**The Computer Communication Lab (236340)**

**Winter 2009-2010**

OLSRv2 protocol simulation and analysis

### First Report

# Submitted by:

###### Assaf Israel

###### Eli Nazarov

###### Asi Bross

### Table of contents:

[1 Introduction 3](#_Toc243570883)

[1.1 OLSR introduction 3](#_Toc243570884)

[1.2 Neighborhood Discovery Protocol (NHDP) 3](#_Toc243570885)

[1.3 Optimized Link State Routing Ver. 2 4](#_Toc243570886)

[2 Project Description 4](#_Toc243570887)

[2.1 Purpose 4](#_Toc243570888)

[2.2 Main Goals 4](#_Toc243570889)

[2.2.1 Utilization Analysis 4](#_Toc243570890)

[2.2.2 Network Reliability 4](#_Toc243570891)

[2.2.3 Topology 4](#_Toc243570892)

[2.2.4 Scalability 4](#_Toc243570893)

[2.3 Stretched Goals 4](#_Toc243570894)

[2.3.1 Topology mobility 4](#_Toc243570895)

[2.3.2 Secondary MPR sets 4](#_Toc243570896)

[2.4 GUI 4](#_Toc243570897)

[2.5 Output 4](#_Toc243570898)

[3 General Layout 4](#_Toc243570899)

[4 Main Modules 4](#_Toc243570900)

[4.1 Various modules details 5](#_Toc243570901)

[5 Main Algorithm 5](#_Toc243570902)

[5.1 Event Generation and Manipulation 5](#_Toc243570903)

[5.2 Throughput Analysis 5](#_Toc243570904)

[6 Appendixes 5](#_Toc243570905)

[6.1 Appendix 1: Estimated project timetable 5](#_Toc243570906)

[6.2 Appendix 2: Requirements and Assumptions 5](#_Toc243570907)

[6.3 Appendix 3: Screen shots 5](#_Toc243570908)

# Introduction

## OLSR introduction

The Optimized Link State Routing Protocol (OLSR) is developed for mobile ad hoc networks. It operates as a table driven, proactive protocol which exchanges topology information with other nodes 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.

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

Nodes select MPRs such that there exist a path to each of its 2-hop neighbors via a node selected as an MPR. These MPR nodes then source and forward control messages called TC that contains the MPR selectors. The forwarding path for TC messages is not shared among all nodes but varies depending on the source.

OLSR is well suited to large and dense mobile networks, as the optimization achieved using the MPRs works well in this context. The larger and more dense a network, the more optimization can be achieved. OLSR uses hop-by-hop routing, i.e., each node 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 node 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 node in a Mobile Ad-hoc Network (MANET). This enables the node, 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 nodes databases.

NHDP uses only one type of messages, called *"Hello" messages*. These messages are sent between each node and every node that is in its broadcast radius (1-hop neighbors), but not beyond. Meaning, these messages are not forwarded by any receiver. There role is passing information about a node's 1-hop neighbors to its 1-hop neighbors. A node, 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 can also 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, node C is a 2-hop neighbor of node A, if it's a 1-hop neighbor of node B that is a 1-hop neighbor of node A, and C is not a 1-hop neighbor of node A.

Figure ‑

*"Hello" messages* are sent by every participating node, and by every node's *network interface*, in a manner that will extend the network connectivity as much as possible. E.g. – node C can be a 2-hop neighbor of node A, via node B but not share the same *network interface* with node 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 node, than 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 nodes that noticed their neighbor's disappearance, the change will be noted so other neighbors could be aware of the change as quick as possible.

A node 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 curtain *Minimum Interval* from the last message transmission.

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

Optimized Link State Routing Ver.2 (OLSRv2) is 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 sub-set of 1-hop neighbors (see 1.2) of each station. Each MPR should have a bi-directional (symmetric) with its selector, and the set of MPRs should cover all the 2-hop (see 1.2) neighbors of the node, meaning that the union of all neighbors of MPRs gives the group of 2-hop neighbors of the node (MPR selector).

MPRs are responsible for forwarding control massages, and are used to calculate a route to all known nodes in the network. Basically, the route between two nodes in the network is a sequence of hops through MPRs. The last MPR is the target node or the target is a 1-hop neighbor of the last MPR.

