Abstract
MANET has been a widely discussed network type in the last decade, and there have been many proposals for MANET routing protocols. This thesis studies and compares several proposals and focuses mainly on the proposals of extending the existing OSPF protocol for MANET. Two of the OSPF wireless extension proposals OSPF-MDR and OSPF-MPR are presented and compared.

OSPF-MDR is implemented in the network simulation environment J-Sim. The implementation is an extension and modification of an existing OSPF implementation in J-Sim. A few multi-hop wireless network scenarios are simulated and analyzed using the OSPF-MDR routing protocol.
Preface

This thesis was written as part of my master degree in Computer Science at the University of Oslo, Department of Informatics. The thesis work was done at UNIK – University Graduation Centre.

Through the thesis writing and programming, I gain a deeper understanding of OSPF and its wireless extension and other wireless routing protocols for MANETs. It is a process of learning the J-Sim simulation tool, understanding the existing OSPF implementation and implementing the extension. The J-Sim environment is powerful but complicated, and the learning curve was quite flat at the beginning though it took off at the end. A lot of time has been spent on reviewing the existing OSPF implementation and finding out what is missing for further wireless extension. The implementation of the wireless extension has been a struggling of the algorithms and the balancing of the design and effects.

I would like to thank Professor Knut Øvsthus for good advice and guidance, and Dr. Reinert Korsnes for his precious help on understanding of MDR Selection algorithm, and the users of the j-sim-users mailing list for their help on J-Sim, and my family for their patience and understanding.

Ling Shi
Oslo, November 15th 2006
# Contents

1 Introduction ................................................................................................................. 1  
1.1 Thesis organization .................................................................................................. 1  
2 Routing Protocols for Mobile AD-HOC Networks ........................................................... 2  
2.1 Introduction to MANETs and its routing protocols ..................................................... 2  
2.2 MANETs-only routing protocols for MANETs ........................................................... 2  
2.2.1 Ad-hoc On-demand Distance Vector ................................................................... 2  
2.2.2 Optimized Link State Routing Protocol ............................................................... 5  
2.3 Wireless extension to OSPF ..................................................................................... 8  
2.3.1 OSPF features related to wireless extensions ....................................................... 8  
2.3.2 OSPF MPR Extension for Ad Hoc Networks [OSPFMPR] ........................................ 9  
2.3.3 MANET Extension of OSPF using CDS Flooding [OSPFMDR] ............................ 10  
2.3.4 Comparing wireless extensions to OSPF ............................................................ 11  
3 Tools and methods .................................................................................................... 13  
3.1 Used tools .............................................................................................................. 13  
3.2 J-Sim .................................................................................................................... 13  
3.2.1 Introduction to J-sim ............................................................................................ 13  
3.2.2 J-Sim Standard Packages ................................................................................. 13  
3.2.3 Core Service Layer in INET ............................................................................. 14  
3.2.4 The wireless extension of J-Sim ......................................................................... 16  
4 The OSPF-MDR extension .......................................................................................... 18  
4.1 New MANET interface ............................................................................................ 18  
4.1.1 Modified data items ............................................................................................ 18  
4.1.2 New data items ................................................................................................. 18  
4.1.3 New Configurable Interface Parameters ............................................................ 19  
4.2 Changes to Neighbor Data Structure ..................................................................... 19  
4.3 Hello protocol ....................................................................................................... 19  
4.3.1 Hello data structure and Sending Hello Packets .............................................. 19  
4.3.2 Receiving Hello Packets .................................................................................. 20  
4.4 MDR selection algorithm ....................................................................................... 22  
4.4.1 Phase 1: Neighbor Connectivity Matrix (NCM) creation ................................ 22  
4.4.2 Phase 2: MDR selection ................................................................................... 24  
4.4.3 Phase 3: Backup MDR selection ....................................................................... 28  
4.4.4 Phase 4: (Backup) MDR parent selection ......................................................... 30  
4.5 Changes to Interface state machine ...................................................................... 32  
4.6 Adjacency maintenance ......................................................................................... 34  
4.7 Flooding procedure ............................................................................................... 36  
4.7.1 Flooding LSAs ................................................................................................... 36  
4.7.2 Retransmitting LSAs ......................................................................................... 36  
4.7.3 Sending the LS ACKs ....................................................................................... 36  
4.7.4 Receiving the LS ACKs ................................................................................... 36  
4.8 Changes to LSAs ................................................................................................... 37  
4.9 Routing table calculation ....................................................................................... 37  
5 The OSPF-MDR implementation in J-Sim ................................................................ 38  
5.1 The current OSPF implementation in J-Sim .............................................................. 38  
5.1.1 OSPF_Interface ............................................................................................... 38  
5.1.2 OSPF_Neighbor .............................................................................................. 38  
5.1.3 OSPF_Hello .................................................................................................... 38  
5.1.4 OSPF class ...................................................................................................... 39  
5.1.5 Summary of the existing OSPF implementation .............................................. 41
5.2 Implementing the OSPF-MDR extension in J-Sim.................................................. 41
5.2.1 Implementing a new MANET interface............................................................... 41
5.2.2 Changes to the neighbor data structure .............................................................. 43
5.2.3 MANET Hello....................................................................................................... 43
5.2.4 Interface State Machine....................................................................................... 48
5.2.5 MDR Selection....................................................................................................... 48
5.2.6 Adjacency Maintenance in OSPF-MDR.............................................................. 50
5.2.7 Router_LSA ......................................................................................................... 50
5.2.8 Flooding procedure.............................................................................................. 51
5.2.9 Routing table calculation..................................................................................... 51
5.2.10 Sending and broadcasting of the packets in OSPF-MDR.................................... 51

6 Simulation and Scenarios ......................................................................................... 52
6.1 A Four-Node’s scenario ......................................................................................... 52
6.1.1 Wireless network scenario and TCL scripts........................................................ 52
6.1.2 The simulation result for Four-Node’s scenario .................................................. 55
6.2 A Six-Node’s scenario ........................................................................................... 57
6.2.1 Simulation results for a six-node’s scenario ....................................................... 57
6.2.2 MDR Selection results of the Six-Node’s scenario ............................................. 59

7 Conclusions ............................................................................................................... 62
7.1 Thesis summary....................................................................................................... 62
7.2 Future works.......................................................................................................... 62

List of figures
Figure 1: AODV - broadcasting RREQ ........................................................................... 3
Figure 2: AODV – sending RREP back to the source ..................................................... 3
Figure 3: AODV – route established............................................................................... 4
Figure 4: AODV – link breakage and RERR................................................................ 4
Figure 5: AODV – re-establishing the route after link breakage ................................... 5
Figure 6: Pure flooding [OLSR]..................................................................................... 6
Figure 7: MPR flooding [OLSR].................................................................................... 6
Figure 8: Wireless topology example for MPR calculation............................................. 7
Figure 9: Result of MPR calculation on a wireless example.......................................... 8
Figure 10: OSPF-MPR is an extention to OSPF inspired by MPR.................................. 10
Figure 11: The class pyramid in J-Sim, adopted from [JSIMNS]...................................... 14
Figure 12: The CSL Service/Event port, adopted from [JSIMINET]............................... 15
Figure 13: The composable CSL, adopted from [JSIMINET]........................................ 15
Figure 14: Schematic of a wireless node in J-Sim, adopted from [JSIMWIRELESS]...... 17
Figure 15: NCM calculation on a wireless example...................................................... 24
Figure 16: MDR selection on a wireless example......................................................... 25
Figure 17: Backup MDR selection on a wireless example.............................................. 30
Figure 18: MDR Parent selection in MDR selection on a wireless example.................... 32
Figure 19: Events that cause MDRNeighborChange................................................... 33
Figure 20: Changes to Interface State Machine for a MANET interface......................... 34
Figure 21: Changes to Neighbor State Machine for a MANET interface......................... 35
Figure 22: Class diagram OSPF_Interface, OSPF_Neighbor and OSPF_Hello in the OSPF
implementation.................................................................................................................. 40
Figure 23: Class diagram OSPF_LLS and OSPF_LLS_TLV in the OSPF-MDR
implementation.................................................................................................................. 44
Figure 24: Forming the OSPF Hello Packet in OSPF-MDR........................................... 46
Figure 25: Processing the OSPF Hello Packet in OSPF-MDR........................................ 47
Figure 26: A four nodes’ scenario

Figure 27: The object overview for the simulation and the composed CLS for the wireless node

Figure 28: The throughput for the four nodes’ scenario, FIFO queue, queue capacity=40, packetsize=1500, packetrate=0.01

Figure 29: The throughput for the four nodes’ scenario, PreemptPriorityQueue, queue capacity=40, packetsize=1500, packetrate=0.01

Figure 30: A six nodes’ scenario

Figure 31: Simulation result for the six node’s scenario, PPQ, queue capacity=40

Figure 32: Simulation result for the six node’s scenario, PPQ, queue capacity=40

Figure 33: Result of MDRNeighborChange event on neighbor node 0 when node 4 changes state to twoway

Figure 34: Result of MDRNeighborChange event on node 0 when neighbor node 1 changes state, mdr level and RNL list

List of tables
Table 1: CDS and bi-connected CDS in OSPF and OSPF-MDR
Table 2: Comparison between OSPF-MDR and OSPF-MPR
Table 3: CSL services and sub components
Table 4: LSA Fullness

Appendix
A. TCL simulation code for two wireless scenarios
B. Installation of simulation tools in a windows environment
C. The walk-through process of making extension to OSPF implementation
D. Javadoc for OSPF package of the OSPF-MDR implementation
1 Introduction

Mobile Ad-hoc Networks (MANETs) are wireless network with mobile nodes. Mobile means the nodes are free to move themselves randomly, which may cause the rapid and unpredictable change of the network topology. Ad-hoc means that each node is willing to forward data for other nodes. MANETs can be used in a wide range of situations, like moving vehicles and exhorsted area without wired networks. A well known scenario is in the battle field without any network infrastructures.

It has been designed many routing protocols to meet the requirements for MANET since 1990s, for example, Ad-hoc On-demand Distance Vector [AODV], Optimized Link State Routing Protocol [OLSR]. There has been some robustness discussion around these protocols and extending OSPF for wireless network is presented as an alternative. OSPF is a well developed and implemented routing protocol for wired networks around the world. The main reasons of extending OSPF are to reuse the existing structure around robustness like adjacency forming, and also to use one and the same system for both wired and wireless routing and thereby to reduce the transit-overhead between a wired and a wireless system.

There exist different researches around wireless extension for OSPF as described later in 2.3. In this thesis I will implement internet draft [OSPFMDR].

It is normal to implement the wireless extension in a simulation environment before running the protocol in a real network. Though the latter may be optimal, running in a real network is more time-consuming and budget-dependent; therefore I chose to do the implementation in J-Sim. J-Sim is a java based simulation tool for networking programming which provides a rich library for coding and testing.

1.1 Thesis organization

The second chapter will simply introduce four of the main routing protocols for MANET. The third chapter will introduce J-Sim as an API and its routing protocol related packages. The fourth chapter will focus on the wireless extension OSPF-MDR protocol and its main modifications to the existing OSPF, where MDR is an abbreviation for MANET Designated Router. The fifth chapter will present the implementation of OSPF-MDR for J-Sim. The sixth chapter will show the simulation and scenarios, and its comparison to OLSR results. The seventh chapter will make the conclusion and point the future work.
This chapter first introduces the MANETs and its routing protocols, and then describes and compares two of the representative existing MANETs-only routing protocols: AODV and OLSR, and finally presents and compared two proposals of wireless extension to OSPF: OSPF-MPR and OSPF-MDR.

2.1 Introduction to MANETs and its routing protocols

The characteristics of MANETs are described in [MANET] as 1) dynamic topologies 2) bandwidth-constrained, variable capacity links 3) energy-constrained operation 4) limited physical security. No existing wired routing protocols can effectively handle this type of network; therefore new routing protocols are necessary to meet the requirements for MANETs.

Among the existing routing protocols for MANETs, most of them are designed for MANETs use only and do not support the wired networks, which are abbreviated as MANETs-only routing protocols here after in the thesis. However, there has been a reverse trend in the last few years, i.e. the wireless extensions to the existing OSPF protocol.

MANET was initially defined either as a separate network or a stub network connected to the wired network. Due to the totally different characteristics of MANETs, a MANETs-only protocol fits naturally for the purpose. In the last few years, the MANETs get more and more integrated with the wired network, overhead between wireless and wired networks is large when they use different routing protocols. Other common weakness of MANETs-only routing protocols is lacking of robustness, for example, they do not provide backup routers to take over the routing when link to one of the routers is broken. In OSPF, the set of MDRs forms a connected dominating set (CDS), and the set of MDRs and BMDRs forms a biconnected dominating set, and the Backup DRs provide robustness through biconnected redundancy [ADOSPFMDR]. Extending the existing OSPF will of purpose both reduce the overhead and keep the robustness nature of OSPF.

2.2 MANETs-only routing protocols for MANETs

One way to categorize the protocols is reactive and proactive. AODV and OLSR are two of the most representative routing protocols for MANETs in these two categories.

2.2.1 Ad-hoc On-demand Distance Vector

AODV is a reactive routing protocol for wireless network. Reactive means it only establish a route to the destination on demand, therefore the route discovery traffic is minimized. However the route establishing process is normally long and resource consuming, and the reaction for link break or topology change is normally slow for reactive routing. AODV makes great efforts to reduce the route establishing time and to reduce the response time for link breakages changes in topology.

Route Requests (RREqs), Route Replies (RREPs), and Route Errors (RERRs) are the message types defined by AODV, and they play the central roles in the route establishing and route maintenance processes.
A simple AODV route establishing scenario is presented here. A source node ‘S’ starts the route establishing process by broadcasting a message to other AODV nodes referred to as RREQ when it needs a root to a destination ‘D’ and the route is not available. When the RREQ arrives at a neighbor node, the neighbor will generate a RREP if it itself is the destination or it has an active route to the destination; otherwise, it will update the route table by the reverse route etc. and then continue to broadcast the RREQ.

Figure 1: AODV - broadcasting RREQ

The RREP will be sent back from the destination node to the source node, using the reverse route defined in the routing table, as shown in figure 2.

Figure 2: AODV – sending RREP back to the source

Finally the route is established when the RREP arrives at the source node. The route is again a reverse of the RREP route.
AODV allows mobile nodes to respond to link breakages changes in network topology in a timely manner [AODV]. Figure 4 shows a scenario where node E moves and the link between C and E is broken. Node C will send a RERR message to its upstream neighbors, and then RERR is propagated until it reaches the source node. The source node found no route available, and then starts a new RREQ to discover a route, shown in figure 5.
AODV uses unique sequence numbers to avoid loop and also to simplify the implementation. Timeout of inactive routes minimize the maintenance in the routing table. Though so many enhancements, it still faces the bottleneck of scalability in some network scenarios with large amount of nodes as shown in [AODVSCA].

2.2.2 Optimized Link State Routing Protocol

OLSR is a proactive routing protocol for wireless network. Proactive means the protocol maintains the total routing table regularly. With routing table available, the time to establish the communication is short compared to reactive protocols. However, the topology control traffic and routing table calculation are heavy because of the continuous routing discovery. OLSR focuses on minimizing the traffic burden as described below.

OLSR uses Hello messages for neighbor sensing, and the messages also help to build up the 1-hop and 2-hops neighbor tables. Those neighbor tables are then used to calculate the MPRs initialized by changes in the neighborhood. A node’s MPRs are a subset of its symmetric one-hop neighbors that cover all the node's two-hop neighbors during flooding of broadcasting information [OLSR].

MPRs have multifunctions in the OLSR. For the first, it reduces the flooding traffic of Topology control messages. Topology control traffic is flooded periodically to detect the changes of the topology so as to update the routing table continuously. By only allowing MPRs to forward broadcastings, OLSR reduces the TC traffic tremendously. The following two figures show clearly the difference between pure flooding and MPR flooding, where pure flooding requires much more retransmissions to diffuse a message up to 3-hop. The larger and denser network becomes the more reduction of retransmissions it will achieve by using MPR flooding. Therefore OLSR is suitable for large and dense networks.
In addition to reducing flooding traffic, MPRs are also selected to declare the link state traffic, and are selected to form the route from a node to any destination in the network during route calculation. The multi-roles of MPRs make them especially interesting in the later discussion in 2.3.2.

