc255069104)

[4.7.3 OLSRv2 Layer 18](#_Toc255069105)

[5 Appendixes 18](#_Toc255069106)

[5.1 Requirements and Assumptions 18](#_Toc255069107)

[5.2 Future Work 18](#_Toc255069108)

# Introduction

## OLSR introduction

The Optimized Link State Routing Protocol (OLSR) is developed for mobile ad hoc networks. It operates as a proactive protocol (table driven) which exchanges topology information with other stations of the network regularly. It was developed to work independently from other protocols. Likewise it makes no assumptions about the underlying link-layer. The protocol inherits the stability of a link state algorithm and has the advantage of having routes immediately available when needed due to its proactive nature. OLSR stack location is beneath the transport layer as supplement to the network layer. Each station can operate as a router to different implementations of local networks.

OLSR is well suited to larger and denser mobile networks then other naive routing options. The larger and more dense a network, the more optimization can be achieved. OLSR uses hop-by-hop routing, i.e., each station uses its local information to route packets.

OLSR is designed to work in a completely distributed manner and does not depend on any central entity. The protocol does not require reliable transmission of control messages since each station sends control messages periodically, and can therefore sustain a reasonable loss of some messages.

## Neighborhood Discovery Protocol (NHDP)

The NHDP objective is to maintain a 1-hop and 2-hop neighbors set for each station in a Mobile Ad-hoc Network (MANET). This enables the station, to use this information later, when implementing the OLSR protocol. NHDP is a table driven protocol, meaning maintenance messages are used in order to maintain the stations databases.

NHDP uses only one type of messages, called *"Hello" messages*. These messages are sent between each station and every station that is in its broadcast radius (1-hop neighbors), but not beyond. Meaning, these messages are not forwarded by any receiver. Their role is passing information about a station's 1-hop neighbors to its 1-hop neighbors. A station, which receives a message, can add the sender to its 1-hop neighbors set. If the receiver is stated in the senders 1-hop neighbors set, he can state that the link is bi-directional (i.e. messages are send and received by both parties). The receiver construct a 2-hop neighbors set by joining all the received sender's 1-hop neighbors sets and subtracting its own 1-hop neighbors set. Meaning, station C is a 2-hop neighbor of station A, if it's a 1-hop neighbor of station B that is a 1-hop neighbor of station A, and C is not a 1-hop neighbor of station A.

Figure .

*"Hello" messages* are sent by every participating station, and by every station's *network interface*, in a manner that will extend the network connectivity as much as possible. E.g. – station C can be a 2-hop neighbor of station A, via station B but not share the same *network interface* with station B that B shares with A. This behavior allows the protocol to build the most comprehensive and flexible 1-hop & 2-hop topology map, so future *data messages* could "switch" *network interfaces* in order to achieve the best and fastest possible connectivity to their destination.

*"Hello" messages* are sent periodically according to an adjustable *Refresh Interval*. This means that if a message from a 1-hop neighbor is not received by a station, then this connection is considered lost, and the neighbor is removed from the 1-hop neighbors set. At the next *"Hello" message* that will be sent from the stations that noticed their neighbor's disappearance, the change will be noted so other neighbors could be aware of the change as quickly as possible.

A station does not have to wait until the next scheduled *"Hello" message* transmission, in order to alert the change (new/lost/changed status link). He can issue a dynamically scheduled *"Hello" message* after a certain *Minimum Interval* from the last message transmission.

## Optimized Link State Routing Ver. 2 (OLSRv2)

Optimized Link State Routing Ver.2 (*OLSRv2*) is a proactive protocol (table driven) that uses NHDP protocol described in section 1.2. *OLSRv2* is optimized routing protocol for MANET networks that can handle dynamic topology.