A node’s request from another to be an MPR for it, is passed via attaching new information to *“Hello” messages* generated by the underlying *NHDP*.

OLSRv2 defines only one type of massage, Topology Control (TC). However, OLSRv2 also uses and extends Hello massages that are used and owned by NHDP protocol.

The Hello massages are being modified by OLSRv2 to contain the selected MPRs set of the source node. Upon receiving Hello massage, if the target node was selected as MPR, it will update its status to be an MPR for the source node. Hello massages are never forwarded by the OLSRv2 protocol.

TC massages include the set of all nodes that selected the source node as a MPR. Also, this massage may contain relevant information regarding e.g. the different interfaces of the node and the network (if such exist) that are connected to this node. TC massages are always forwarded, unless this massage was already sent by this station, and are broadcasted to all nodes in the network.

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

Furthermore, OLSRv2 keeps information regarding all TC massages that this station received, processed or forwarded – Processing and Forwarding Information Base. This information helps to lower the amount of TC massages that are being sent by each node.

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

When a packet needs to be transmitted from node A to node B, node A will send it to the first MPR in the current route known to node A. 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 is to create a simulator for OLSRv2 (and NHDP) protocol. Investigate and analyze the behavior and performance of the protocol with different characteristics of the network.

## Main Goals

The simulator will implement the following options:

### Utilization Analysis

Analysis of the optimization achieved using the MPRs will be measured in comparison to a protocol where each node knows the entire world topology. This will be tested by setting the entire 1-hop neighborhood as MPRs for each node.

We will expect to see major improvement in the throughput when setting a minimal set of MPRs for each node.

### Network Reliability

We will compare between different link qualities to determine the effect on the network reliability, we will compare between networks of different message failure rate. We will test the cases where some messages "Hello" or "TC" do not get to their destination, how will it affect the connectivity of the network? , how will it affect the throughput of the network?

### Topology

We will analyze the differences in utilization and reliability under different field topology.

It will be interested to investigate topologies where certain nodes will become bottlenecks, i.e. the topology of clusters of nodes where few nodes are scattered between the clusters. In addition we will look into uniform, sparse and concentrated topologies.

### Scalability

It will be interesting to investigate how scalable the OLSRv2 protocol is, we will analyze the throughput while increasing the number of stations. We will compare the results of the OLSRv2 protocol to the results of a protocol where each node selects all of its neighbors as MPRs.

We expect to see a major improvement in scalability when we set each node with a minimal set of MPRs.

## Stretched Goals [Assaf]

### Topology mobility

When looking into a dynamic scenario, it will be interesting to see how robust the protocol is in regards to the rate of which topology changes. What's the effect, nodes movement has on Utilization? We'll expect to see Utilization holds firm until a curtain threshold of Mobility rate is crossed.

*Jitter Control* could be introduced here, to see its effect on Utilization and Protocol Reliability. This is a mechanism that is intended to cope with topology change issues by accelerating/slowing the rate of which control messages are being generated.

### Secondary MPR sets

When curtain node refuses to transfer data packets, or when it suddenly fails all together, it's possible to recover if nodes can define a backup plan, in the form of a secondary MPR set. These MPR sets are identical to the main MPR sets in type but different in value. Meaning, the MPR that are selected in the secondary set, are selected with the idea of producing a backup system to the main set, so most of them will not be included in the main MPR set, and they might even not be a minimal MPR selection.

Creating a *Secondary MPR set* creates excess load on CPU and Network Bandwidth as more *"Hello"* and *"TC" messages* are being transmitted. This of course, means the Utilization is damaged.

It will be interesting, comparing between Utilization as the Network Reliability between an implementation with and without *Secondary MPR sets* support.

## Output

The output of the simulator will be an Excel document detailing the session parameters and the data gathered. The user will then be able to easily create graphs and diagrams based on that data.

The output fields will be:

* Number of stations
* Map dimensions
* Stations location and movement
* Stations reception radius
* Total run time
* Throughput per station

The graphs that will be presented are:

* Total throughput versus number of stations
* Total throughput versus stations movement
* Total throughput versus area density

The final report will also include an analysis of stations topology on throughput, as well as comparison between different selections of MPR sets (Minimal Vs Maximal).