There exist several algorithms for MPR selection. The algorithm shown here is a heuristics developed by [MPR]. Suppose the calculating node is $x$, $N(x)$ is its one-hop nodes set, $N_2(x)$ is its two-hop neighbors set and $MPR(x)$ is its multipoint relays set.

1. Start with an empty multipoint relay set $MPR(x)$
2. First select those one-hop neighbor nodes in $N(x)$ as the multipoint relays which are the only neighbor of some node in $N_2(x)$, and add these one-hop neighbor nodes to the multipoint relay set $MPR(x)$
3. While there still exist some node in $N_2(x)$ which is not covered by the multipoint relay set $MPR(x)$:
   (a) For each node in $N(x)$ which is not in $MPR(x)$, compute the number of nodes that it covers among the uncovered nodes in the set $N_2(x)$
   (b) Add that node of $N(x)$ in $MPR(x)$ for which this number is the maximum.
The calculation is explained in the following 5 nodes’ example. One-hop and two-hop neighbors are shown in a table by the side of every node.

For Node n0, we start with an empty multipoint relay set MPR(n0). 1-hop neighbors of n0 are n1 and n2, and 2-hop neighbors of n0 are n3 and n4. Since both of the 2-hop neighbors have only one 1-hop neighbor as their neighbors, these 1-hop neighbors (n1, n2) are added to the MPR(n0) set. MPR(n0)=(n1,n2).

For Node n1, we start with an empty multipoint relay set MPR(n1). 1-hop neighbors of n1 are n0, n1 and n3; and n4 is the only 2-hop neighbor. The only 2-hop neighbor n4 has more than one 1-hop neighbors as its neighbors, therefore, no 1-hop neighbors are added to the MPR(n1) set. We then count the number of uncovered 2-hop neighbors each non-MPR 1-hop neighbor can cover. n0 covers 0, n2 covers 1 and n3 covers 1. We choose n2 as the maximum and add it to the MPR(n1) set. Node n3 can replace n2 here because n2 and n3 covers the same amount of 2-hop neighbors. MPR(n1)=n2.

For Node n2, we start with an empty multipoint relay set MPR(n2). 1-hop neighbors of n2 are n0, n1 and n4, and the 2-hop neighbor of n2 is n3. MPR(n2) is calculated after the same procedures as n1. MPR(n2)=n1.

For Node n3, we start with an empty multipoint relay set MPR(n3). 1-hop neighbors of n3 are n1 and n4, and 2-hop neighbors of n3 are n0 and n2. 2-hop neighbor n0 has only one 1-hop neighbor (n1) as its neighbor, therefore n1 is added to the MPR(n3) set. Since the other 2-hop neighbour n2 is also covered by n1, no further calculation is needed. MPR(n3)=n1.

For Node n4, we start with an empty multipoint relay set MPR(n4). 1-hop neighbors of n4 are n2 and n3, and 2-hop neighbors of n4 are n0 and n1. 2-hop neighbor n0 has only one 1-hop neighbor...
neighbor (n2) as its neighbor, therefore n2 is added to the MPR(n4) set. Since the other 2-hop neighbour n1 is also covered by n2, no further calculation is needed. MPR(n4)=n2.

The result of the calculation can be shown on the following figure. Node n1 and n2 are chosen as MPR by the other nodes in the topology. The MPR Selector entry shows the other nodes that have chosen the node as MPR. The MPR Selector entry is none if the node itself is not an MPR.

**Figure 9: Result of MPR calculation on a wireless example**

### 2.3 Wireless extension to OSPF

This section first reviews the wireless extension related OSPF features and then introduces and compares two of the proposals for OSPF wireless extensions: OSPF-MPR and OSPF-MDR. The wireless extensions concentrate on minimizing the flooding traffic like those MANETs-only protocols, while they also focus on adjacency forming and robustness.

#### 2.3.1 OSPF features related to wireless extensions

The wireless extensions are built upon the OSPF v2 [RFC2328] or OSPF v3. A review and understanding of OSPF is a pre requisite for understanding the wireless extensions. Here are some OSPF features that are related to the wireless extension, and most of them will be modified in the later implementations.
2.3.1.1 OSPF interface
OSPF has four interface types: point-to-point, broadcast (e.g. Ethernet), non-broadcast multi-access (NBMA) and point-to-multipoint. Among them, the point-to-multipoint type and broadcast type are most related.

Point-to-MultiPoint network is a mode of non-broadcast network; it treats the non-broadcast network as a collection of point-to-point links.

Broadcast networks support many (more than two) attached routers, together with the capability to address a single physical message to all of the attached routers (broadcast). Neighboring routers are discovered dynamically on these nets using OSPF's Hello Protocol. The Hello Protocol itself takes advantage of the broadcast capability. The OSPF protocol makes further use of multicast capabilities, if they exist. Each pair of routers on a broadcast network is assumed to be able to communicate directly.

Though MANET is a kind of point-to-multipoint network, but some main characteristics as the dynamic topology are not represented in any of the existing interface types, therefore none of the interface types can be used to gain satisfactory results.

2.3.1.2 OSPF Neighbor Data Structure
Each separate conversation between two neighboring routers is described by a neighbor data structure, which means multiple neighbor data structures for multiple network interfaces between two neighboring routers. This data structure contains information for forming/formed adjacencies.

2.3.1.3 OSPF Hello
The main tasks of OSPF Hello are to establish and maintain neighbor relationships. OSPF Hello is also used to dynamically discover neighboring routers on broadcast networks.

2.3.1.4 OSPF Adjacencies
Adjacency is a relationship formed between selected neighboring routers for the purpose of exchanging routing information. Not every pair of neighboring routers becomes adjacent.

Adjacencies are established with some subset of the router's neighbors. Routers connected by point-to-point networks, Point-to-MultiPoint networks and virtual links always become adjacent. On broadcast and NBMA networks, all routers become adjacent to both the Designated Router and the Backup Designated Router [RFC2328].

2.3.2 OSPF MPR Extension for Ad Hoc Networks [OSPFMPR]
OSPF-MPR extension is inspired by techniques from OLSR [OLSR] since a proactive protocol like OLSR is naturally for the protocol. A new wireless interface type is proposed in OSPF-MPR. This type will coexist with the legacy OSPF interface types, which means that routing on wired networks remains as before.

OSPF-MPR extension keeps OSPF’s feature like robust routing over adjacencies and shortest paths; while it also provides the solution for wireless flooding inspired by MPR, and other extensions on adjacency formation and acknowledging operations for MANETs, as shown on the figure below.
One of the main targets of wireless routing is to minimize retransmission of protocol packages and reduce adjacencies. Flooding traffic is reduced by only allowing MPRs to forward the LSUs updates on a wireless interface as in OLSR. MPR selection uses the same heuristics as in OLSR.

Though the MANETs are type of point-to-multipoint networks, ideas from broadcast networks are borrowed to reduce the flooding traffic and form the adjacencies. In broadcast networks, Designated Routers (DRs) and Backup DRs are selected among the routers to form adjacencies and reduce the flooding traffic. Similar to the OSPF operation in a broadcast network, a router in the MANET interface only forms adjacency with a subset of its wireless neighbors. However, no DR or Backup DR are elected in OSPF-MPR, therefore adjacencies beween DRs and BDRs are not applicable here. A node brings up adjacencies only with the neighbors it has included in its MPR set and its MPR Selector set though some nodes (called synch nodes) may bring up adjacencies with all their wireless neighbors to achieve shortest paths [OSPF-MDR]. MPR selection process in [OSPF-MDR] is similar to that described in 2.2.2.

In OLSR Traffic Control (TC) messages performs the topology declaration (advertisement of link state), but no acknowledgement is expected for TC messages as described in [OLSR]. Link State Advertisement (LSA) performs the similar task in OSPF, but acknowledgements (LSAcks) are mandatory. LSAcks are also mandatory in OSPF-MPR. In principle, either implicit or explicit LSA acknowledgement is expected from all the adjacent neighbors after a node has sent an LSA. If no acknowledge is received, the node retransmits the LSA. LSA acknowledgements are aggregated and multicasted to reduce the superfluous retransmissions.

### 2.3.3 MANET Extension of OSPF using CDS Flooding [OSPF-MDR]

MANET Extension of OSPF using CDS Flooding is also called OSPF-MDR. MDR is abbreviated for MANET Designated Router and is the core concept in the extension. OSPF reduces overhead in a broadcast network by electing a Designated Router (DR) and Backup DR, and by having two neighboring routers form an adjacency only if one of them is the DR or Backup DR [OSPF-MDR].

The DRs form a connected dominating set (CDS), while the DRs and Backup DRs form a bi-connected CDS in OSPF. CDS and bi-connected CDS ensure the robustness of OSPF. After the same principle, OSPF-MDR selects MDRs and Backup MDRs and MDRs form a CDS while MDRs and Backup MDRs form bi-connected CDS.
A new interface type is added under the condition that the legacy interface types will function as before. This interface type borrows many ideas from the broadcast interface type though MANET is a kind of point-to-multipoint network.

The OSPF-MDR hellos are used for neighbor discovery as the hellos in the broadcast interface do. In addition to establishing and maintaining neighbor relationships, the OSPF-MDR hellos are also used to exchange 2-hop neighbor information that is necessary for later MDR calculation.

MDR selection is a mechanism to decide if a router is MDR, Backup MDR or MDR other, so will it assign the (Backup) MDR Parent for all the nodes, so that there is a directed diagram from every node to the Rmax in the topology. Adjacencies are formed using MDRs and Backup MDRs in the network.

Interface State Machine is modified to handle the events like MDRNeighborChange. Neighbor State Machine manages the neighbor’s state change and allows adjacency forming only to neighbors with state 2-Way or higher.

LSAs are flooded from a (Backup) MDR router in the MANET network to advertise its neighbors. The amount of neighbors in the router-LSA is decided by a configurable parameter LSAFullness, for example LSAFullness=0 means minimum LSA and only adjacent neighbors are included, LSAFullness=4 means full LSA and all routable neighbors are included in addition to the adjacent neighbors, values from 1 to 3 means only part of the routable neighbors are included. Although the router-LSA may not be full, the routing table calculation includes all routable neighbors for SPF.

### 2.3.4 Comparing wireless extensions to OSPF

Both of the OSPF extensions add a new interface for MANET routing, and this interface coexists with the legacy interfaces. Both of them modify hellos to include LLS for exchanging extended neighbor information.

The main difference between the two proposals is MDR verses MPR selection and also the adjacency forming. The OSPF-MDR has a much more complicated selection algorithm than OSPF-MPR. MDR selection runs on the selecting router and it decides whether it will select itself as (Backup)MDR. MPR selection runs on the selecting router and it decides whether a subset of its neighbors should be MPR.

<table>
<thead>
<tr>
<th></th>
<th>OSPF-MDR</th>
<th>OSPF-MPR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DR</strong></td>
<td>MDR</td>
<td>MPR</td>
</tr>
<tr>
<td><strong>BDR</strong></td>
<td>Backup MDR</td>
<td>-</td>
</tr>
<tr>
<td><strong>Adjacency forming</strong></td>
<td>MDRs and Backup MDRs</td>
<td>MPRs and MPR Selectors</td>
</tr>
</tbody>
</table>

<p>| Table 1: CDS and bi-connected CDS in OSPF and OSPF-MDR |</p>
<table>
<thead>
<tr>
<th>Flooding</th>
<th>(Backup) MDRs</th>
<th>MPR</th>
</tr>
</thead>
</table>

Table 2: Comparison between OSPF-MDR and OSPF-MPR

This thesis chooses to implement the OSPF-MDR proposal because firstly it has the best scalability features [ADOSPFMDR]. Secondly we were advised to test this proposal by Joseph Macker, chair of the MANET group in IETF in his private communication with my supervisor.
3 Tools and methods

This chapter first summarizes the tools used in the implementation, and then introduces J-Sim network simulation environment, and its standard packages, Core Service Layer and wireless extension structure.

3.1 Used tools

The java based network simulation environment J-Sim version 1.3 is used for the implementation. The java version used is j2sdk1.4.2 from SUN. The “make tool” used is ant1.6.2 from Apache. Refer to the Appendix B for details of installation and configuration of the simulation environment.

3.2 J-Sim

This section first gives a brief presentation of J-Sim. Most of the materials are based on the tutorials and white papers from J-Sim’s website www.j-sim.org. J-Sim is a very comprehensive simulation tool, and this section focuses on its features related to INET and wireless network simulation.

3.2.1 Introduction to J-sim

J-Sim is a component based compositional simulation environment. It is an implementation of Autonomous Component Architecture (ACA) for simulation in programming language Java[JSIMACA], the communication between different components are ‘wired’ together via software ports.

Every network protocol object is implemented as a component. For example, the OSPF routing class is a component. All the components should override the following methods:

- process()
- reset()
- duplicate()
- info()
- start()/stop()/resume()

For example, the duplicate() method should take care of all the necessary data items, otherwise they will be missing during the transfer over the network. The start()/stop()/resume() methods only applies when the router is an active component and implements the drcl.comp.ActiveComponent.

J-Sim supports TCL scripting language for designing network scenarios. TCL scripts can easily create java objects, and call the methods in the objects. J-Sim also provides a background thread manager called runtime which is a key to the performance and all simulations are built around the runtime [JSIMACA].

3.2.2 J-Sim Standard Packages

J-Sim provides several standard packages for network simulation. The following figure shows the five-layer class organization in J-Sim.

- The ACA layer provides the basic component and port data structure;
- The NET layer contains network simulation primitives and tools;
- The INET layer contains internetworking components
- Specific network architectures are further derived from the INET layer
• The fifth layer are protocol modules and algorithms for the specific network architecture

The OSPF routing protocol lies among the Internet Module on the fifth layer, so do other INET routing protocols and wireless routing protocols as OLSR and AODV.

**Figure 11: The class pyramid in J-Sim, adopted from [JSIMNS]**

**Internetworking Simulation Platform**

Internetworking Simulation Platform (INET) is a generalized packet switched network model, which is composed of basic, abstract components extracted from the Internet [JSIMNS]. It is provided as an own package in J-Sim. Although INET is derived from internet, it’s general enough to be used for other networking architectures, including wireless networking architecture [JSIMNS].

### 3.2.3 Core Service Layer in INET

A node is a composite component which consists of applications, protocol modules, and a core service layer (CSL). The core service layer is an abstract component which encapsulates the functions of the network layer and the layers beneath the network layer. It provides network services and events to protocols, in the form of inter-component contracts. [JSIMINET]

The following picture shows the INET CSL and its service/event ports.
The sub components provides different services and INET has implemented them as shown in the table below.

<table>
<thead>
<tr>
<th>Services</th>
<th>INET implementation</th>
</tr>
</thead>
</table>

Figure 12: The CSL Service/Event port, adopted from [JSIMINET]

Figure 13: The composable CSL, adopted from [JSIMINET]

The Core Service Layer is however composable, and it can be divided into sub components as shown in the figure below.
<table>
<thead>
<tr>
<th></th>
<th>the data forwarding/delivery service</th>
<th>PacketDispatcher</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>the identity service</td>
<td>Identity</td>
</tr>
<tr>
<td>3</td>
<td>the routing service</td>
<td>RoutingTable</td>
</tr>
<tr>
<td>4</td>
<td>the interface/neighbor service</td>
<td>Hello</td>
</tr>
<tr>
<td>5</td>
<td>the packet filter configuration service</td>
<td>PacketFilter</td>
</tr>
</tbody>
</table>

Table 3: CSL services and sub components

This composable CSL will be used in most of the wireless node compositions because for example the Hello packet may need extended data items and so on. The compact CSL provides standard services and may be enough for legacy routing protocol like OSPF. The composition of CSL/node happens in the TCL scripts when creating the nodes.

3.2.4 The wireless extension of J-Sim

The wireless extension of J-Sim is a collection of components as shown in the figure [JSIMWIRELESS] below. It involves several levels of components.

- The lowest level is the Channel and NodePosition Tracker, and they simulate the wireless medium for simulation.
- The next level simulates the physical characteristics of the mobile nodes, for example, the position, movements, energy consumption and signal strength. This level includes WirelessPhy, MobilityModel, RadioPropagationModel and EnergyModel components.
- The third level provides the MAC link layer functions, and includes ARP, IEEE802.11 and LL, bundled in the drcl.net.mac package. The last level is the wireless routing protocol on the transport layer.

The routing protocols are collected in the drcl.inet.protocol package. Wireless protocols as AODV and OLSR have already been implemented in J-Sim, and the packages are called drcl.inet.protocol.aodv and drcl.inet.protocol.olsr. The new OSPF-MDR routing protocol will be allocated in the same package as OSPF, i.e. drcl.inet.protocol.ospf.