The main concept of the protocol is the use of Multipoint Relays (MPRs). MPRs are used in the OLSR protocol to minimize the overhead of flooding messages in the network by reducing redundant retransmissions in the same region. Each station in the network selects a set of stations in its symmetric 1-hop neighborhood (see ‎1.2) that may retransmit its messages. This set of selected neighbor stations is called the "Multipoint Relay" (MPR) set of that station. The neighbors of station X which are not in its MPR set, receive and process broadcast messages but do not retransmit broadcast messages received from station X. In route calculation, the MPRs are used to form the route from a given station to any destination in the network.

The set of MPRs should cover all the 2-hop (see ‎1.2) neighbors of the station, meaning that the union of all neighbors of MPRs gives the group of all stations until the 2-hop degree of the MPRs selector.

Basically, the route between two stations in the network is a sequence of hops through MPRs. The last MPR is the target station or, the target is a 1-hop neighbor of the last MPR.

A station’s request from another to be its MPR is passed via attaching new information to *“Hello” messages* generated by the underlying *NHDP*.

*OLSRv2* defines only one type of message, *Topology Control (TC).*

*“Hello” messages* are being modified by *OLSRv2* to contain the selected MPRs set of the source station. Upon receiving *“Hello” messages*, if the target station was selected as MPR, it will update its status to be an MPR for the source station. *“Hello” messages* are never forwarded by the *OLSRv2* protocol.

*“TC” messages* include the set of all stations that selected the source station as a MPR. Also, this message may contain relevant information regarding the different interfaces of the station and the network (if such exist) that are connected to this station. *“TC” messages* are always forwarded, unless this message was already sent by this station, and are broadcasted to all stations in the network.

*OLSRv2* keeps different information tables, Topology Information Base, that hold information regarding the MPR selectors of this station, network topology information and information regarding routes to all stations in the network.

Furthermore, *OLSRv2* keeps information regarding all *“TC” messages* that this station receives processes or forwards – Processing and Forwarding Information Base. This information helps to lower the amount of *“TC” messages* that are being sent by each station.

*“TC” messages* are being generated and transmitted periodically by the station once in a specific time, or due to some changes in the topology of the station. Topology change is basically a change in contents of Topology Information Base.

When a packet needs to be transmitted from station A to station B, station A will send it to the first MPR in the current route known to station B, and each MPR will forward the packet to the next MPR in the route.

Figure ‎1.2

# Project Description

## Purpose

The main purpose of this project was to create a simulator for the *OLSRv2* protocol. Investigate and analyze the behavior and performance of the protocol by setting different network, protocol and layout characteristics. The simulator was created with a wide range of adjustable parameters that can produce a wide spectrum of analytic measurements.

## Main Goals

The simulator that we created can produce a wide range of analytic measurement; hence we decided to focus our research on the following topics.

### MPR Selection Analysis

MPR selection can have a significant impact on the network utilization, a minimal selection of 1-hop MPRs that cover the entire 2-hop neighborhood of a single station may result in a better network utilization in comparison to a selection of the entire 1-hop neighborhood as MPRs.

### Data Send Mode Analysis

We decided to compare between the propagation of data packets through the MPRs vs. the propagation of data packets through the entire 1-hop neighborhood. In MPR mode, the data packets can be rerouted through the entire known topology (to each relay station), while in the Normal mode, data packets can only be rerouted through the selected MPRs.

### Topology Analysis

The simulator we created supports setting the topology layout in one of two modes, either UNIFORM or CLUSTER. We decided to investigate the throughput received in each of these topology layouts. Some of the feedback we anticipated was to see a drop down in utilization when switching to Cluster mode, because of bottleneck congestion.

## Analysis and Conclusions

We've automated the simulation to take measures of different protocol settings in uniform environment settings. The environment settings we've chosen are a grid of 500x500 with 30 stations capable of transmitting/receiving in a radius of 100 distance units. The Hello and TC intervals were set to 30 & 90 time units respectfully.

The following figures illustrate the simulation in various protocol settings.

Figure .

­

Figure .