If asked, the team will provide more feedback on other aspects that will be included in the final report.

# General Layout

The OLSRv2 will illustrate a MANET network of nodes that move dynamically. The simulator will simulate both high and low dense areas. There will be no specific router that will route the massages but each node will calculate the routes by itself. Furthermore, all nodes can be spread uniformly over the area as well as in other formations e.g. several groups.

Figure 3-1: Group formation

Figure 3-2: Uniform Distribition

# Main Modules

The simulator will consist of the following main modules:

1. Log
2. GUI
3. Event Generator
4. Dispatcher
5. OLSRv2 Layer

Figure ‎4‑1shows the different modules interaction.



Figure ‑

## GUI

The *GUI* will allow the user to enter different specification for the simulator and view a graphical presentation of the MANET and the nodes.

Map of nodes will be presented as circles (or dotes) and each control massage or data massage that will be transmitted will be represented by a line between two nodes or in some other way.

The user will be able to choose initial formation of the node distribution in the area, this will be the initial configuration for the Topology Manager and Event Generator.

Furthermore, the user will be able to view different measurements and analysis reports at each time, by clicking on appropriate buttons that will be placed near the network map.

The GUI model will communicate with the *Log* model in order to get different information to display reports and graphs and with the *Event Generator* if any specification of the network is changed.

For an example please see section ‎6.3.

## 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.

Since data from all parts of the system, concentrate here, the *Log* is the idle place to process the data and produce more meaningful numbers, like the different throughput calculations, and data messages failure rate.

Beside the regular data reports the *Log* will also hold Error reports, that contains information about system health, and also reports about data messages that have failed to reach their destination (points to protocol failure).

The main type of data the *Log* will collect is information about different messages that will spark from the different *OLSRv2 Layers* (Nodes). Each control message (*"Hello"* or *"TC"* message) as well as each data message that is being sent (passed to the *Tasks Queue*) will be logged for later processing.

## Tasks Queue

Since the simulator main concept is that it is an Event driven simulator, we need to support easy handling of Events. Events are basically tasks that are stamped with a few special attributes, like destination and time of execution, and are designated for different object throughout the system.

Because all Tasks need to be processed in a chronological order, a simple priority queue is used to sort the tasks. From one end tasks can be pushed into the queue by the *Event Generator* or by the different *OLSRv2 Layers*, and at the other end tasks are being popped by the *Dispatcher* and distributed according to their type.

## 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 curtain node, and is designated to another node in the network.

In order to safely create these events, the *Event Generator* needs to consult with the *Topology Manager*. For example, upon creating a new station the *Event Generator* should check that a station does not already exists, with the same parameters (id, location). Another example is when creating *Data event*, the *Event Generator* should check with the *Topology Manager* if the source or target stations exists.

Since each inquiry from the *Topology Manager* is like looking at a network layout from the past (the *Tasks Queue* may still hold events) the *Event Generator* must hold a reference to all of the *Topology events* that are still in the *Tasks Queue* as these events hold the missing information required to update the topology picture that will exist when the currently created event will eventually be handled by the *Dispatcher.* For example: At T0 node “X” exist at the *Topology Manager*. A *Topology event* is created by the *Event Generator* that instructs the *Topology Manager* to destroy node “X” at T1 () and is pushed by it to the *Tasks Queue*. Immediately afterwards the *Event Generator* wants to create a new *Data message* that will be directed by the *Dispatcher* to node “X” at T2 (). The *Event Generator* inquire the *Topology Manager* about the state of node “X” and sees that it existed at T0. If the *Event Generator* would have the *Data message* based on that information, the *Dispatcher* would have discarded it at T2, since it’s no longer exists at T2 (was destroyed at T1). The *Dispatcher* would have registered an un-justified error notice (not related to OLSRv2 performances) and we’ve lost an event at T2. However, since the *Event Generator* has a reference to the event which is still at the *Tasks Queue* it knows the node will no longer exist at T2 so it must change the event characteristics (event’s target node, or event’s essence).

For more information about the specific algorithms that are used to handle these events, see section ‎5.1.