Almost all the objects are collected in the drcl.inet.mac package except the PktDispatcher and Hello from drcl.inet.core and the wireless routing protocol from drcl.inet.protocol. The PktDispatcher is connected with the routing protocol and the MAC Link Layer to send or receive packets. The routing protocol can also communicate with the MAC_802_11 to receive the linkbroken events.
Figure 14: Schematic of a wireless node in J-Sim, adopted from [JSIMWIRELESS]
4 The OSPF-MDR extension

This chapter will present the OSPF-MDR extension in details by going through the main extensions and modifications, as shown by the following list:

- New MANET interface type
- Changes to Neighbor Data Structure
- Changes to Hello protocol
- MDR and Backup MDR selection
- Changes to Interface state machine
- Adjacency forming
- Flooding procedure
- Changes to LSAs
- Routing table calculation

This section only discusses part of the draft, please consult [ref til deraft] for any detailed information. (eller noe slikt poenget er å gi en referanse her slik at du slipper å referere gjennom hele teksten under)

4.1 New MANET interface

This new interface type supports MANET type networks and its data structure includes some modified or new data items, and new configurable interface parameters. Most of these data items are necessary for MDR related tasks.

4.1.1 Modified data items

The data item “type” defines the type of the interface, and it has four predefined OSPF types (point-to-point, broadcast, point-to-multipoint, NBMA) as defined in [RFC2328]. Adding a new MANET type to the predefined value will allow the routing protocol identify the MANET network and then apply MANET routing on it.

The data item “state” defines the state of the interface, and it also exists in legancy OSPF. For a MANET interface type, the DR state implies an MDR state and the Backup DR state implies a Backup MDR state.

4.1.2 New data items

Several new data items are related to (Backup) MDR selection algorithm described later in 4.4.

- MDR Level
  MDR Level has three predefined values: 2 for MDR, 1 for backup MDR and 0 for MDR other.

- MDR Parent
  Each non-MDR selects an MDR Parent. It will be a neighboring MDR if it exists.

- Backup MDR Parent
  When the bi-connected adjacencies option is chosen (ref. parameter AdjConnectivity), each non-MDR selects a Backup MDR parent. It will also be a neighboring MDR/BMDR if it exists.

- Router Priority
Router Priority is a weigh variable to decide the willingness for a router to be MDR. This variable can be dynamically changed based on different criteria, like willingness and battery capacity etc.

- **Hello Sequence Number (HSN)**
  HSN is a sequence number carried by Hello Sequence TLV.

- **Lost Neighbor List (LNL)**
  LNL is a list of the Router IDs of neighbors that recently turned to status Down, included in the Lost Neighbor List TLV of Hello Packets.

### 4.1.3 New Configurable Interface Parameters

The new configurable interface parameters include 2HopRefresh, HelloRepeatCount, AckInterval, BackupWaitInterval, AdjConnectivity, MDRConstraint, LSAFullness. These parameters will help decide the features of hello, adjacency, LSA and calculation of MDRs.

### 4.2 Changes to Neighbor Data Structure

Some new data items are added to the Neighbor data structure to make it MANET compatible. Some of them are necessary for the MDR calculation, like RNL and Report2Hop; some of them show the neighbor’s MDR level and its (Backup) MDR parent; some of them show if the router and the neighbor are dependent on each other; and some show if the neighbor is routable. Here is a list of the data items:

- Neighbor Hello Sequence Number (NHSN)
- Reported Neighbor List (RNL)
- Report2Hop
- Neighbor’s MDR Level
- Neighbor’s MDR Parent
- Neighbor’s BMDR Parent
- Child
- Dependent
- Dependent Selector
- Routable

### 4.3 Hello protocol

This section will describe the modifications and extensions to the Hello data structure and the sending and receiving of MANETs Hellos and.

#### 4.3.1 Hello data structure and Sending Hello Packets

The MANET Hello is an extension to the Hello in an OSPF broadcast network. This subsection presents the modification and extension to the Hello data structure and the options related to sending of Hello packets.

#### 4.3.1.1 DR and BDR

The existing DR and Backup DR field are modified to reflect MANET needs. These fields are modified to indicate not only MDR or Backup MDR, but also MDR Parent and Backup MDR Parent. For the DR field, if the router is an MDR, the field is the router’s own Router ID; otherwise it is the router’s MDR Parent, or is 0.0.0.0 if the MDR Parent is null. For the Backup DR, if the router is a Backup MDR, the field is the router’s own Router ID, if the
router is an MDR, this field is the router’s MDR Parent; otherwise it is the router’s Backup MDR Parent, or is 0.0.0.0 if the Backup MDR Parent is null.

### 4.3.1.2 Option D
Differential Hellos are used to reduce the overhead caused by the full Hellos. A new option bit D defined in ospfv3 indicates that only differential information is present. The parameter 2HopRefresh from the MANET interface structure decides the frequency of sending full Hellos. When 2HopRefresh is equal 4, every fourth hello is full state hello that contains full neighbor state information, others are differential hellos. To make all the hellos full state hello, set the 2HopRefresh to be 1.

### 4.3.1.3 Option L and LLS
The Hello packets are extended by LLS (Link Local Signalling) to include MANET required information, such as 2-hop neighbor information. A new option bit L defined in ospfv3 indicates that the Hello packet includes LLS. TLVs (Type Length Value) are the building blocks for the LLS. The full state hellos and differential hellos include different TLVs as described below.

The full state Hellos require Hello Sequence (HS) TLV and may include the Heard Neighbor List (HNL) TLV if the router has neighbors in state Init, and the Reported Neighbor List (RNL) TLV that includes all neighbors in state two-way or higher, excluding any Dependent Neighbors.

The differential Hellos require HS TLV and may include HNL TLV, RNL TLV, and Lost Neighbor List (LNL) TLV based on need. The HNL, RNL and LNL TLVs are all constructed as state-change lists, i.e. the lists include only nodes that transformed to the special state within the last HelloRepeatCount Hellos. In this way, it will reduce the overhead of transferring the Hello messages.

### 4.3.2 Receiving Hello Packets
When a Hello packet is received by a router, the neighbor state machine and interface state machine fire events based on received information, as well as the variables in neighbor and interface data structures are updated.

The L bit indicates that the Hello packet contains LLS. The HS TLV is required and the neighbor state machine will execute the HelloReceived event, otherwise the hello should be discarded. The HS number will be stored in the neighbor’s datastructure.

Then the DR and Backup MDR fields are examined to set MDR level to the router. When the neighbor selects the router as MDR, i.e. the DR field is equal to the router’s own ID, the MDR level of the router is set to 2. When the neighbor selects the router as Backup MDR, i.e. the Backup DR field is equal to the router’s own ID, the MDR level of the router is set to 1. Otherwise the MDR level is set to 0.

When the neighbor selects the router as dependent neighbor, i.e. the router appears in the neighbor’s DNL TLV, the router’s Dependent Selector variable is set to 1. Otherwise, the router’s Dependent Selector variable is set to 0.
Since the full state hellos’ TLVs are different from differential hellos’, the processing is different.

When the hello received is full state, the Report2Hop variable of the neighbor data structure is set to 1. If the neighbour itself appears in one of the three neighbor lists, HNL, RNL or DNL, the neighbor state machine will execute event 2-WayReceived, otherwise 1-WayReceived. If the received RNL list is different from the RNL in the neighbor data structure, the receiving interface’s state machine should schedule the MDRNeighborChange event. The RNL in the neighbor data structure should be replaced by a union of received RNL and DNL TLV neighbor lists.

When the hello received is differential hello, the Report2Hop variable of the neighbor data structure is set to 0. If the LNL TLV exists, nodes in the LNL TLV should have newly changed to state DOWN, which are lower than the RNL’s 2-Way and above state. If the router itself is one of the lost neighbors, i.e. appears in the LNL TLV, the neighbor state machine should fire event 1-WayReceived; and then removed the router from the RNL list of the neighbor data structure if it is there. If the received LNL TLV includes nodes from the RNL list of the neighbour data structure, remove them and schedule MDRNeighborChange.

If the HNL TLV exists in the differential hello, nodes in the HNL TLV should have newly changed state to INIT, which are lower than the RNL’s 2-Way or higher state. If the router itself appears in the HNL TLV, then the neighbor state machine should fire event 2-WayReceived, and remove the router from the RNL of the neighbor data structure if it is there. If the received HNL TLV includes nodes from the RNL list of the neighbour data structure, remove them and schedule MDRNeighborChange.

If an RNL TLV exists in the differential hello, nodes in the RNL TLV should have newly changed state to 2-Way or higher. If the router itself appears in the RNL TLV and not in the LNL or HNL TLV, the neighbor state machine should fire event 2-WayReceived, and add the router to the RNL of the neighbor data structure if it is not there. If a node in RNL TLV is not included in RNL of the neighbour data structure, add it and schedule MDRNeighborChange.

If the router itself is not in any of the differential TLVs, and the state is 2-way or greater, and the HS number does not expire, then the neighbor state machine should fire 2-WayReceived. If neither 2-WayReceived nor 1-WayReceived is fired, 1-WayReceived should be fired.

Changes to the neighbor’s dataitems do not depend on full or differential hellos, and they will cause the interface state machine to fire MDRNeighborChange event and the neighbor state machine to fire the AdjOK? event.

- Change of the neighbor’s Router Priority will require MDRNeighborChange event;
- Change of a bidirectional neighbor’s MDR level will require MDRNeighborChange and AdjOK? events;
- Change of the neighbor’s Child status or Dependent Selector status from 0 to 1 will require AdjOK? event;
- Change of the neighbor’s state from less than 2-Way to 2-Way or greater will require MDRNeighborChange and AdjOK? events;
- Change of the neighbor’s state from 2-Way or greater to less than 2-Way will require MDRNeighborChange.
Here is an overview of the changes occurs to the neighbor and interface data structure, and the events fired by the neighbor and interface state machines.

<table>
<thead>
<tr>
<th>Neighbor data structure update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report2Hop (full state hello)</td>
</tr>
<tr>
<td>RNL (full state hello RNL TLV; differential hello LNL HNL RNL TLVs)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface data structure update</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDR level (DR/Backup DR field)</td>
</tr>
<tr>
<td>Dependent Selector (DNL TLV)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Neighbor state machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>HelloReceived (HS TLV required)</td>
</tr>
<tr>
<td>1-WayReceived / 2-WayReceived (full state hello/HNL RNL DNL TLV)</td>
</tr>
<tr>
<td>1-WayReceived (differential hello/LNL TLV)</td>
</tr>
<tr>
<td>2-WayReceived (differential hello/HNL TLV)</td>
</tr>
<tr>
<td>AdjOK? (change of the neighbour’s MDR level, Child status, Dependent Selector status, state)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface state machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDRNeighborChange (full state hello/new RNL; differential hello/changed RNL; change of the neighbor’s Router Priority, MDR Level, state)</td>
</tr>
</tbody>
</table>

### 4.4 MDR selection algorithm

The MDR selection algorithm determines if a router is a MDR, Backup MDR or MDR other. MDRs and Backup MDRs are the central concept of OSPF-MDR. MDRs are used to provide a minimal set of relays (connected backbone network) for flooding LSAs. When AdjConnectivity=2, the Backup MDRs are used to provide Bckup relays (biconnected backbone network) to flood LSAs when the flooding by MDRs does not succeed. Dependent Neighbors and (Backup) MDR Parent are also selected by the algorithm, and they are used for decide which neighbors should become adjacent.

The MDR selection algorithm uses a triplet (MDR Level, RtrPri, RID) as a combined priority value. This means the algorithm prefers a router that is already MDR. In the start phase where no MDR is selected, the algorithm prefers those routers with higher RtrPri, which can be a dynamic value based on the router’s battery level etc.

The algorithm is divided in 4 phases. The first phase creates the neighbour connectivity matrix (NCM); the second phase selects MDR and decides dependent neighbors; phase 3 selects Backup MDR and decides dependent neighbors; phase 4 selects the MDR Parent and Backup MDR Parent.

#### 4.4.1 Phase 1: Neighbor Connectivity Matrix (NCM) creation

Suppose we are calculating the MDR for a router A, NCM is a matrix describing if the neighbors of the router A are neighbor to each other. The matrix has two type of values 1 or 0, 1 means that the two neighbors of router A are neighbors to each other, 0 means they are not. The calculation is dependent on two variables: Report2Hop and RNL. Both of the variables are part of the neighbor data structure, and are set during the processing of Hello packets.
Suppose (j, k) is a neighbor pair of the router A, the NCM value is symmetric, i.e. NCM(j,k)=NCM(k,j) and it is set after the following criteria:

1). If the Report2Hop is 1 for both j and k, i.e. both of them have received full hello from the router A, then NCM (j,k)=NCM(k,j)=1 only if j belongs to k’s RNL and k belongs to j’s RNL.

2). If the Report2Hop is 1 for j and 0 for k, i.e. only j has received full hello from the router A, then NCM(j,k)=NCM(k,j)=1 only if k belongs to j’s RNL.

3). If the Report2Hop is 0 for both j and k, i.e. none of them has received full hello from the router A, then NCM(j,k)=NCM(k,j)=0.

In the following example I use the same topology as in MPR selection. Suppose that Report2Hop=1 for all the nodes, and RNL is complete for all the nodes. Since MDR is calculated for every node, we need to calculate the NCM for every node in this phase.

The node n0 has two 1-hop neighbors, n1 and n2. The NCM for n0 is a symmetric matrix of n1 and n2. Since n1 and n2 are neighbors to each other, NCM(n1,n2)=NCM(n2,n1)=1. The node n1 has three 1-hop neighbors, n0, n2 and n3. Since n0 and n2 are neighbors to each other, NCM(n0,n2)=NCM(n2,n0)=1; n0 and n3 are not neighbors, NCM(n0,n3)=NCM(n3,n0)=0; n2 and n3 are not neighbors, NCM(n2,n3)=NCM(n3,n2)=0.

For the total results, refer to the NCM tables in the figure below.
Phase 2: MDR selection

This phase selects the MDR and sets up the dependent neighbors set.

1) The set of dependent neighbors is initially empty.
2) If the router has the highest triplet value, it selects itself as MDR and all its MDR neighbors as dependent neighbors, and its Backup MDR neighbors as dependent neighbors when bi-connected adjacencies are required, i.e. when AdjConnectivity = 2.
3) Otherwise if one of the neighbor has the highest triplet value, call it Rmax.
4) Based on the NCM from phase 1, calculate the shortest path from Rmax each other neighbour u and assign it to hops(u). All the nodes on each path should have a higher triplet than the router itself. If no such path found, hop(u) is set to be infinity. When hops(u) <= MDRConstraint for each neighbor u, then the router doesn’t select itself as MDR. Otherwise if at least one of the neighbor’s hops(u) > MDRConstraint, the router selects itself as MDR, and selects Rmax and all its MDR neighbors with hops(u) > MDRConstraint, and all its Backup MDR neighbors with hops(u) > MDRConstraint when bi-connected adjacencies are required.

The hops(u) calculation can be based on a breath-first-search (BFS) algorithm, modified to allow only nodes with higher triplet than the router itself as intermediate nodes.
Figure 16: MDR selection on a wireless example
The MDR level for all the nodes are initially 0, and in this example the RtrPri is equal for all nodes, then the RouterID is the only deciding variable of the triplet in the start. Suppose that MDRConstraint=2.

MDR selection for node n0
- n0’s 1-hop neighbors are n1 and n2.
- Since n0’s triplet value is not the largest, hops(u) calculation is required for MDR selection.
- Node n2 has the largest triplet value of the neighbors and is chosen as the Rmax.
- A link cost table c(u,v) is calculated for all the neighbor pairs of the calculating node n0. c(n1,n2)=1 because n1 has a larger triplet value than n0 and NCM(n1,n2)=1; c(n2,n1)=1 for the same reason. The link cost makes sure that only neighboring nodes with higher triplet value can be the intermediate nodes.
- Then hops(u) is initialized to be infinity for all the neighbors u other than Rmax n2, hops(n2)=0.
- Rmax n2 is added to the FIFO queue. While the FIFO queue is not empty, dequeue the head element, i.e. in the beginning n2, for each neighbors v of the calculating node n0 such that c(n2, v)=1. n2 has only cost effective path to n1: if hops(n1)>hops(n2)+1, then set hops(n1)=hops(n2)+1, hops(n1) had value infinity and is larger than hops(n2)+1, therefore hops(n1) is changed to be 1, set parent(n1)=n2, and add n1 to the tails of the queue. Run the dequeue process again, n1 is removed from the queue, only neighbor n2 of the calculating node n0 has cost effective path to n1, i.e. c(n1,n2)=1: because hops(n2)<hops(n1)+1, no changes are made. The queue is empty and no further calculation is needed.
- The final result is hops(n1)=1 and hops(n2)=0.
- None of the hops(u) are greater than MDRConstraint, therefore n0 doesn’t select itself as MDR and selects no dependent neighbors.