Given a Uniform topology layout and a coherent Data transmission method (Through all neighbors) we can isolate the MPR Selection effect quite well and see that Selecting an entire station neighborhood as it's MPR set, has drastic effects on utilization, especially when taking movement into effect. When selecting the entire neighborhood as MPR it is unimportant whether Data is sent by MPR alone or by all Neighbors, since they are both the same.

Figure .

One can see that limiting the Data transfer to be handled only by the MPRs have a slightly cumbersome effect on utility, especially when taking movement into effect.

Figure .

Topology had drastic effects on the results of our measurements. We can see that pass a curtain point of, when the rate for new data packets is too high, the bottleneck nodes succumb to load and utilization drop sharply. Also, since movement of stations was limited to within their clusters we do not see much difference in utilization between the three lowest speeds of movement. A possible explanation is the topology support for a low change rate in a station neighborhood. Although stations keep moving, their surrounding is moving with them in a fashion where all the protocol database tables need very little change.

Figure .5

Figure ‎2.5 shows an interesting phenomenon. It illustrates how the fragility of this protocol, when the load of extra TC messages in a cluster can have a drastic effect on utilization. Since very few nodes acts as "gateways" between clusters, they are now being interrupted by continues TC messages, that takeout a lot of bandwidth.

Figure .6

In an opposite fashion to the observation in a Uniform topology, selecting the mode in which data could be passed only through MPRs, actually helped keeping the utilization high even in higher data packets rates. Although it had a slight decrease in utilization in lower data rates, it could very well be a solution to these kinds of scenarios.

### Utilization

OLSRv2 protocol does not excel in utilization. Throughout our experiments we yet to see the protocol go over 0.1 utilization, and even this number was achieved in a static environment, with a very high Hello & TC intervals.

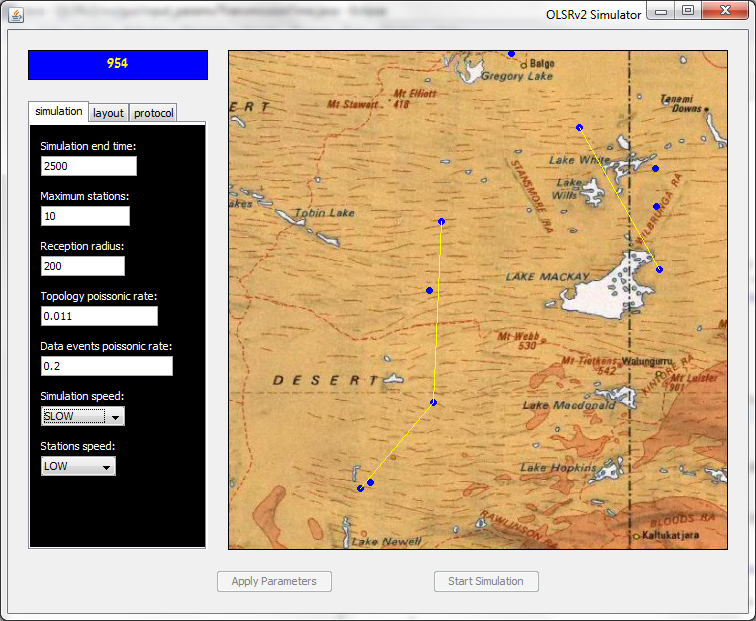
In many of our experiments we noticed the drastic effect that the Hello & TC intervals had, as well as the effect station population density had. We've noticed that although higher densities have better connectivity, they pose a major downwards force when it comes to congestion produced primarily because of the protocol overhead in Flood messages (TC).

Currently the protocol does not support Jitter optimizations that could have raised the utilization much, especially in the dynamic scenarios.

## Output

### GUI

The output generated by the GUI shows the state of the stations and the data packets transmissions while they occur, the size attributes of the simulated world can be altered through the "World size" entry in the Layout tab. The speed in which the graphical simulation is presented can be changed during the simulation run by altering the "Simulation speed" entry in the Simulation tab.