## Dispatcher

The *Dispatcher*’s name basically reveals its role. It’s designed to retrieve the top task from the *Tasks Queue* (The next task in a chronological order), quickly 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 nodes of the messages.
* *“Hello”* or *“TC” messages* – are passed to all of the *OLSRv2 Layers* that represents nodes which are within the *Reception Radius* of the originator node location. The list of nodes which answer this criterion is given to the *Dispatcher* by requesting the *Topology Manager* which has the knowledge regarding the nodes physical whereabouts in space.

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

## Topology Manager

The *Topology Manager* Job is to maintain information about the simulated physical attributes of nodes. The combined data of all the nodes attributes defines the network physical picture at a curtain time. The attributes that will be stored per node within 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 curtain node to a new location (considering that the new location is still within the defined space boundaries)
* *Reception Radius –* each node might have a different reception radius, simulating changes that might be induced because of different platforms or terrain conditions that real world station are expose to.

The *Topology Manager* may be asked by the *Dispatcher* to supply a list of nodes which *Coordinates* are within a curtain node’s *Reception Radius*. That way the *Dispatcher* can decide which *OLSRv2 Layers* needs to be on the receiver end of a curtain *“Hello”* or *“TC” message* task.

## OLSRv2 Layer

This module implements the OLSRv2 logic of a single node it a MANET environment. This module is split into two sub-layers:

* *NHDP Layer* – implements the NHDP logic. Is responsible for some of the *“Hello” messages* tasks processing, and the invocation of all them.
* *OLSR Layer* – implements the OLSRv2 logic. Is responsible for adding additional information to the *“Hello” messages* (such as MPR information) as detailed in the *OLSR* description (Section ‎1.3), as well as generating and processing *“TC” messages* and *Data messages*.

Passing messages to nodes in the *OLSRv2 Layer* node *Reception Radius* is done, by pushing the relevant events to the *Tasks Queue* with information about the message source. The *Dispatcher* in its turn will redirect that message to the appropriate *OLSRv2 Layers*.

Messages events which are produces as a byproduct of an earlier event will be tagged with a timestamp of the previous event’s timestamp in addition to a small delta which represent the network’s propagation delay as well as the nodes computation time.

*OLSRv2 Layer* objects are created/destroyed by the *Dispatcher* upon receiving a *Topology event* which corresponds to that action.

The *OLSRv2 Layer* alerts the *Log* about new messages that will be pushed to the *Tasks Queue* for later throughput calculations, as well as errors that might occur because of a protocol failure. E.g. when a *Data message* task is received which the destination is unknown to the *OLSRv2 Layer* (may have not been updated yet, with a new node’s appearance).

# Main Algorithm

## Event Generation

Event Generation and Manipulation algorithm handles the generation of the main events in the systems according to the actions received from the GUI model. This algorithm is placed in the Event Generator model (see 4.4) and Tasks Queue (see 4.3).

In general, when the user makes an action in the GUI, for example add a new node or send massage, the Event Generator will create a formatted event according to the type and information that it receives from the GUI and enters the event to the Tasks Queue.

**GenerateEvent:**

1. Each EVENT\_GEN\_TIME
   1. Create event according to current schema.
   2. Get information from Topology Manager for the event.
   3. Check if the event is consistent with the current topology.
   4. If (yes) insert event into Tasks Queue.
   5. Otherwise, goto a.

## Protocol implementation (per node)[Eli]

Each node will receive a massage from the Dispatcher indicating that some event regarding this node has occurred in the system. This can be event triggered by user as well as an internal event such as Hello or TC massage.

**HelloMsgProcessing(msg): (NHDP layer)**

1. If (msg is invalid) disregard;
2. Get neighbor list.
3. If (no such neighbor as msgsrc)
   1. Add to Neighbor set
   2. Create Event VALID\_TIME\_EXPIRES in curr\_time+valid\_time.
4. Get Lost Neighbor list.
5. For each (msgsrc address that is lost)
   1. If (address not in Lost Neighbor list)
      1. Add new entry to list
6. Get Link Set
7. For each (new msgsrc address)
   1. Add new entry to set
   2. Create Event VALID\_TIME\_EXPIRES in curr\_time+valid\_time.
8. For each (existing mgsrc address)
   1. Update valid time
   2. Create Event VALID\_TIME\_EXPIRES in curr\_time+valid\_time.
   3. If (msglink\_state == symmetric) update the nodes entry in Link Set to symmetric.
9. Get 2-hop set
10. For each (neighbor of the msgsrc)
    1. If (no entry in 2-hop set exist)
       1. Add new entry.
       2. Create Event VALID\_TIME\_EXPIRES in curr\_time+valid\_time.
    2. Otherwise, update the existing info accordingly.
11. **OLSRv2HelloMsgProcessing(msg).**