MDR selection for node n1
- n1’s 1-hop neighbors are n0, n2 and n3.
- Since n1’s triplet value is not the largest, hops(u) calculation is required for MDR selection.
- Node n3 has the largest triplet value of the neighbors and is chosen as the Rmax.
- A link cost table c(u,v) is calculated first for all the neighbor pairs of the calculating node n1. c(n2,n1)=1 because n2 has a larger triplet value than n1 and NCM(n2,n1)=1. Though NCM(n1,n2)=1, c(n1,n2)=infinity because n1 has a smaller triplet value than n1. All the other pairs have infinity cost because their NCM is not 1.
- Then hops(u) is initialized to be infinity for all the neighbors u other than Rmax n3, hops(n3)=0.
- Rmax n3 is added to the FIFO queue. While the FIFO queue is not empty, dequeue the head element, i.e. in the beginning n3, for each neighbors v of the calculating node n1 such that there exists a cost-effective path from n3 to v, i.e. c(n3, v)=1, no nodes meet the criteria. The queue is empty and no further calculation is needed.
- The final result is hops(n3)=0, hops(n0)=infinity and hops(n2)=infinity.
- Both hops(n0) and hops(n2) are greater than MDRConstraint, therefore n1 selects itself as MDR, none of its neighbors are selected as MDR or Backup MDR, therefore no dependent neighbors are selected yet.

MDR selection for node n2
• n2’s 1-hop neighbors are n0, n1 and n4.
• Since n2’s triplet value is not the largest, hops(u) calculation is required for MDR selection.
• Since Node n1’s MDR Level=2, n1 has the largest triplet value of the neighbors and is chosen as the Rmax.
• A link cost table c(u,v) is calculated first for all the neighbor pairs of the calculating node n2. c(n1,n0)=1 because n1 as an MDR has a larger triplet value than n2 and NCM(n1,n0)=1. Though NCM(n0,n1)=1, c(n0,n1)=infinity because n0 has a smaller triplet value than the calculating node n2. All the other pairs have infinity cost because their NCM is not 1.
• Then hops(u) is initialized to be infinity for all the neighbors u other than Rmax n1, hops(n1)=0.
• Rmax n1 is added to the FIFO queue. While the FIFO queue is not empty, dequeue the head element, i.e. in the beginning n1, for each neighbors v of the calculating node n2 such that there exist a cost-effective path from n1 to v, i.e. c(n1, v)=1. n1 has only cost effective path to n0: If hops(n0)>hops(n1)+1, then set hops(n0)=hops(n1) +1, hops(n0) had value infinity and it’s smaller than hops(n1)+1, therefore it’s changed to be hops(n1)+1=1, set parent(n0)=n1, and add n0 to the tails of the queue. Run the dequeue process again, n0 is removed from the queue. n0 has no cost effective path to any of the neighbors v of the calculating node n2, i.e. c(n0,v)<1, therefore no changes are made. The queue is empty and no further calculation is needed.
• The final result is hops(n1)=0, hops(n0)=1 and hops(n4)=infinity
• hops(n4) is greater than MDRConstraint, therefore n2 selects itself as MDR, its MDR neighbor n1 will be added to its dependent neighbors list.
• The change of n2’s MDR level will cause a recalculation of MDRs for all it’s neighbors. For example, n1 will add n2 to its dependent neighbors list.

MDR selection for node n3
• n3’s 1-hop neighbors are n1 and n4.
• Since n3’s triplet value is not the largest, hops(u) calculation is required for MDR selection.
• Since Node n1’s MDR Level=2, n1 has the largest triplet value of the neighbors and is chosen as the Rmax.
• A link cost table c(u,v) is calculated first for all the neighbor pairs of the calculating node n3. c(n1,n4)=c(n4,n1)=0 because n1 and n4 are not neighbors, i.e. NCM(n1,n4)=NCM(n4,n1)=0.
• Then hops(u) is initialized to be infinity for all the neighbors u other than Rmax n1, hops(n1)=0.
• Rmax n1 is added to the FIFO queue. While the FIFO queue is not empty, dequeue the head element, i.e. in the beginning n1, for each neighbors v of the calculating node n3 such that there exist a cost-effective path from n1 to v, i.e. c(n1, v)=1. n1 has no cost effective path to the only other neighbor n4, therefore nothing is done. The queue is empty and no further calculation is needed.
• The final result is hops(n1)=0 and hops(n4)=infinity
• hops(n4) is greater than MDRConstraint, therefore n3 selects itself as MDR. It selects its MDR neighbor n1 as dependent neighbor. None of its neighbors are selected as Backup MDR, therefore no other dependent neighbors are selected yet.
• The change of n3’s MDR level will cause a recalculation of MDRs for all it’s neighbors. For example, n1 will add n3 to its dependent neighbors list.
MDR selection for node n4

- n4’s 1-hop neighbors are n2 and n3.
- Since n4’s triplet value is not the largest, hops(u) calculation is required for MDR selection.
- Since n3 has the largest triplet value of the neighbors and is chosen as the Rmax.
- A link cost table c(u,v) is calculated first for all the neighbor pairs of the calculating node n3. c(n2,n3)=c(n3,n2)=0 because n1 and n4 are not neighbors, i.e. NCM(n2,n3)=NCM(n3,n2)=0.
- Then hops(u) is initialized to be infinity for all the neighbors u other than Rmax n3, hops(n3)=0.
- Rmax n3 is added to the FIFO queue. While the FIFO queue is not empty, dequeue the head element, i.e. in the beginning n3, for each neighbors v of the calculating node n4 such that there exist a cost-effective path from n1 to v, i.e. c(n1, v)=1. Because n3 has no cost effective path to the only other neighbor n2, nothing is done. The queue is empty and no further calculation is needed.
- The final result is hops(n3)=0 and hops(n2)=infinity
- hops(n2) is greater than MDRConstraint, therefore n4 selects itself as MDR. It selects its MDR neighbor n2 and n3 as dependent neighbor.
- The change of n4’s MDR level will cause a recalculation of MDRs for all its neighbors. For example, n2 and n3 will add n4 to their dependent neighbors list.

4.4.3 Phase 3: Backup MDR selection

The dependent neighbor list will be further updated by the Backup MDR neighbors. The NCM helps determine whether or not there exist two node-disjoint paths from Rmax to each other neighbour u, using only intermediate nodes that are neighbors with a larger triplet value. If such paths exist from Rmax to each other neighbor, then the router does not select itself as a Backup MDR, and selects no additional Dependent Neighbors. Otherwise, the router selects itself as Backup MDR if it is not already MDR. If the parameter AdjConnectivity=1, the following neighbors will be selected as a Dependent Neighbor: Rmax, and each MDR/BMDR neighbour u where there does not exist two node-disjoint paths from Rmax to u.

The way to find node-disjoint path is based on the Suurballe-Tarjan algorithm.

1) Compute the second link cost c2(u,v) for each pair of neighbors u,v. Link cost c2 shows the difference of cost by using two different paths. If there does not exist an effective path c(u,v), c2(u,v) is set to be infinity, otherwise set c2(u,v)=1+hops(u)-hops(v), i.e. if hops(u)+1=hops(v), c2(u,v)=0.
2) Set hops2(u)=infinity for all neighbors u other than Rmax, and set hops2(Rmax)=0. Initially, all neighbors u are unlabeled.
3) The labelled Rmax divides the BFS tree (defined by the parents selected in phase 1) into smaller unlabeled sub-trees, one for each child of node Rmax. Pairs of neighbors u,v are then chosen from different sub-trees so that the paths are disjoint. The hops2(v) is set to be the minimum value of c2(u,v), i.e. if hops2(v)>c2(u,v), then hops2(v)=c2(u,v).
4) While there exists an unlabeled node with a finite value of hops2, let k be the unlabeled node with minimum hops2 value, label k. This will divide the unlabeled sub-tree into smaller unlabeled sub-trees, one parent sub-tree and one or more child sub-trees. When the parent sub-tree does not exist or is already labelled, do nothing and continue with the iteration. Otherwise, first update the hops2 value for all the parent nodes based on the newly labelled node k’s hops2 value and c2(k,u), then
update the hops2 value for all the children nodes, and at last update the hops2 value for all the parent nodes again.

If the resulting hops2 value is finite for a neighbor u, then there exist two disjoint paths to u. If the hops2(u)=0, then both disjoint paths have the same length, i.e. hops(u).

Here is an example on selecting Backup MDRs, using the same network topology as in 4.4.2.

Backup MDR selection for node n0
- First calculate the link cost table c2 for node n0. Since c(n1,n2)=1, c2(n1,n2)=1+hops(n1)-hops(n2)=2. Since c(n2,n1)=1, c2(n2,n1)=1+hops(n2)-hops(n1)=0.
- n2 is Rmax, hops2(n2)=0; hops2(n1)=infinity.
- n2 is labeled, n2 divides the BFS tree into smaller unlabelled sub-trees, one for each child of n2. Node n2 has only one child, therefore only one child sub-tree exists with n1 as the only node on the tree. Since there exist no pairs of neighbors from different sub-trees, hops2(n1) is still infinity.
- n1 is unlabelled, but hops2(n1)=infinity, therefore there does not exist any unlabelled node with a finite value of hops2, no further calculation of hops2 is made.
- hops2(n1)=infinity, there does not exist two disjoint paths to u. Node n0 has not selected itself as MDR before, it selects itself as Backup MDR. If the AdjConnectivity=2, n0 selects Rmax n2 as Dependent Neighbor, and also MDR n1 as Dependent Neighbor since hops2(n1)=infinity.
- The change of MDR level for n0 will trigger the MDRNeighborChange event for its neighbor n1 and n2, but will not cause changes because n0 still has a lower triplet value than all its neighbors.

Backup MDR selection for the other four nodes in the topology is not necessary since they have already selected themselves as MDR in the previous phase.
**4.4.4 Phase 4: (Backup) MDR parent selection**

The last step in MDR selection is to select (Backup) MDR parent. The parent’s triplet value is always larger than the router except that the parent can be null when the router has the largest triplet value in the network. This forms a directional graph from nodes with lower triplet values to higher triplet values. The MDR parent and BMDR parent if exists will form an adjacency with the calculating router.

Figure 17: Backup MDR selection on a wireless example
Suppose that Rmax is the node with the largest triplet value among the bi-directional neighbors, and its value is larger than the calculating router itself. Otherwise if the calculating router has the largest triplet value, the Rmax is null.

If the calculating router selects itself as MDR, then the parent is equal to Rmax, i.e. the neighbor with largest triplet value, or is null if the router itself has the largest triplet value.

Otherwise if the calculating router is BMDR or MDR Other, the parent is selected to be an existing adjacent MDR neighbor if such a neighbor exists, otherwise the Rmax is selected as MDR parent. It avoids forming new adjacencies and reuses the existing adjacencies.

Backup MDR parent is only selected if AdjConnectivity=2 and the calculating router is an MDR Other. Backup MDR parent should not choose the same node as the MDR Parent; therefore it will choose the second candidates after the MDR Parent, first from the existing adjacent MDR neighbor list, then from the bidirectional neighbors with the next largest triplet value.

I will continue to select MDR parents on the same network topology. Suppose there are no adjacencies before the selection. The MDR Parent selection process is described as below:

- n0 is an Backup MDR, it should reuse the existing adjacency if possible. Since there is no adjacencies for n0, n0 choose Rmax n2 as its MDR parent, an adjacency is formed between n0 and n2.
- n1 is an MDR, it chooses Rmax n3 as its MDR parent, an adjacency is formed between n1 and n3.
- n2 is an MDR, it chooses Rmax n4 as its MDR parent, an adjacency is formed between n2 and n4.
- n3 is an MDR, it chooses Rmax n4 as its MDR parent, an adjacency is formed between n3 and n4.
- n4 is an MDR, it chooses Rmax as its MDR parent, and Rmax=null since n4 has the largest triplet value.

Suppose that AdjConnectivity=2, Backup MDR parents need to be selected. Backup MDR Parents should only be selected for MDR others, no MDR Others exist in this case; no Backup MDR Parents are selected.

The figure below shows the MDR Parent relationship by arrows from each child node pointing to a parent node.
4.5 Changes to Interface state machine

Though the MANET interface is new, there are no new states for it. The meanings of the states are otherwise modified to reflect the MANET property. For example, DR state is replaced by MDR state and Backup state is replaced by BMDR state, so as DR other is replaced by MDR other state. Waiting state describes in MANET the waiting period between receiving the hello message and the possible MDR selection, in order to prevent the unnecessary MDR selections.

Some of the events that cause interface state changes, like InterfaceUp, InterfaceDown, WaitTimer, LoopInd and UnloopInd follow the [RFC2328] description. Two other events are modified to fit the MANET property. BackupSeen event is never invoked for MANET. NeighborChange event is replaced by MDRNeighborChange event that runs the MDR Selection algorithm.

As described earlier in 4.3.2, processing hello messages reveals the changes in the neighbors, and then initiates MDRNeighborChange event. As shown below, both change of a neighbour and change of the calculating router’s router priority will trigger the MDRNeighborChange event.
The received interface events like MDRNeighborChange trigger an action and change the interface state, and the action taken depends on the current interface state. This process is called interface state transition and is organized by the Interface State Machine.

Most of the interface state transitions follow [RFC2328] standard except InterfaceUp, WaitTimer and MDRNeighborChange.

**InterfaceUp event** triggers state transition when the current interface is Down. It will start the Hello Timer to send out periodical hellos. The resulting interface state depends on if the router is eligible to be an MDR. If it is not eligible, the state change to DR other, otherwise the state changes to be waiting and waiter timer is started.

**WaitTimer event** triggers state transition when the state is Waiting. It will run the MDR Selection algorithm to calculate the MDR Level for the interface, and its dependent neighbors and MDR Parent. The change of interface state may invoke the AdjOK? neighbor event to maintain the adjacencies.

**MDRNeighborChange** triggered state transition runs the MDR selection algorithm to recalculate the MDR Level, dependent neighbors and MDR Parent for the interface. The change of interface state may invoke the AdjOK? neighbor event to maintain the adjacencies.
4.6 Adjacency maintenance

Adjacency is formed between (Backup) MDRs. Therefore the change of MDR Levels for nodes in the topology, either the neighbors or the router itself will trigger the AdjOK? event. AdjOK? is also invoked when a neighbor state changes from Init to 2-Way, and the neighbor is selected to be the (Backup) MDR Parent or the neighbour selects the router to be its (Backup) MDR Parent.

The following state-event diagram shows changes made to the Neighbor State Machine for MANET the interface. Transition from the current state to a new state is triggered by an event, and may be decided by a criterion, shown as choice in the diagram.
When the neighbor state is 2-Way, the AdjOK? event decides whether a new adjacency should be formed. One of the following criteria should be true:

1) Both the router and the neighbor are (Backup) MDRs and there are “dependent” relation between the router and the neighbor. The neighbor is either a Dependent Neighbor, i.e. dependent on the router, or the neighbor is a Dependent Selector, i.e. the router is dependent on the neighbor.
2) The router is a (Backup) MDR and is the neighbor’s parent.
3) The neighbor is a (Backup) MDR and is the router’s parent.

When the neighbor state is ExStart or higher, the AdjOK? event decides whether an existing adjacency should be kept, the criteria are either the router or the neighbor is (Backup) MDR.
In addition to the changes to Neighbor State Machine and AdjOK? event, the procedures of sending and receiving Database Description packets are also modified for the MANET interface.

4.7 Flooding procedure

Link State Update (LSU) packets help forwarding Link State Advertisements (LSAs). Each LSU packet may contain several LSAs. The reliability of the flooding procedure is based on that every LSA should be acknowledged separately, although several acknowledgements can be bundled together in a single packet.