### SQL

The simulator we created simulates the OLSRv2 protocol under various conditions. in order to analyze it's output we use a logger module (see 4.3) that logs all of the different events (see 2.3.2.2) while the simulation is running, when the simulation ends, all of the logged events are recorded into an SQL database (Our choice was a simple MySQL instance). Since all of the events that occurred during the execution are saved into the SQL database when the simulation ends, different queries can be performed on the database table, and analyzed data can be pulled out.

We used a MySQL eclipse plug-in in order to create and modify the SQL table, an SQL proxy can be configured to save the data generated on a local host or on a designated remote host.

#### SQL Table Structure

The SQL table consists of the following columns.

**INDX** - A progressive index of the table's rows.

**VIRTUAL\_TIME** - The virtual time the event occurred.

**NODE\_ID** – A UUID that represents the station that caused the event to be logged.

**EVENT\_TYPE** - The type of the event (see 2.3.2.2).

**GLOBAL\_SOURCE** - The initiator source of the event (i.e. the source of the Data Message).

**LOCAL\_SOURCE** - The UUID of the station from which the message was received.

**LOCAL\_TARGET** - The UUID of the station that received the message.

**GLOBAL\_TARGET** - The final recipient of the event (i.e. the target of the Data Message).

**X\_COOR** – The x coordinate of the station in the plane.

**Y\_COOR** - The y coordinate of the station in the plane.

**RADIUS** - The reception/transmission radius of the station.

**LOST** – A Boolean indicator that states if the event was lost (i.e. in message events).

**ERROR** - Boolean flag stating if some error occurred.

**DETAILS** - More details regarding this event.

#### Event Types

The followings are the main types of events that can be logged into the database under the EVENT\_TYPE column.

**DATA\_SENT\_EVENT** – Processing the data message.

**BUSSY\_MSG\_IGNORED** – The station is busy, thus ignored the received message.

**DATA\_SENT\_FROM\_SOURCE** – Message was sent from the source.

**DATA\_REACHED\_2\_TARGET** – Message reached its destination.

**DATA\_NOT\_SENT\_NO\_ROUTE** – Message was not sent because the source couldn't find a route.

**DATA\_LOSS\_NO\_ROUTE** – Relay station couldn't find a route, thus the message was lost.

**DATA\_IGNORED\_AT\_RELAY** – Station received a message that was not addressed to her.

**DATA\_SENT\_FROM\_RELAY** – Relay station sent the message to the next station in the route.

**DATA\_REACHED\_2\_RELAY** – Relay station on the route received data message.

**LOCAL\_TARGET\_NOT\_PHYSIBLE** – There was a message sent to a station that doesn’t exist in its 1-hop neighborhood (the station didn’t learn yet that the station is no longer there).

**TTL\_LIMIT\_REACHED** – TTL of the data message is over (the message is dropped).

# Topology Layout

The *OLSRv2* illustrates a MANET network of stations that moves dynamically. The simulator simulates both high and low dense areas. We have implemented variations for the topology of the stations. Uniform topology is a topology where the stations are spread uniformly over the plane. The Cluster topology is when the stations are spread over the plane in groups. Using this topology we can check when the messages are routed through only few stations in order to reach stations in other clusters (“bottle-neck” stations).