**OLSRv2HelloMsgProcessing(msg): (OLSRv2)**

1. If the massage is invalid discard.
2. If any local address are marked as MPR in msgneighbors set.
   1. Mark msgsrc in Neighbor set MPR\_SELECTOR = true.
3. Otherwise, Mark msgsrc in Neighbor set MPR\_SELECTOR = false.
4. If new node with link\_type=SYMETRIC is add or removed, or node is lost, or 2-hop node is added or removed
   1. **CalculateMPRSet** () //Recalculate the MPR group.
5. Update Advertised Neighbor set.
6. Generate TC massage.
7. Enter TC massage event.

**TCMsgProcessing(msg):**

1. If the massage is invalid discard.
2. If this massage has already been processed or forwarded discard.
3. Get Advertised Remote router set
4. If (msgsrc not in set)
   1. Add new entry to set
   2. Create Event VALID\_TIME\_EXPIRES in curr\_time+valid\_time.
5. Otherwise ,
   1. Update valid time
   2. Create Event VALID\_TIME\_EXPIRES in curr\_time+valid\_time.
6. Get Topology set
7. If (msgsrc not in set)
   1. Add new entry to set
   2. Create Event VALID\_TIME\_EXPIRES in curr\_time+valid\_time.
8. Otherwise ,
   1. Update valid time
   2. Create Event VALID\_TIME\_EXPIRES in curr\_time+valid\_time.
9. For each x :
   1. CalculateRoute(curr\_node, x);

**GenerateMasseges():**

1. Each EVENT\_GEN\_TIME
   1. Generate Hello massage.
   2. Generate TC massage.

**DataSend(msg, dest):**

1. Get Routing set.
2. Find tuple with Destanation\_addr = dest
3. Create Data Massage Event with from= this\_node, to = next\_hop, destination = dest.
4. Enter Event to Tasks Queue.

**CalculateMPRSet():**

**CalculateRoute(x,y):**

Implantation of BFS algorithm to find the shortest path between x and y.

## Dispatcher [Asi]

1. While queue isn't empty
2. Pull next event from queue
3. If (event.type == NEW\_STATION)
4. Add new station to Topology Manager
5. If (event.type == STATION\_MOVED)
6. Update Topology Manager with new station coordination
7. If (event.type == STATION\_REMOVED)
8. Remove station from Topology Manager
9. If (event.type == NEW\_DATA\_PACKET)
10. Execute method send\_data\_message(station target) in station source
11. If (event.type == SEND\_HELLO\_MESSAGE)
12. Get station's neighbors from Topology Manager
13. For each neighbor do:
14. Execute received\_hello\_message(station source, time)
15. If (event.type == SEND\_TC\_MESSAGE)
16. Get station's neighbors from Topology Manager
17. For each neighbor do:
18. Execute received\_tc\_message(station source, time)
19. If (event.type == CHECK\_STATION\_VALIDITY)
20. Execute check\_station\_valid(station target) in station source

## Topology Manager [Asi]

## Data collection Throughput Analysis (Logger)[Asi]

# Appendixes

## Appendix 1: Estimated project timetable

02.11.2009 – First Report Submission

05.11.2009 – First Report Feedback

13.12.2009 – Project Review

15.1.2010 - Project Presentation and Final Report Submission

## Appendix 2: Requirements and Assumptions [Assaf]

### 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 project team will try it best to accommodate the protocols attributes and description as explained in these documents, however modification may be necessary because of time and complexity constrains.
* The project team will produce more/different features/data as asked by the Project Guidance crew.
* As mention in the *Project Description* section (‎2), eventually some goals may not be implemented because of time issue constrains. The Project team will, of course, try it best to accommodate all of the described features and Goals.

### Requirements

* The simulator will be implemented in Java, so JVM is needed for execution.

## Appendix 3: Screen shots [Asi]