4.7.1 Flooding LSAs

The flooding procedure starts when an LSU packet is received. Every LSA in that LSU packet is filtered through a list of pre-requirements to eliminate the not-to-be-forwarded LSAs. In addition to the requirements from [RFC2328], for a MANET interface, only LSUs received from a neighbor with state 2-Way or higher will be forwarded because neighbors with state lower than 2-Way are not routable.

The second process of the flooding procedure is to select the outgoing interfaces and add the LSA to the appropriate neighbor’s Link State retransmission lists. The purpose is to reduce the unnecessary transmission of LSAs to neighbors that have already received them before. For example, a neighbor need not forward the LSA if the LSA or an ACK for this LSA has already been received from it or the neighbor is already covered by a previous transmission. A MANET interface should not forward the LSA if none of the bidirectional neighbors need to forward the LSA. The interface will only forward an LSA if the router is a (Backup) MDR for this interface. When the router is a Backup MDR for the interface the LSA is not immediately flooded from the interface, i.e. the interface will wait BackupWaitInterval before deciding whether to flood the LSA.

4.7.2 Retransmitting LSAs

An LSA is retransmitted by a configurable interval until its ACK is received. The retransmitting happens always on an adjacency. In the MANET interface, no adjacencies are made between MDR others; therefore an MDR other only retransmits the LSA to its (Backup) MDR Parents. To reduce the network traffic, the retransmissions are bundled in a single LSU packet whenever possible.

4.7.3 Sending the LS ACKs

When a router receives an LSA, it should decide to send a delayed ACK, an immediate ACK or no ACK. An immediate ACK is sent for example when a retransmission is received on a (Backup) MDR to prevent other adjacency neighbors to retransmit the same ACK. A delayed ACK is normally sent when a retransmitted LSA is received on an MDR other or when a new LSA is received. No ACK is sent when a duplicated LSA is received.

4.7.4 Receiving the LS ACKs

Receiving the LS ACKs follows [RFC2328] with some small modifications. Each LS ACK in the received LS ACK packet is processed separately. If the LSA in the LS ACK does not exist in the database or is newer than that in the router’s database copy, an ACK is added to the Acked LSA list. When the LSA in the LS ACK is the same as that in the router’s database copy, remove the sending neighbor from the BackupWait Neighbor list if it exists.
4.8 Changes to LSAs

In a MANET interface, each router advertises only a subset of its MANET neighbors as point-to-point links in its router-LSA. Fully adjacent neighbors are not enough for calculating shortest path; therefore other neighbors are added to the router-LSA to achieve different level of shortest path. These non-adjacent neighbors should be routable. A neighbour is routable either when its state is Full or when its state is 2-Way or higher and there exists a route to the neighbor and the neighbor agrees the connection to the router is bidirectional. Which neighbors to advertise is based on a configurable parameter LSAFullness.

<table>
<thead>
<tr>
<th>Type</th>
<th>LSAFullness</th>
<th>Router-LSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0</td>
<td>Only fully adjacent neighbors are included</td>
</tr>
<tr>
<td>Min-cost</td>
<td>1 or 2</td>
<td>Minimum set of neighbors necessary to provide a shortest path between each pair of neighbors</td>
</tr>
<tr>
<td>MDR Full</td>
<td>3</td>
<td>Only (Backup) MDRs originate Full LSA, MDR others originate minimum LSA</td>
</tr>
<tr>
<td>Full</td>
<td>4</td>
<td>All routable neighbors are included</td>
</tr>
</tbody>
</table>

Table 4: LSA Fullness

4.9 Routing table calculation

The routing table calculation is modified to always include all the routable neighbors for shortest path calculation. Only when the parameter LSAFullness=4, all the routable neighbors are included in the router-LSA, only part of the neighbors are included in the router-LSA when LSAFullness=3. Therefore the routing table should be recalculated whenever the list of full routable tables changes even if the router-LSA does not change.
5 The OSPF-MDR implementation in J-Sim

This chapter describes the implementation of OSPF-MDR in J-Sim. It first introduces the current OSPF implementation in J-Sim and makes comments on the extension related features. Then it presents the wireless extension implementation in J-Sim network simulation environment in details divided into several subsections.

5.1 The current OSPF implementation in J-Sim

The current OSPF implementation is bundled into a java package called package drcl.inet.protocol.ospf. The current implementation is mainly limited to the point-to-point network type. OSPF-MDR will reuse several features related to broadcast network and point-to-multipoint network, but neither of the network types is implemented yet.

5.1.1 OSPF_Interface

The OSPF_Interface class includes the data items and methods for the Interface Data Structure defined in chapter 9 of [RFC2328]. The OSPF interface type is either point-to-point, broadcast, NBMA, Point-to-MultiPoint or virtual link, and all the types are assigned with a final static int value in the OSPF class.

Implemented data items are interface id, type, state, neighbor list etc. Data items like DR or BDR are not implemented.

The Interface State Machine in OSPF-MDR is described earlier in section 4.5. The implemented Interface State Machine events are: interface_up, interface_down and neighbor_change. The other Interface State Machine events such as WaitTimer, BackupSeen, LoopInd and UnloopInd are not implemented. Other implemented methods in the OSPF_Interface class are used to maintain the neighbor list like add_new_neighbor and remove_neighbor, or to create the interface like ospf_interface_create etc. The modification of Interface State Machine is later implemented in 5.2.4.

DR and BDR selections are invoked by the Interface State Machine for broadcast and NBMA networks as specified in [RFC2328], and are not implemented.

5.1.2 OSPF_Neighbor

The OSPF_Neighbor class includes the data items and methods for the Neighbor Data Structure defined in chapter 10 of [RFC2328]. Neighbor data structure describes the information exchanged with other adjacent nodes, and it is bounded to a specific OSPF router interface.

Implemented data items are neighbor id, neighbor’s router id, neighbor’s IP address etc. Data items like DR or BDR are not implemented. Several implemented methods are related to link state advertisement, for example, ospf_add_summary, ospf_remove_summary, ospf_add_request etc.

5.1.3 OSPF_Hello

Hello packages are sent out on broadcast and NBMA networks to discover and maintain neighbor relationships.
The OSPF_Hello class includes data items and methods from the [RFC2328], but the implementation is limited and not complete. A single shot hello is sent after the InterfaceUp event to confirm the 2-Way. The OSPF developers use otherwise the J-Sim’s drcl.inet.core.Hello to discover neighbor to simplify its implementation.

The OSPF_Hello maintains a neighbor_id_list, and implements methods to add or remove the neighbors from the list.

5.1.4 OSPF class

The OSPF class is the main class for the routing protocol. It provides the necessary data items for the router like area, interface list etc. It provides the functions for sending and receiving packets, for example hellos and LSAs and data packets. It also maintains the routing table.

5.1.4.1 Sending hellos

The OSPF implementation use the drcl.inet.core.Hello as underlying Hello service provider, and another method called ospf_send_single_hello() to ensure the two-way connection. The drcl.inet.core.Hello will take care of the tasks like neighbour discovery. The ospf_send_single_hello() sends out a hello without starting the next hello timeout event. This method is called by neighborUpEventHandler() and will be executed only once to make sure the two-way connection.

OSPF implementation also provides a simple sending hello method called ospf_send_hello() in the OSPF class though it is not in use. The sending hello process is triggered by an OSPF_TIMEOUT_HELLO event in the timeout() method. This ospf_send_hello() method builds a hello package by adding the neighbor ids and then broadcast the hello on the interface, and starts the next hello timeout event to ensure the periodical hello timeouts.

5.1.4.2 Receiving hellos

When the hello packet arrives the data port of the OSPF router, the dataArriveAtDownPort() method is triggered, and it checks the message type before it calls the Process_Hello() method. The method checks if it is a two-way connection, and then calls the Neighbor State Machine events _nbr_Hello_Received() and _nbr_Twoway_Received() .

The following figure shows the relationship between OSPF_Interface, OSPF_Neighbor and OSPF_Hello. Every OSPF router can have several interfaces, and each interface can have be connected to several neighbors. The router broadcast hellos to all its neighbors on a specified interface, and hellos are also received on a specified interface to update the interface’s neighbor information.
The OSPF implementation relies on CSL Hello to maintain the neighbor events, therefore the Nbr_Inactive_EVT is not initialized when processing hellos.

### 5.1.4.3 Adjacency Maintenance in OSPF implementation

The Neighbor State Machine events are implemented in OSPF.java class, among them _nbr_Twoway_Received and _nbr_determineAdjacency are related to Adjacency maintenance as described earlier in section 4.6.

The _nbr_Twoway_Received events are triggered when a hello is received and the neighbor’s state is changed to twoway. In the OSPF implementation the _nbr_determineAdjacency is only called when the neighbor state changes to twoway, and the criteria to form the adjacency is simple, i.e. the interface state is PTOP, DR or BDR. Both of the methods will be modified later in the OSPF-MDR implementation as described later in section 5.2.6.

The sending of DBDESC packets is implemented as ospf_send_dbdesc method in OSPF.java class. It is called by _nbr_Twoway_Received method to send DBDESC packets after the neighbor’s state changes to twoway. The Processing of DBDESC packets is implemented as Process_DBdesc method, and it will trigger the _nbr_Exchange_Done event and change the neighbor state to be Full.
5.1.4.4 Router LSA and Network LSA
There are two types of LSA, Router LSA and Network LSA. The Router LSA is implemented here for Point-To-Point network, and it is sent from every router to its neighbors when for example one of its neighbors changes state to be Full. The ospf_make_router_lsa method creates the Router LSA. The Router LSA is reused though modified in OSPF-MDR as described later in section 5.2.7.

5.1.4.5 LSA_flooding
The flooding of LSAs is implemented as the ospf_lsa_flood_interface method. To save the bandwidth, the LSAs are packed together to be LSUpdate packet before flooding out on the interface. Processing of the received LSUpdate packets is implemented by the Process_LSupdate method. The LSU packets are divided into LSAs and every LSA is processed by calling the lsa_receive method. The lsa_receive method will decide if the LSA should be further flooded or the routing table should be updated.

5.1.4.6 SPF and Routing table calculation
Updating the routing table is done by calling the ospf_lsbdb_install method. The ospf_lsbdb_install method will further call the lsa_change method if the received LSA does not exist before or newer. The lsa_change method will call the ospf_spf_calculate to calculate the shortest path and update the routing table.

5.1.5 Summary of the existing OSPF implementation
The OSPF implementation is limited to point-to-point network type and therefore is not completed in several ways. An optimal solution may be to complete the OSPF implementation before extending it with MDR. Due to the time and resource limitation, only the MANET reused features are added in this OSPF-MDR implementation. However, a complete OSPF implementation will be on the future work list.

5.2 Implementing the OSPF-MDR extension in J-Sim
The sections below in this chapter present my implementation of the OSPF-MDR in J-Sim network simulation environment, divided into different subsections based on the description in Chapter 4.

5.2.1 Implementing a new MANET interface
This section describes the modifications and extensions I have done to the existing OSPF Interface implementation to realize the OSPF-MDR specification as described in section 4.1.

5.2.1.1 New or modified data items for MANET interface
The OSPF_Interface class is modified by first adding the missing [RFC2328] data structures that are to be reused in the MANET interface. Router priority (rtr_pri) and router id (rtr_id) are used to calculate the triplet value in the MANET to decide if a router has a better chance to be MDR. Designated Router (dr) and Backup Designated Router (bdr) are reused as MANET Designated Router (mdr) and Backup MANET Designated Router (bmdr), therefore are also necessary in the OSPF-MDR implementation.

Then new MANET data items as described in 4.1.2 are added to the OSPF_Interface class. The rtr_pri should be a dynamical parameter controlled by the other factors like router’s willingness and battery levels etc., but it is not realized in this implementation. The MDR
Level (mdr_level), MDR Parent (mdr_parent) Backup, MDR Parent (bmdr_parent), Hello Sequence Number (hsn) and Lost Neighbor List (LNL) are added. To simplify the implementation of sending and receiving hellos, the Heard Neighbor List (HNL), Reported Neighbor List (RNL) and Dependent Neighbor List (DNL) are also added to the OSPF_Interface though they are not included in the change list of interface in [OSPFMDR].

The Type data item already exists, but the available type values defined in OSPF class are extended by the new interface type IF_MANET.

```java
final static int  IF_MANET = 5;
```

The State data item already exists, and the available values for the State data item are defined in the OSPF class. The only change here is the meaning of IFS_BDR is changed to be BMDR and IFS_DR is changed to be DR.

```java
final static int  IFS_BDR  = 6;
final static int  IFS_DR  = 7;
```

Other configurable parameters like MDRConstraint, AdjConnectivity etc. are added as described in 4.1.3.

### 5.2.1.2 New or modified methods for MANET interface

The OSPF_Interface constructor and the reset( ) method are updated with the new data items. For example, mdr_level is initialized to be 0, i.e. MDR Other.

The ospf_interface_create( ) method did not assign the type variable when creating the interface, and this error is now corrected so that the MANET type will be assigned and be recognizable later.

The add_new_neighbor( ) method is modified so that the neighbor is added only if it is not already on the list to avoid duplicates.

A new neighbor change event called MDRNeighborChange( ) is added, so are the related methods to select (Backup) MDRs and (Backup) MDR Parents. Details will come later in 5.5.

### 5.2.1.3 Interface creation

A point-to-point network creates a new interface for every pair of neighbors. MANET is a point-to-multipoint network where several wireless nodes can share the same interface. To simplify the implementation and avoid a bug in TCL script, the thesis supposes that all MANET nodes in the topology share one interface (interface 0) though the multi-interfaces scenarios are also normal in a MANET network.

In theory the TCL script can also set the type of the interface by calling the _setInterfaceInfo method with more parameters. The following example set the interface type to be 5 (IF_MANET). However this method does not work well in practice and need further error checking. A temporary solution is to hardcode the interface type when creating the interface. The final target to develop a routing protocol that supports multiple interface types will depend on that the method of setting types in TCL works as it should be.

```bash
! hello _setInterfaceInfo 0 [java::new drc1.inet.data.InterfaceInfo 5 2 -8 -1 -1 -1]
```

The OSPF_Interface object is created in the OSPF class by the new_interface( ) method when a received hello does not have a target interface. This method will retrieve the property of the interface from the InterfaceInfo and set them on the new object.
5.2.2 Changes to the neighbor data structure

The OSPF_Neighbor class is modified by first adding the missing [RFC2328] data structures that are to be reused in the MANET interface. The variables dr and bdr represents the neighbor’s idea of the Designated Router and Backup Designated Router, and they are defined only for broadcast, NBMA in [RFC2328]. Both dr and bdr are reused in MANET, although the definition is modified to be MDR and BMDR. The rtr_pri represents the neighbor’s router priority from the last received hello, and it is necessary for MDR selection therefore added to the neighbor class.

Then new MANET data items as described in 4.2 are added to the OSPF_Neighbor class. The variables such as nhsn, report2hop and RNL are updated whenever a hello is received from the corresponding neighbor as described earlier in 4.3.2. The nhsn records the last received hello sequence number from the neighbor. The report2hop is set to be true when the last received hello is full. The RNL is updated by the RNL TLV of the hello packet. The variables such as mdr_level, mdr_parent, bmdr_parent, dependent are updated by the MDR selection process as described earlier in 4.5. The variable routable is updated during the LSA flooding process.

Some other variables are necessary for MDR selection, such as hops, hops2, labeled and parent. The variable “hops” records the distance to Rmax during selection of MDR. The variable “hops2” records the difference of distance to Rmax using two disjoint paths during the selection of Backup MDR. The variable “labeled” is a help variable for the selection of Backup MDR.

5.2.3 MANET Hello

The CSL Hello does not include the necessary data items for OSPF-MDR and the sending or processing functions are not completely implemented, therefore extensions are needed for both the hello message data structure and the sending and processing functions. The MANET Hello implementation is covered by TCL scripting, OSPF class, OSPF_Hello class, OSPF_Packet class and OSPF_Interface class.

5.2.3.1 Extending the OSPF_Hello class

The OSPF_Hello class is extended by modifying and adding necessary data items for the MANET interface.

Some of the data items from OSPF are missing but are necessary for OSPF-MDR and they are added in this implementation, for example the DR and BDR. They have the same data type as router ids.

Though the Hello packet's Neighbor Router ID list should not necessary on the MANET interface as stated in [OSPFMDR] 4.1, it is still used to simplify the implementation by making it easy to look up the neighbor object after a given neighbor router id.