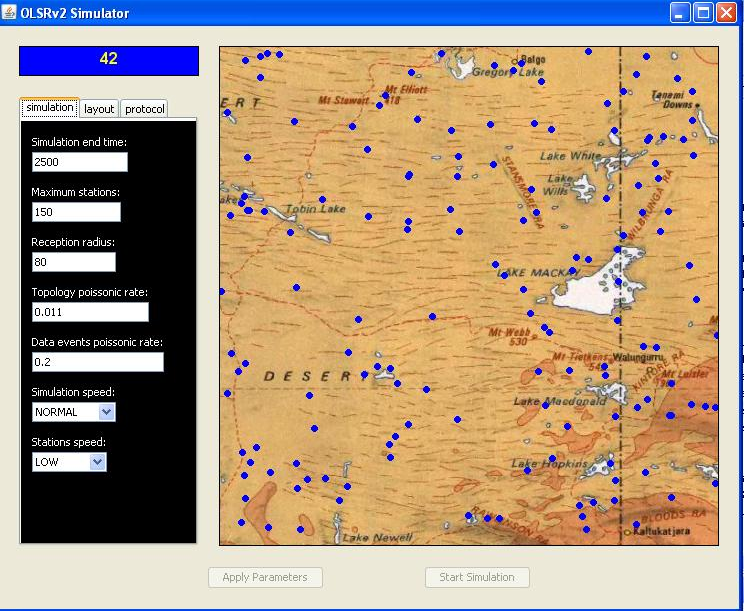


Figure 3-1: Uniform formation

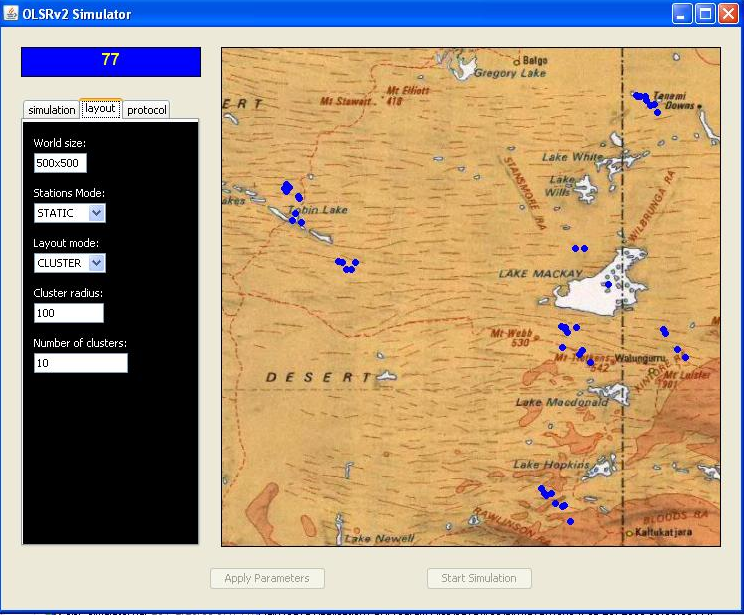


Figure 3-2: Cluster Distribition

# Implementation

## Class Diagram



## GUI

The GUI allows the user to set different specification for the simulator execution it enables the user to see the current state of the MANET by painting a graphical presentation of the stations and the transmission of data messages.

The GUI is designed using swing (a graphical library in java). The world state is dynamically updated using a dedicated thread the pulls the topology events from a dedicated queue and updates the graphical state of the world, the events are inserted into the dedicated queue by the dispatcher module during the simulation execution. The implementation of the GUI maintains its own simulation ticks, hence makes it possible to the user to set the simulation speed.

### Input Parameters

#### Simulation parameters

* Simulation end time – The amount of logical time units the simulation execution should run.
* Maximum stations – The maximum number of stations that can exist simultaneously. In static mode this represent the number of stations (No topology changes occur), in dynamic mode it represent the initial and maximum number of stations.
* Reception radius – Represents the radius of reception of each station.
* Topology poissonic rate – The expected number of topology events that can occur in a single time unit.
* Data events poissonic rate - The expected number of data messages events that can occur in a single time unit.
* Simulation speed – The current speed of the simulation graphic display, it sets the rate at which a single simulation tick occurs, this parameter can be changed during the simulation run, since this only effect the GUI representation it does not have any effect on the simulation results. The values for this parameter are SLOW, NORMAL, FAST, REAL TIME – the "REAL TIME" value displays the current simulation state while the others may have a certain delay in the graphic presentation.
* Stations speed – In dynamic mode it set the amount of time the station moves between two points.