The OSPF implementation is based on OSPFv2. Two option fields from OSPFv3, option L and option D are required and added in the OSPF-MDR implementation. Both of them are l
bit field to indicate true or false. Option L is true when the hello packet is attached with LLS. Option D is true when the hello is differential and false when the hello is full.

The OSPF_Hello data structure is also extended with Link Local Signaling (LLS). The LLS data structure is represented by a new class called OSPFMDR_LLS. Another new class called OSPFMDR_LLS_TLV is added to represent the TLV data structure. OSPFMDR_LLS is a Vector collection of OSPFMDR_LLS_TLVs. It maintains the collection by the methods such as add_lls_tlvs( ) and update_tlv( ) etc. The OSPFMDR_LLS_TLV class includes the three main data items, i.e. type, length and value. The variable “value” in OSPFMDR_LLS_TLV is again a Vector collection of the Integer objects of the neighbors’ router ids. It is also possible to make the variable “value” a collection of the OSPF_Neighbor objects of the neighbors in the j-sim implementation. An object based collection will be easier in java based implementation, while in the real world the transferring of objects are unnecessary and waste of bandwidth, compared to only transferring the router ids. The TLV data structure means to be so flexible that all types of TLVs can be covered by it.

The following class diagram shows the relationship between OSPF_Hello, OSPFMDR_LLS and OSPFMDR_LLS_TLV classes. Every hello has one LLS, and every LLS is a collection of different LLS_TLVs.

![Figure 23: Class diagram OSPF_LLS and OSPF_LLS_TLV in the OSPF-MDR implementation](image)

The duplicate( ) method is updated by adding all the new data items on the duplicating list. It will duplicate the hello object when transferring from one port to another port; without the duplication, the new data items will be missing when reaching the target port.

A new method called get_LLS_TLV_values( ) is added to retrieve the LLS TLV values after the given LLS TLV type. This method returns a Vector list of all the LLS TLV values for the specified LLS TLV type.
Six different types of LLS TLVs are defined in the OSPF-MDR. The final static values are added into the OSPF class as the other constant values.

```java
final static int OSPFMDR_LLS_TLV_TYPE_HNL=11;
final static int OSPFMDR_LLS_TLV_TYPE_RNL=12;
final static int OSPFMDR_LLS_TLV_TYPE_LNL=13;
final static int OSPFMDR_LLS_TLV_TYPE_HSN=14;
final static int OSPFMDR_LLS_TLV_TYPE_MDR=15;
final static int OSPFMDR_LLS_TLV_TYPE_DNL=16;
```

5.2.3.2 Hello configuration in the TCL script

For each node, the hello is created as a drcl.inet.core.Hello object, and its interface (service_if@) is connected to the interface (service_if@) of the node’s OSPF router.

```tcl
mkdir drcl.inet.core.Hello    hello
connect ospf/.service_if@ -and hello/.service_if@
```

The interface port should be added in the OSPF class to support the communication with the Hello component.

```java
Port ifport = addPort(".service_if", false);
```

The interfaceinfo of the hello is defined in the TCL script to decide the interface id and local address. The following line defines interface 0 for the hello of node 2, the interface’s local address is 2 and mask is FFF8. Refer to 5.2.3 on interface creation.

```tcl
! hello _setInterfaceInfo 0 [java::new drcl.inet.data.InterfaceInfo 2 -8]
```

5.2.3.3 Hello timeout in OSPF-MDR

The hello should be periodical and the timeout timer should be started when the router starts. OSPF class implements ActiveComponent so that method _start() is invoked at the start, and the hello timeout timer is started in the _start() method.

The hello interval should be a random number, otherwise the receiving of hellos can be denied because the routers are busy with sending activities.

```java
double interval = OSPF_Interface.MinHelloInterval + ((OSPF_Interface.MaxHelloInterval - OSPF_Interface.MinHelloInterval) * rand.nextDouble());
```

When a hello times out, the timeout() method is called in the OSPF class. If the interface for the hello is not defined yet, first create and bring up a new interface, and then send out the hello message on this interface. The new_interface() method will query the IFQuery for the interfaceinfo, and then build a new interface. The interface_up() method will bring up the interface. The hello message is sent by calling ospf_send_hello() message.

5.2.3.4 Sending hello in OSPF-MDR

The ospf_send_hello() method will build up a hello message and then call the ospf_message_broadcast() method to broadcast it, and it will also start the next hello timeout on the interface. The ospf_message_broadcast() will build an OSPF packet and add the hello object to the packet body, and it will keep forwarding packets on the PTOP network and broadcast hellos on a MANET network.
The `ospf_send_hello()` method is called for both MANET and non-MANET hellos. A Boolean variable `isMANET` will help decide if it should build and send a MANET hello or not. Some of the fields are missing but are necessary for OSPF-MANET and they are added to the hello, such as `dr`, `bdr` and `rtr_pri`. The MANET hello has additional fields to record `mdr_level` and two new option fields: `options_L`, `options_D`. The hello assigns the variables with corresponding values from the interface except the two option fields. The variable `options_L` is set to be true to indicate that there is LLS attached to the hello. The variables `option_D` is true if the hello is a differential hello, is false if the hello is full. Every TwoHopRefresh hello is full. The thesis has not completed the differential hello implementation; therefore the TwoHopRefresh is set to be 1 so that all the hellos are full. LLS is attached to the MANET hello as an object. The LLS object contains different TLVs depending on if the hello is full or differential. The full hello LLS includes HSN TLV, RNL TLV if it exists and HNL TLV if it exists. The differential hello LLS includes HSN TLV, differential HNL, differential RNL and differential LNL.

The following figure illustrates the process of forming the OSPF Hello Packet. OSPF_Hello object is built by retrieving data from the TCL-script configurable InterfaceInfo and the OSPF_Interface object of the sending interface. The hello object is then encapsulated into an OSPF_Packet object as the packet body. It is also possible to encapsulate the hello object into a UDP packet object before encapsulating it into the OSPF_Packet object, depending on the requirement for the transport.

**Figure 24: Forming the OSPF Hello Packet in OSPF-MDR**

### 5.2.3.5 Processing hello in OSPF-MDR

When a hello packet arrives, the `dataArriveAtDownPort()` method in the OSPF class is called and it checks that the type of the received packet. The `Process_Hello_MANET()` method is called for MANET hellos while `Process_Hello()` method is called for legacy OSPF hellos in the OSPF class.

The `Process_Hello_MANET()` method still refers to the `ospf_nbr_lookup_by_routerid()` method to retrieve the neighbor object based on the router id of the hello source. The RNL
and HNL are then retrieved from the hello packet. If the router is among one of the list, the
twoway is set to be true. The neighbor state machine events _nbr_hello_received( ) and
_nbr_twoway_received( ) are called in the OSPF class as needed to update the neighbor’s
state. Then the other neighbor variables are updated by the information from the hello. During
the updating, a Boolean variable mdrchange is set to be true when any of the following
changes happens to the neighbor: state, MDR Level, router priority and RNL. If mdrchange is
true, the MDRNeighborChange( ) event is triggered on the interface.

The following figure illustrates the process of receiving and processing of a hello packet on
the MANET interface.

```
Figure 25: Processing the OSPF Hello Packet in OSPF-MDR
```

Finally the neighbor inactive timer is started. An event handle is assigned to the Neighbor
Timeout Event, and the timeout event can be canceled if another hello is received within the
timeout period. If no new hello is received, the Nbr_Inactive.EVT is fired and will be
handled by the general timeout( ) method, and the neighbor’s state will change to be down.

Another alternative to detect the broken links in MANET is to utilize the MAC layer’s
linkbroken detection. Modifications in both TCL script and OSPF class are needed for this
purpose. The OSPF class should create a linkbroken event port and implement the
LinkBrokenEventHandler( ) method. In TCL script connect the MAC’s linkbroken port to the
OSPF router’s linkbroken port. This alternative is not realized in this implementation.

```
connect mac/ linkbroken@ -and ospf/ linkbroken@
```
5.2.4 Interface State Machine
This subsection describes the modification and extensions done to the Interface State Machine in the implementation. The existing Interface State Machine implementation in OSPF was described shortly earlier in 5.1.1.

5.2.4.1 MDRNeighborChange
The MDRNeighborChange event is triggered when some MDR Selection related properties have changed for one of the interface’s neighbors, for example: state, router priority, MDR Level, RNL etc. The interface can be in any of the three MDR levels when the event is triggered, and the result will update the MDR level of the interface and trigger other adjacency forming related events. This method is implemented in OSPF_Interface class and coexists with the neighbor_change method from the legancy OSPF.

The method will return without action when the RNL list is null, which means no twoway neighbors are detected for the interface. Otherwise the method will continue the MDR Selection. It will create a temporary OSPF_Neighbor object based on the triplet values of the calculating router, and then find out the Rmax among the twoway neighbors. Both the variables are used in the later MDR selection process. The DNL list is initialized if it is empty and it will be updated during the MDR Selection process. Another variable Rmax1hopNeighbors is initialized and it will be updated during MDR Selection and will be reused during Backup MDR Selection. The MDR Selection implementation is described later in 5.2.5.

5.2.4.2 InterfaceUp
To simplify the simulation, the InterfaceUp( ) is called when a new wireless interface is created to bring up the interface for sending/receiving hello packets.

5.2.4.3 WaitTimer
The WaitTimer is not modified.

5.2.5 MDR Selection
The MDR Selection is implemented in the OSPF_Interface class. The MDR Selection is divided into several phases as described in 4.4. The first phase is to create the Neighbor Connectivity Matrix, the second phase is to select the MDR, the third phase is to select the Backup MDR and the fourth phase is to select the (Backup) MDR Parent.

5.2.5.1 Phase 1: Neighbor Connectivity Matrix (NCM) creation
This phase starts with the twoway neighbors list RNL of the interface, and decides if each pair of the interface’s neighbors are neighbor to each other. They are neighbor to each other only when at least one of the two neighbors have received full hello from the interface (report2hops=1), and the other neighbor is among the RNL list of the neighbor that has received full hello. It is realized by the computeNCM() method and will return a two-dimensional array as the NCM. The NCM is symmetric, which means NCM(x,y)=NCM(y,x), and value 1 means the neighbor x and neighbor y are neighbors to each other suppose that x and y are two of the interface’s neighbors.
5.2.5.2 Phase 2: MDR Selection

The MDRSelection( ) method decides if the current router should be a MDR, Backup MDR or MDR Other, i.e. updates the interface’s MDR Level. The dr field is also updated by the calculating router’s own id if it selects itself as MDR. When the current router has the highest triplet value, it selects itself as MDR. Otherwise another matrix called Link Cost Matrix is calculated to help decide the link cost between every other neighbor and the Rmax based on the NCM from phase 1 and the triplet value. The start node of the link should have higher triplet value than the calculating router, and the start node and end node should be neighbors (NCM=1) so that the link cost from the start node to the end node can be 1. The current router will first calculate the minimum hops between every other neighbor to Rmax if the path exists by allowing links with LCM=1.

If the path does not exist or the hops for any of the neighbors are larger than the parameter MDRConstraint, it will select itself as MDR. The default value of MDRConstraint is three and if results in a minimum MDR set, the value 2 will result in a larger MDR set because there are larger chances for a router to select itself to be MDR when only 2 hops are allowed from the Rmax to the other neighbors.

The calculating of minimum hops is realized by an FIFO queue as a LinkedList object. The Rmax is added to the queue and then the Rmax’s 1 hop neighbors among RNL list, and then the Rmax’s 2 hops neighbors among RNL list and so on. The calculation ends when the queue is empty.

The method will also update the Rmax1hopNeighbors and return the LCM as result, and both of them will be used later in Backup MDR Selection.

5.2.5.3 Phase 3: Backup MDR Selection

The selection of Backup MDR is realized by the BackupMDRSelection( ) method. It starts with calculating the second Link Cost Matrix for all the neighbors on the RNL list. The matrix is based on the first Link Cost Matrix and the hops values from phase 2. If c(u,v) is infinity, then set c2(u,v) to infinity, otherwise set c2(u,v) = 1 + hops(u) - hops(v).

The next step is to calculate the hops2 value of every two-way neighbor in the RNL list. The process is based on a disjoint path algorithm as described in the appendix of [OSPFMDR]. It starts from labeling the Rmax and set its hops2 to be 0. Then it continues with Rmax’s 1 hop neighbor list (Rmax1hopNeighbors). Every 1 hop neighbor forms a subtree based on the “parent” variable set in phase 2, the parent value forms a directed graph since only a neighbor with higher triplet value can be a parent. It traverses all the sub trees and combines all the possible neighbor pairs from different sub trees. Here is a scenario: Suppose Rmax has three 1 hop neighbors among the RNL, then there exists three sub trees. Suppose tree 1 has node A and node B, tree 2 has node C and node D, three 3 has node E, then the complete pair list should be pairs from tree 1 and tree 2 (A, C) (C, A) (A, D) (D, A) (B, C) (C, B) (B, D) (D, B), pairs from tree 1 and tree 3 (A, E) (E, A) (B, E) (E, B) and pairs from tree 2 and tree 3 (C, E) (E, C) (D, E) (E, D). The hops2 values for the pairs is asymmetric which means that hops (A, C) can be different from hops (C, A).

Suppose that every pair has a start node u and an end node v, if hops2(v)=hops2(u)+c2(u,v), hops2(v)=hops2(u)+c2(u,v). After that, a labeling process traverses all the unlabeled nodes. The unlabeled node with minimum value of hops2 is chosen by the
getMinimumHops2Unlabeled( ) method, both of the node’s parent and children are updated with a smaller hops2 value if it applies. The process continues until all the nodes are labeled. When the calculation ends, the hop2(v) records the minimum value of hops2 for each twoway neighbor. If any of the neighbors has a infinity hops2 value, there does not exist a disjoint path for the neighbor and the router selects itself as Backup MDR, the DNL list is updated by those neighbors with infinity hops2 values.

5.2.5.4 Phase 4: Selection of the (Backup) MDR Parent

The selection of (Backup) MDR Parent is realized by the MDRParentSelection( ) method. If a router has a higher triplet value than the Rmax, its MDR Parent is null. If a router has a lower triplet value than Rmax and the router is an MDR, which implies that Rmax is also an MDR, the router selects the Rmax as MDR Parent. Otherwise if a router is a Backup MDR or MDR Other, the Rmax is selected as MDR Parent when no adjacency exists from before; when adjacent neighbors already exist, choose any of the adjacent neighbors to be MDR Parent to reuse the existing adjacency.

Backup MDR Parent is selected only if AdjConnectivity=2 and the calculating router is a MDR other. The same as choosing the MDR Parent, the existing adjacencies are prioritized. The Backup Parent is selected to be any adjacent neighbor that is an MDR or BMDR, other than the selected Parent if such a neighbor exists. If no such neighbor exists, the Backup MDR is chosen to be a neighbor with the highest triplet value other than the chosen MDR Parent.

5.2.6 Adjacency Maintenance in OSPF-MDR

The _nbr_Twoway_Received and _nbr_determineAdjacency are two of the Neighbor State Machine events and are modified in OSPF-MDR, refer to section 4.6 and 5.1.4.3.

The _nbr_Twoway_Received event calls the MDRNeighborChange Interface State Machine event if the interface type is MANET, otherwise it calls the neighbor_change event.

The _nbr_determineAdjacency event decides whether a neighbor should be adjacent on MANET interface as well as on other types of interfaces. When the interface type is MANET, the criteria listed in 4.6 are used.

5.2.7 Router_LSA

As described earlier in section 4.8, Router_LSA is used in MANET interface as well as in Point-To-Point interface although the method ospf_make_router_lsa is modified for the MANET interface. The LSA_Fullness parameter should be used to determine which subset of the neighbors should be included in the LSA, ref section 9 of [OSPFMDR]. This is not fully implemented in this thesis and currently only LSA_Fullness=0, i.e. only fully adjacent neighbors are included in the LSA. LSA_Fullness=0 results in the minimum amount of LSA flooding overhead, but do not provide routing along shortest path.
5.2.8 Flooding procedure
The modification to the Flooding procedure in OSPF-MDR as described earlier in section 4.7 is not implemented due to time limits on the thesis. The flooding procedure from the legacy OSPF is reused here.

5.2.9 Routing table calculation
The modification to the Routing table calculation in OSPF-MDR as described earlier in section 4.9 is not implemented yet due to time limits on the thesis. The routing from the legacy OSPF is reused here.

5.2.10 Sending and broadcasting of the packets in OSPF-MDR
This implementation in this section is not described in the [OSPF-MDR], but is a result of testing in J-Sim.