#### Layout parameters

* World size – Sets the size of the area in which stations can exist, the input value is in the format of <width>x<height>, in the GUI representation it is changed dynamically and covers a pixel area in the given size, the boundaries are between 100x100 and 500x500.
* Stations mode – Set the nature of the stations, STATIC value means that all stations are created in the beginning of the simulations and they don’t change their position, DYNAMIC value means that stations are constantly moving in space and MIXED means that Stations will decide to move every once in a while (in a non-deterministic fashion), it also means that new stations can be created and existing stations can be destroyed.
* Layout mode – Set the topology layout of the stations, the values are UNIFORM and CLUSTER, the latter means that the stations are organized in groups.
* Cluster radius – The radius of a single cluster, all the stations are inside this radius. Applicable only if layout mode is CLUSTER.
* Number of clusters – The total number of clusters in the topology. Applicable only if layout mode is CLUSTER.

#### Protocol parameters

* Entry validity period – The amount of unit time a table entry is valid in the station's control tables.
* Hello message interval – The amount of unit time between consecutive hello message transmissions of the same station.
* TC message interval - The amount of unit time between consecutive TC message transmissions of the same station.
* Transmission time – The amount of unit time it takes for a transmission of a packet.
* Protocol mode – The mode in which control packets are being transferred, NORMAL mean through all 1-hop neighbors, ALL MPRs mean through all the MPRs of a station.
* Protocol data send mode – The mode in which data packets are being transferred, NORMAL means through any 1-hop neighbor, MPRs mean through MPRs only.

### World Topology

The stations location are represented as circles on a <width>x<height> panel that represents the world in which the stations reside. The stations distribution can be defined through the Layout mode entry which enables a uniform or cluster distribution. Data packets that are sent between stations are represented as a line that connects the two participating stations and is visible for a period of "Transmission time" as specified by the user.

The graphical updated state of the stations are handled using a dedicated thread that maintains a correct presentation of the simulation current state. The dedicated thread keeps the state updated by pulling events from a dedicated queue, the events are analyzed and the topology changes occur, in addition the queue holds events of data packets that are being sent by the stations, the dedicated thread analyzes these events and alerts the GUI to draw a line between the two participating stations, this line represents the transmission of a data packet. Control messages (HELLO/TC) are not represented in the GUI as packets being, though they are.

## Log

This module is responsible for all the data handling of the system. It acts as a hub for different components to log their data in.

The log module saves the different events into an excel spread sheet while the simulation is running, and once the simulation is over it dumps all the gathered data into a predefined SQL server, hence in the end of the simulation run, the user receives both an excel sheet and an SQL database that hold all of the events that occurred during the simulation.

The Log module is implemented as a singleton; hence it can be accessed easily from all the participating components. The Log module uses an SQL Writer module that encapsulates all the interactions with the SQL server. The Log module saves all the data into the SQL database in the end of the simulation run; hence keeps the simulation execution free from any database write delays.

## Event Generator

The *Event Generator* is responsible for the creation of two types of events:

* *Topology events* – Creation/Migration/Destruction of stations.
* *Data messages* – These events simulate data, that’s being received by the OLSR Layer at a certain station, and is designated for another station in the network.

In order to safely create these events, the Event Generator needs to hold a set of the stations labels and coordination. It queries the set prior to any event generated in order to maintain consistency.

For example, upon creation of a new station the *Event Generator* checks that the station does not already exists, either its label or coordination. Another example is when creating *Data event*, the *Event Generator* should check if the source or target stations exists.

The *Event Generator* is implemented as a singleton, hence only one exist during the simulation run, it maintains an internal tick mechanism that is responsible to generate events according to the their poissonic rate as supplied by the user.

## Dispatcher

The *Dispatcher*’s name basically reveals its role. It is designed to retrieve the top task from the *Tasks Queue* (The next task in a chronological order), inspect it, and redirect it to the relevant objects which in turn process it.