Sending or broadcasting of the packets should be generally modified for the MANET interface to suit for the MAC medium. The MAC medium silently drops a packet when the medium is busy. For example, a router x broadcasts a hello to its neighbor y and z, y and z will receive the hello and send one DBDESC each back to the x at the same time. The two DBDESC packets will then block out each other. Therefore it is necessary to set a random timeout timer before sending out the packets. The packets are kept in an FIFO queue and to be sent out one by one each time an OSPF_TIMEOUT_SEND_MANET is triggered. The OSPF_TIMEOUT_SEND_MANET a new timeout type added in the OSPF_TimeOut_EVT.java class. A new class called OSPF_MESSAGE is created to simplify the storing of the messages in the queue.
6 Simulation and Scenarios

This chapter presents a four-node’s wireless network scenario and a six-node’s scenario and their simulation results.

6.1 A Four-Node’s scenario

A four nodes’ scenario (see Appendix A) is used here to run the simulation. The similar scenario is also used in the OLSR J-Sim implementation [OLSRJSIM]. It will make easy to compare the results from both simulations.

As shown in the figure below, the four nodes form a multi-hop scenario. The twoway neighbor relationships are represented by the double-arrowed lines between the nodes. Data packets are sent from node 0 to node 3. In the middle of the simulation the node 1 is turned off, thereby all the links to node 1 are broken. The next subsections will first present the TCL script for the scenario and then the simulation result of the scenario.

![A four nodes’ scenario](image)

6.1.1 Wireless network scenario and TCL scripts

This subsection presents the construction of the TCL script for the four-node’s scenario.

6.1.1.1 Create the topology

The TCL script first creates the topology by setting up the wireless nodes. The following script creates a node0 as a drcl.comp.Component object.

```tcl
set node0 [mkdir drcl.comp.Component n0]
```

Then the wireless objects described on Figure 14: Schematic of a wireless node in J-Sim are added as sub components for every node, the hello and ospf objects are among them. The PktDispatcher, RT and Identity from drcl.inet.core are added and the PktDispatcher object is bound with the RT and Identity objects.

```tcl
cd n0
mkdir drcl.inet.mac.LL ll
mkdir drcl.inet.mac.ARP  arp
mkdir drcl.inet.core.queue.FIFO  queue
```
mkdir drcl.inet.mac.Mac_802_11      mac
mkdir drcl.inet.mac.WirelessPhy     phy
mkdir drcl.inet.mac.FreeSpaceModel propagation
mkdir drcl.inet.mac.MobilityModel    mobility
mkdir drcl.inet.core.Hello       hello
mkdir drcl.inet.protocol.ospfmdr.OSPF ospf
set PD [mkdir drcl.inet.core.PktDispatcher     pktdispatcher]
set RT [mkdir drcl.inet.core.RT            rt    ]
set ID [mkdir drcl.inet.core.Identity      id    ]
$PD bind $RT
$PD bind $ID

Then the ports of the sub components are connected or “wired” together. For example, the following line will connect the interface port of the ospf component to the interface port of the hello component.

connect ospf/.service_if@ -and hello/.service_if@

Then the sub components are initialized. For example, the router id is set to be 0 for node 0.

! id setDefaultID   0

The static mode of the hello is enabled. The _setInterfaceInfo line configures the interface id, type mask etc.

! hello setStaticEnabled true
! hello _setInterfaceInfo 0 [java::new drcl.inet.data.InterfaceInfo 5 0 -8 -1 -1 -1]

The position of the node is configured by calling the setPosition method of the mobility component.

! mobility setTopologyParameters  2000.0    0.0   2000.0    0.0   300.0   300.0   10.0
! mobility setPosition 0.0 0.0 0.0 0.0

6.1.1.2  Set up the channel

The drcl.inet.mac.Channel class is used to set up the channel for all the wireless nodes in the simulation. The channel’s capacity is set to be 6.

mkdir drcl.inet.mac.Channel channel
! channel setCapacity 6

The drcl.inet.mac.NodePositionTracker is used to tracks the positions of all the wireless nodes in the simulation. The tracker is initialized by grid.

mkdir drcl.inet.mac.NodePositionTracker tracker
! tracker setGrid 2000.0 0.0 2000.0 0.0 300.0 300.0

Then the channel and the tracker components are connected together.

connect channel/.tracker@ -and tracker/.channel@

The tracker is connected to the report port of the mobility components of every wireless node.

connect /example/n0/mobility/.report@ -and /example/tracker/.node@
connect /example/n1/mobility/.report@ -and /example/tracker/.node@

Then the down port of every node’s WirelessPhy component is connected to the node port of the channel. The channel again attaches the WirelessPhy port from the node.

connect n0/phy/down@ -to channel/.node@
! channel attachPort 0 [! /example/n0/phy getPort .channel]
connect n1/phy/down@ -to channel/.node@
! channel attachPort 1 [! /example/n1/phy getPort .channel]
6.1.1.3 Set up the data traffic

The source and destination node is set first.

```tcl
set _dst 3
set _src 0
```

The data traffic is set up and connected to the source’s PacketDispatcher port.

```tcl
set traffic [java::new drcl.net.traffic.traffic_PacketTrain 1500 0.01]
mkdir [java::call drcl.inet.InetUtil createTrafficSource $traffic 0 $_dst 0] n$_src/cbr
connect -c n$_src/cbr/down@ -and n$_src/pktdispatcher/17@up
```

Then the traffic monitor and plotter components are created to monitor and present the simulation results.

```tcl
set plot_ [mkdir drcl.comp.tool.Plotter .plot]
set tm_ [mkdir drcl.net.tool.TrafficMonitor .tm]
set cfile_ [mkdir drcl.comp.io.FileComponent .cfile]
$cfile_ open "bytecount.txt"
connect -c n3/pktdispatcher/17@up -to $tm_/in@
connect -c $tm_/bytecount@ -to $plot_/0@1
connect -c $tm_/bytecount@ -to $cfile_/in@
```

6.1.1.4 Debugging in the TCL scripts

The “setflag trace true” script is used to trace and debug the events on different ports, especially useful when for example a packet is missing during the transformation.

```tcl
setflag trace true ospf/down@
setflag trace true pktdispatcher/0@down
setflag trace true pktdispatcher/103@up
setflag trace true ll/up@
setflag trace true ll/.mac@
setflag trace true mac/.linklayer@
setflag trace true mac/up@
setflag trace true queue/output@
setflag trace true mac/down@
setflag trace true phy/up@
setflag trace true mobility/.query@
```

Debugging can also be set on the routing protocol component of a specified node. The following line sets debug flag on node 0’s ospf routing component.

```tcl
setflag debug true -at sample n0/ospf
```

6.1.1.5 The objects overview and the wireless CSL

The following figure first lists up the objects in the simulations, i.e. the 6 wireless nodes, the channel, the tracker and the test object for trace files. Then it shows the sub components of a wireless node: the hello, the routing protocol ospf, the pktdispatcher and so on. Finally it presents the connections between the ports of all the sub components.
6.1.2 The simulation result for Four-Node’s scenario

The routing table for node 0 is as follows before the node 1 is turned off. Node 1 is the
nexthop for packets sending from node 0 to node 3.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Hop</th>
<th>Cost</th>
<th>Seq</th>
<th>Age</th>
<th>Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,2,0)(0,-1,0)</td>
<td>2-{0}</td>
<td>ROUTER,INTRA_AREA,cost(1),seq(INIT+2),age(6)</td>
<td>timeout:NaN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0,1,0)(0,-1,0)</td>
<td>1-{0}</td>
<td>ROUTER,INTRA_AREA,cost(1),seq(INIT+2),age(6)</td>
<td>timeout:NaN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0,3,0)(0,-1,0)</td>
<td>1-{0}</td>
<td>ROUTER,INTRA_AREA,cost(2),seq(INIT+1),age(7)</td>
<td>timeout:NaN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After the node 1 turns off, the routing table changes and node 2 becomes the nexthop for
packets sending from node 0 to node 3.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Hop</th>
<th>Cost</th>
<th>Seq</th>
<th>Age</th>
<th>Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,2,0)(0,-1,0)</td>
<td>2-{0}</td>
<td>ROUTER,INTRA_AREA,cost(1),seq(INIT+3),age(2)</td>
<td>timeout:NaN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0,3,0)(0,-1,0)</td>
<td>2-{0}</td>
<td>ROUTER,INTRA_AREA,cost(2),seq(INIT+2),age(2)</td>
<td>timeout:NaN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The figure below shows the throughput result of the four nodes’ simulation when a
FirstInFirstOut (FIFO) queue is used. The cylinder-like patterns are obvious and the
throughput is not stable.
The data and control traffics share the same queue after FIFO principle, when the data sending rate is high the collisions are high too on the mac_802_11 layer. Some of the control traffics like LSA and DBDESC are ACKable while no ACK exists for HELLO in this implementation. Since HELLO is broadcasted, it is hard to implement the ACK since the sender is not sure which ACKs are expected. Missing of HELLOs over DEAD_INTERVAL will turn the neighbor’s state to be DOWN, and therefore causes unnecessary linkbreak.

![Plot](example.plot)  
**Figure 28:** The throughput for the four nodes’ scenario, FIFO queue, queue capacity=40, packetsize=1500, packetrate=0.01

An alternative is use a PreemptPriorityQueue (PPQ) and the MACClassifier function. The data traffic in InetPacket has TypeOfService (TOS) value 0x00 and the control traffic has TOS value 0x01. TOS of the control traffics are defined in the java code. The TOS of the data traffics are defined by TCL script when data traffic is set up.

The next figure shows the throughput result of the four-node’s scenario when a PreemptPriorityQueue (PPQ) is used. When the node 1 is turned off at time 50, the throughput reduces first dramatically and then turns quickly back to the previous level when a new route is formed. The throughput lies relatively stable around 70-80kbytes per second compared to the figure above. We can still observe small cylinder-like patterns on the throughput figure both before and after the breakpoint. More explanation on it comes later in section 6.2 in the 6-node’s simulation where the phenomenon is more obvious.
A six nodes’ scenario is used here to run the simulation. As shown in the figure below, node 0, 1, 2, 3 are aligned and node 4, 5 are aligned and they form a multi-hop scenario. The two-way neighbor relationships are represented by the double-arrowed lines between the nodes. The simulation results will be discussed in the next subsections.

6.2.1 Simulation results for a six-node’s scenario
As shown on the figure below, the cylinder-like patterns are more obvious when we run the six nodes’ scenario although the PPQ is already used.
The tracing of the simulation informs that when a HELLO packet arrives at down port of the MAC_802_11, the high rated data traffics block out the HELLO packet and cause the neighbor to be unnecessary DOWN. The other related configurations for the traffic are the packetsize, packetrate and queue capacity. It is possible to change the data packetsize and packetperiod as follows in the TCL script.

```
# Set up traffic source/sink:
# send a packet of size 1500 bytes every .01 second
set packetSize_ 1500
set period_ .01
set traffic [java::new drcl.net.traffic.traffic_PacketTrain $packetSize_ $period_]
```

After some experiment, a change of packetperiod from 0.01s to 0.05s or larger will return a much stable throughput as shown below. Packetperiod 0.01s means a packetrate of 100 packets /s and packetperiod 0.05s means a packetrate of 20 packets /s. When the queue capacity is 40 packets /s, the packetrate of 20 packets /s is 50% of the capacity and has a good balance between control and data traffics.
6.2.2 MDR Selection results of the Six-Node’s scenario

The two figures below show the part of the simulation results of MDR Selection on node 0.

The first figure shows the result of an MDRNeighborChange event a hello received from neighbor 4 signaling the state change from down to twoway. The node 0’s RNL list is updated by node 4. The node 0 does not select itself as MDR since there are no neighbors with hops larger than MDRConstraint, and it does not select itself as Backup MDR since no 1 hop neighbor exists for Rmax. Node 0 selects node 4 as its MDR Parent and form an adjacency with it.
The second figure shows the result of an MDRNeighborChange event triggered by a hello received from its neighbor node 1 which has changed mdr_level, RNL and state since the last received hello. The MDRNeighborChange event starts the MDR Selection process by first finding the Rmax among its twoway neighbors and calculates the NCM matrix. The LCM and hops are calculated, since none of the twoway neighbors’ hops are larger than MDRConstraints, the node 0 does not choose itself as MDR, and it select node 1 as its parent. It continues to select Backup MDR. The LCM2 and hops2 are calculated. Since there are no two disjoint paths for node 4, node 0 selects itself as Backup MDR. It selects node 4 as its MDR Parent since node 4 is an existing adjacent node. No Backup MDR Parent is chosen since node 0 is not MDR Other. The final result is mdr_level=1 and mdr_parent=4.

The simulation result shows that it is not necessary that the node with highest triplet value will always be chosen as MDR Parent. Although a neighbor changes to be the Rmax among the neighbors, the calculating router will not break the existing adjacency and form a new one with the neighbor. A tendency to keep the existing adjacency is strong in the algorithm.
Figure 34: Result of MDRNeighborChange event on node 0 when neighbor node 1 changes state, mdr level and RNL list

```
****Router 0 nbr:1 mdr_level changed 0 -> 2
****Router 0 nbr:1 rnl changed null -> [4, 5]

****Router: 0 nbr:1 state changed 0 -> 3
OSPF-MDR:MDRNeighborChange, RNL=[4, 1]
Rmax=1 for router 0
NRM[0][0]=NCR(4,4)=NCR(4,4)=0
NRM[0][1]=NCR(4,1)=NCR(1,4)=0
NRM[1][0]=NCR(1,4)=NCR(4,1)=1
NRM[1][1]=NCR(1,1)=NCR(1,1)=0

LCM[0][0]=LCM(4,4)=99999 NRM=0 compareTriplet=0
LCM[0][1]=LCM(4,1)=1 NRM=1 compareTriplet=0
LCM[1][0]=LCM(1,4)=1 NRM=1 compareTriplet=0
LCM[1][1]=LCM(1,1)=99999 NRM=0 compareTriplet=0

Rmax 1 hops 0
Node 1 index=1 popped from the FIFO queue!
Node 1 is added to the FIFO queue. 1 hops to Rmax. Parent node 1
Node 4 index=0 popped from the FIFO queue!

LCM[0][0]=LCM(4,4)=99999 NRM 0 LCM99999
LCM[0][1]=LCM(4,1)=2 NRM 1 LCM1
LCM[1][0]=LCM(1,4)=0 NRM 1 LCM1
LCM[1][1]=LCM(1,1)=99999 NRM 0 LCM99999
Number of Rmax:hopNeighbors: 1
Subtree: 0 size 1
No node-disjoint paths for this node: 4
BackupMDRSelection - Router: 0 MDRLevel=1
DepNeighbor: node 1 mdr_level=2
Router: mdr_level=1 one of its adjacency MDR neighbors=4 is chosen as mdr_parent
Backup MDRParent Selection - Router:0 is not necessary since mdr_level=1 AdjConn
Router: 0 mdr_level=1 mdr_parent=4 mdr_parent=-1
Router: 0 mdr_parent: 4 is adjacent
```

Figure 34: Result of MDRNeighborChange event on node 0 when neighbor node 1 changes state, mdr level and RNL list
7 Conclusions
The chapter presents a summary of the thesis and a description on expected future works.

7.1 Thesis summary
The thesis studies routing protocols for MANETs. It categorized the routing protocols into MANETs-only protocols and wireless extension to OSPF. The MANETs-only protocols are represented by the reactive protocol AODV and the proactive protocol OLSR. Both of them are studied with wireless network scenarios. Then the two proposals for wireless extension to OSPF, i.e. OSPF-MDR and OSPF-MPR are studied and compared. The thesis focuses on the OSPF-MDR proposal in further writing and presents OSPF-MDR’s modification on the OSPF data structures in details and a wireless network scenario is analyzed to describe the MDR Selection process.

Then it presents the implementation of OSPF-MDR in J-Sim network simulation environment. During the implementation, the legacy OSPF protocol is studied and compared to the wireless extension. The extension to Interface, Neighbor and Hello data structures, the Interface State Machine, Neighbor State Machine and adjacency forming for MANET are realized in the J-Sim environment. The MDR Selection algorithm is the heaviest programming part of the thesis, while many efforts are also done to understand the legacy implementation before making the extension and to configure the simulations. Some modifications to the data structure other than that is specified in [OSPFMDR] are also noted during the implementation.