The *Dispatcher* specific behavior depends on the type of task it has retrieved from the *Tasks Queue*:

* *Topology events* – are passed directly to the *Topology Manager*, who in turn updates its records based on the information given.
* *Data messages* – are passed to the *OLSRv2 Layers* which identifies as the source stations of the messages.
* *“Hello”* or *“TC” messages* – are passed to all of the *OLSRv2 Layers* that represents stations which are within the *Reception Radius* of the originator station location. The list of stations which answer this criterion is given to the *Dispatcher* by requesting it from the *Topology Manager* which has the knowledge regarding the stations physical whereabouts in space.

Whenever the *Dispatcher* notices that an event destination is missing (e.g. *Data message* is sent to a non-existing station) it will log this event at the *Log* and will discard the event.

## Topology Manager

The *Topology Manager*'s job is to maintain information about the simulated physical attributes of stations. The combined data of all the stations attributes defines the network physical picture at a certain time. The attributes that are stored per station in the *Topology Manager* are:

* *Coordinates* in space – the simulated physical location at the defined 2D space. These Coordinates can be changed by the *Topology Manager* upon receiving a *Topology event* from the *Dispatcher* which instructs it to migrate a certain station to a new location (considering that the new location is still within the defined space boundaries)
* *Reception Radius –* each station might have a different reception radius, in this version of the simulator the option isn't given to the user although it is well supported, thus all stations uses the default assigned reception radius.

The topology manager supplies a wide range of options regarding the current topology state, it allows:

1. Creating a new station.
2. Destroying an existing station.
3. Changing the position of a given station.
4. Receiving a station's full 1-hop neighborhood. (inside its reception radius)
5. Checking if a station exists, either by its location or by its ID.

## Protocol

This module implements the *OLSRv2* logic of a single station in a MANET environment. The module is split into three layers:

* *Protocol Layer*
* *NHDP Layer*
* *OLSR Layer*

### Protocol Layer

This is a logical layer that makes the connection between the different parts of the simulator and the protocol layers. Due to the fact that the simulator is event driven all messages are being sent as events to the dispatcher and all the intervals (Hello/TC) are measured by special events, thus when the dispatcher processes an event it invokes an appropriate method in the protocol layer of the station. This method translates the event received from the dispatcher to an appropriate message of the protocol or an event related to the work flow, e.g. Hello/TC interval ended, and passes it for process in the protocol layers. In addition this layer is responsible for generating, processing and routing data messages according to different topology data sets that are maintained by the OLSRv2 and NHDP layers.

### NHDP Layer

Implements the NHDP layer, NHDP layer is responsible for processing and generating *Hello messages*. Upon receiving *Hello messages* this layer updates the different data sets it maintains and generates *Hello messages* if any change happened in the 1-hop topology of the station.

### OLSRv2 Layer

Implements the *OLSRv2* logic, this layer is responsible for adding additional information to the generated *Hello messages* (MPR information) as detailed in the *OLSR* description (Section ‎1.3), as well as generating and processing *TC messages* and *Data messages*. Upon receiving a *TC message* the layer updates different data sets that hold information regarding the global topology of the network; it re-calculates MPRs for this station and calculates routes to all the stations it learned.

# Appendixes

## Requirements and Assumptions

* Low level implementation details as described in **draft- ietf-manet-olsrv2-10** and in **draft-ietf-manet-nhdp-10** like packet formation or jitter aspects are not in the scope of this simulation.
* The simulator is implemented in Java, so JVM is needed for execution.
* My SQL should be installed in order to get SQL support.

## Future Work

* Our simulation allows many more comparisons to be made, simply because much of the information is stored in an SQL database. We've frequently used queries to observe numbers like the data bounce percentage, it's reasons (connectivity loss, congestion, improper routing tables etc.).
* Relative short amount of work can be done to add Jitter support. This was not done during the project because of priority and scope issues.
* We've made it easy to support more Topology inquiries in the form of Layouts. Please see the project Javadoc for more information.