Two wireless multi-hop network scenarios are constructed and simulated using the new OSPF-MDR routing protocol in J-Sim. The simulations present the result of routing table before and after the linkbreak, the result of the MDR selection algorithm and the throughput. The throughputs are simulated and analysed under different configurations to reduce unstability shown as cylinder-like patterns.

7.2 Future works
The OSPF-MDR implementation is not completed yet. The MDR Selection algorithm should be tested with more network scenarios, and performance optimization is needed. The differential Hello is not completely implemented and the modification on adjacency forming, flooding of LSA and routing table are not completely implemented in this thesis. The configuration interface type in the TCL script does not work as expected and the bug should be solved to allow simulation on different interface types.

A higher ambition will be to complete the OSPF implementation for all interface types including MANET, so that a hybrid network scenario with both wired and wireless networks can be simulated and analyzed.

It will also be interesting to implement the other wireless extension proposal OSPF-MPR, and then compare the simulation results from both proposals.
References


Appendix A

TCL simulation code for a four nodes’ scenario

# four node scenario
# #
# #
# #     /-- n1 --n3
# n0 -- | /      /
#    \
# -- n2 /
# #

cd [mkdir -q drcl.comp.Component /example]

# create the topology
puts "create topology...

# create node 0
puts "create node 0"
set node0 [mkdir drcl.comp.Component n0]

# create necessary directories
mkdir drcl.inet.mac.LL          ll
mkdir drcl.inet.mac.ARP         arp
mkdir drcl.inet.core.queue.PreemptPriorityQueue    queue
mkdir drcl.inet.mac.Mac_802_11  mac
mkdir drcl.inet.mac.WirelessPhy  phy
mkdir drcl.inet.mac.FreeSpaceModel  propagation
mkdir drcl.inet.mac.MobilityModel mobility
mkdir drcl.inet.core.Hello      hello
mkdir drcl.inet.protocol.ospfmdr.OSPF ospfmdr
set PD [mkdir drcl.inet.core.PktDispatcher pktdispatcher]
set RT [mkdir drcl.inet.core.RT rt]
set ID [mkdir drcl.inet.core.Identity id]
SPD bind SRT
SPD bind SID

! queue setLevels 2
! queue setClassifier [java::new drcl.inet.mac.MacPktClassifier]

# enable route_back flag at PktDispatcher
! pktdispatcher setRouteBackEnabled true

# connect components in node 0
puts "connect components in node 0"
connect phy/.mobility@   -and mobility/.query@
connect phy/.propagation@ -and propagation/.query@
connect mac/down@ -and phy/up@
connect mac/up@ -and queue/output@
connect ll/.mac@ -and mac/.linklayer@
connect ll/down@ -and queue/output@
connect ll/.arp@ -and arp/.arp@
connect -c pktdispatcher/0@/down -and ll/up@
connect -c ospfmdr/down@ -and pktdispatcher/698@/up
connect ospfmdr/.service_rt@ -and rt/.service_rt@
connect ospfmdr/.service_id@ -and id/.service_id@
connect ospfmdr/.service_if@ -and hello/.service_if@
connect hello/.service_id@ -and id/.service_id@
connect mac/.linkbroken@ -and ospfmdr/.linkbroken@

! arp setAddresses 0 0
! ll setAddresses 0 0
! mac setMacAddress 0
! phy setNid 0
! mobility setNid 0
! id setDefaultID 0
! hello setStaticEnabled true
! hello _setInterfaceInfo 0 [java::new drcl.inet.data.InterfaceInfo 0 -8]

# maxX maxY minX minY dX dY dZ
! mobility setTopologyParameters 2000.0 0.0 2000.0 0.0 300.0 300.0 10.0
! mobility setPosition 0.0 0.0 200.0 0.0

! mac disable_PSM
! arp setBypassARP true

! queue setMode "packet"
! queue setCapacity 40

cd ..

# create node 1
puts "create node 1"
set node1 [mkdir drcl.comp.Component n1]
cd n1

mkdir drcl.inet.mac.LL ll
mkdir drcl.inet.mac.ARP arp
mkdir drcl.inet.core.queue.PreemptPriorityQueue queue
mkdir drcl.inet.mac.Mac_802_11 mac
mkdir drcl.inet.mac.WirelessPhy phy
mkdir drcl.inet.mac.FreeSpaceModel propagation
mkdir drcl.inet.mac.MobilityModel mobility
mkdir drcl.inet.core.Hello hello
mkdir drcl.inet.protocol.ospfmdr.OSPF ospfmdr
set PD [mkdir drcl.inet.core.PktDispatcher pktdispatcher]
set RT [mkdir drcl.inet.core.RT rt]
set ID [mkdir drcl.inet.core.Identity id]
$PD bind $RT
$PD bind $ID

! queue setLevels 2
! queue setClassifier [java::new drcl.inet.mac.MacPktClassifier]

#enable route_back flag at PktDispatcher
! pktdispatcher setRouteBackEnabled true

puts "connect components in node 1"
connect phy/.mobility@ -and mobility/.query@
connect phy/.propagation@ -and propagation/.query@
connect mac/down@ -and phy/up@
connect mac/up@ -and queue/output@
connect ll/.mac@ -and mac/.linklayer@
connect ll/down@ -and queue/up@
connect ll/.arp@ -and arp/.arp@
connect -c pktdispatcher/0@down -and ll/up@
connect -c ospfmdr/down@ -and pktdispatcher/698@up
connect ospfmdr/.service_rt@ -and rt/.service_rt@
connect ospfmdr/.service_id@ -and id/.service_id@
connect ospfmdr/.service_if@ -and hello/.service_if@
connect hello/.service_id@ -and id/.service_id@
connect mac/.linkbroken@ -and ospfmdr/.linkbroken@
! arp setAddresses 1 1
! ll setAddresses 1 1
! phy setNid 1
! mac setMacAddress 1
! mobility setNid 1
! id setDefaultID 1
! hello setStaticEnabled true
! hello _setInterfaceInfo 0 [java::new drlc.inet.data.InterfaceInfo 1 -8]

! mobility setTopologyParameters 2000.0 0.0 2000.0 0.0 300.0 300.0 10.0
! mobility setPosition 0.0 200.0 300.0 0.0

! mac disable_PSM
! arp setBypassARP true

! queue setMode "packet"
! queue setCapacity 40

cd ..

# create node 2
puts "create node 2"
set node2 [mkdir drc1.comp.Component n2]
cd n2

mkdir drc1.inet.mac.LL ll
mkdir drc1.inet.mac.ARP arp
mkdir drc1.inet.core.queue.PreemptPriorityQueue queue
mkdir drc1.inet.mac.Mac_802_11 mac
mkdir drc1.inet.mac.WirelessPhy phy
mkdir drc1.inet.mac.FreeSpaceModel propagation
mkdir drc1.inet.mac.MobilityModel mobility
mkdir drc1.inet.core.Hello hello
mkdir drc1.inet.protocol.ospfmdr.OSPF ospfmdr
set PD [mkdir drc1.inet.core.PktDispatcher pktdispatcher]
set RT [mkdir drc1.inet.core.RT rt]
set ID [mkdir drc1.inet.core.Identity id]
$PD bind $RT
$PD bind $ID

! queue setLevels 2
! queue setClassifier [java::new drc1.inet.mac.MacPktClassifier]

#enable route_back flag at PktDispatcher
! pktdispatcher setRouteBackEnabled true

puts "connect components in node 2"
connect phy/.mobility@ -and mobility/.query@
connect phy/.propagation@ -and propagation/.query@
connect mac/down@ -and phy/up@
connect mac/up@ -and queue/output@
connect ll/.mac@ -and mac/.linklayer@
connect ll/down@ -and queue/up@
connect ll/.arp@ -and arp/.arp@
connect -c pktdispatcher/0@down -and ll/up@
connect -c ospfmdr/down@ -and pktdispatcher/698@up
connect ospfmdr/.service_rt@ -and rt/.service_rt@
connect ospfmdr/.service_id@ -and id/.service_id@
connect ospfmdr/.service_if@ -and hello/.service_if@
connect hello/.service_id@ -and id/.service_id@
connect mac/.linkbroken@ -and ospfmdr/.linkbroken@

! arp setAddresses 2 2
! II setAddresses 2 2
! phy setNid 2
! mac setMacAddress 2
! mobility setNid 2
! id setDefaultID 2
! hello setStaticEnabled true
! hello/setInterfaceInfo 0 [java::new drc1.inet.data.InterfaceInfo 2 -8]

#       maxX maxY minX minY dX dY dZ
! mobility setTopologyParameters 2000.0 0.0 2000.0 0.0 300.0 300.0 10.0
! mobility setPosition 0.0 200.0 100.0 0.0

! mac disable_PSM
! arp setBypassARP true

! queue setMode "packet"
! queue setCapacity 40

cd ..

# create node 3
puts "create node 3"
set node3 [mkdir drc1.comp.Component n3]
cd n3

mkdir drc1.inet.mac.LL       ll
mkdir drc1.inet.mac.ARP      arp
mkdir drc1.inet.core.queue.PriorityQueue queue
mkdir drc1.inet.mac.Mac_802_11 mac
mkdir drc1.inet.mac.WirelessPhy phy
mkdir drc1.inet.mac.FreeSpaceModel propagation
mkdir drc1.inet.mac.MobilityModel mobility
mkdir drc1.inet.core.Hello   hello
mkdir drc1.inet.protocol.ospfmdr.OSPf ospfmdr
set PD [mkdir drc1.inet.core.PktDispatcher pktdispatcher]
set RT [mkdir drc1.inet.core.RT rt]
set ID [mkdir drc1.inet.core.Identity id]
$PD bind $RT
$PD bind $ID

! queue setLevels 2
! queue setClassifier [java::new drc1.inet.core.DSTClass] [java::new drc1.inet.mac.DSTClass]

#enable route_back flag at PktDispatcher
! pktdispatcher setRouteBackEnabled true

puts "connect components in node 3"
connect phy/.mobility@ -and mobility/.query@
connect phy/.propagation@ -and propagation/.query@
connect mac/down@ -and phy/up@
connect mac/up@ -and queue/output@
connect ll/.mac@ -and mac/.linklayer@
connect ll/down@ -and queue/up@
connect ll/.arp@ -and arp/.arp@
connect -c pktdispatcher/0@down -and ll/up@
connect -c ospfmdr/down@ -and pktdispatcher/698@up
connect ospfmdr/service_rt@ -and rt/service_rt@
connect ospfmdr/service_id@ -and id/service_id@
connect ospfmdr/service_if@ -and hello/service_if@
connect hello/service_id@ -and id/service_id@
connect mac/.linkbroken@ -and ospfmdr/.linkbroken@

! arp setAddresses 3 3
! Il setAddresses 3 3
! phy setNid 3
! mac setMacAddress 3
! mobility setNid 3
! id setDefaultID 3
! hello setStaticEnabled true
! hello/setInterfaceInfo 0 [java::new drcl.inet.data.InterfaceInfo 3 -8]

# max X  maxY  minX  minY  dX   dY   dZ
! mobility setTopologyParameters 2000.0 0.0 2000.0 0.0 300.0 300.0 10.0
! mobility setPosition 0.0 300.0 300.0 0.0

! mac disable_PSM
! arp setBypassARP true

! queue setMode "packet"
! queue setCapacity 40

cd..

puts "create channel"

mkdir drcl.inet.mac.Channel channel
mkdir drcl.inet.mac.NodePositionTracker tracker
# maxX  minX  maxY  minY  dX   dY
! tracker setGrid 2000.0 0.0 2000.0 0.0 300.0 300.0

connect channel/.tracker@ -and tracker/.channel@
connect /example/n0/mobility/.report@ -and /example/tracker/.node@
connect /example/n1/mobility/.report@ -and /example/tracker/.node@
connect /example/n2/mobility/.report@ -and /example/tracker/.node@
connect /example/n3/mobility/.report@ -and /example/tracker/.node@

! channel setCapacity 4

connect n0/phy/down@ -to channel/.node@
! channel attachPort 0 [! /example/n0/phy getPort .channel]

connect n1/phy/down@ -to channel/.node@
! channel attachPort 1 [! /example/n1/phy getPort .channel]

connect n2/phy/down@ -to channel/.node@
! channel attachPort 2 [! /example/n2/phy getPort .channel]

connect n3/phy/down@ -to channel/.node@
! channel attachPort 3 [! /example/n3/phy getPort .channel]

# Source and sink
puts "setup source and sink..."

set_dst 3
set_src 0
set traffic [java::new drcl.net.traffic.traffic_PacketTrain 1500 0.01]
mkdir [java::call drcl.inet.InetUtil createTrafficSource $traffic 0 $dst 0] n$_src/cbr
connect -c n$_src/cbr/down@ -and n$_src/pktdispatcher/17@up

puts "Set up TrafficMonitor & Plotter..."
set plot_ [mkdir drcl.comp.tool.Plotter .plot]
set tm_ [mkdir drcl.net.tool.TrafficMonitor .tm]
set cfile_ [mkdir drcl.comp.io.FileComponent .cfile]
$cfile_ open "bytecount.txt"
connect -c n3/pktdispatcher/17@up -to $tm_/in@
connect -c $tm_/bytecount@ -to $plot_/0@1
connect -c $tm_/bytecount@ -to $cfile_/in@

########################################################################Start simulation########################################################################
puts "simulation begins..."
set sim [attach_simulator .]
#setflag debug true -at \{DEBUG_SAMPLE DEBUG_NEIGHBOR DEBUG_SEND DEBUG_SPF
DEBUG_REFRESH DEBUG_LSA DEBUG_ACK DEBUG_TIMEOUT DEBUG_DETAIL\} n*/ospf
#setflag debug true -at \{DEBUG_NEIGHBOR\} n0/ospf
run .
script \{rt n0/rt\} -at 45 -on $sim
script \{cat n0/rt\} -at 45 -on $sim
script \{cat n*/ospfmdr\} -at 45 -on $sim

script \{stop n1\} -at 50 -on $sim
script \{rt n0/rt\} -at 56 -on $sim
script \{cat n0/rt\} -at 56 -on $sim
script \{cat n*/ospfmdr\} -at 56 -on $sim
script \{rt n0/rt\} -at 100 -on $sim
$sim stopAt 100

TCL simulation code for a six nodes’ scenario
# six node scenario
#
#n4--n5
# | \
# | / |
# | / |
# | / |
# n0--n1--n2--n3

cd [mkdir -q drcl.comp.Component /example]

# create the topology
puts "create topology..."

# create node 0
puts "create node 0"
set node0 [mkdir drcl.comp.Component n0]
cd n0
mkdir drcl.inet.mac.LL              ll
mkdir drcl.inet.mac.ARP             arp
mkdir drcl.inet.core.queue.PreemptPriorityQueue    queue
mkdir drcl.inet.mac.Mac_802_11 mac
mkdir drcl.inet.mac.WirelessPhy phy
mkdir drcl.inet.mac.FreeSpaceModel propagation
mkdir drcl.inet.mac.MobilityModel mobility
mkdir drcl.inet.core.Hello hello
mkdir drcl.inet.protocol.ospfmdr.OSPF ospfmdr
set PD [mkdir drcl.inet.core.PktDispatcher pktdispatcher]
set RT [mkdir drcl.inet.core.RT rt]
set ID [mkdir drcl.inet.core.Identity id]
$PD bind $RT
$PD bind $ID
! queue setLevels 2
! queue setClassifier [java::new drcl.inet.mac.MacPktClassifier]

# enable route_back flag at PktDispatcher
! pktdispatcher setRouteBackEnabled true

# connect components in node 0
puts "connect components in node 0"
connect phy/.mobility@ -and mobility/.query@
connect phy/.propagation@ -and propagation/.query@
connect mac/down@ -and phy/up@
connect mac/up@ -and queue/output@
connect ll/.mac@ -and mac/.linklayer@
connect ll/down@ -and queue/up@
connect ll/.arp@ -and arp/.arp@
connect -c pktdispatcher/0@down -and ll/up@
connect -c ospfmdr/down@ -and pktdispatcher/698@up
connect ospfmdr/.service_rt@ -and rt/service_rt@
connect ospfmdr/.service_id@ -and id/service_id@
connect ospfmdr/.service_if@ -and hello/service_if@
connect hello/service_id@ -and id/service_id@
#connect mac/.linkbroken@ -and ospfmdr/.linkbroken@

! arp setAddresses 0 0
! ll setAddresses 0 0
! mac setMacAddress 0
! phy setNid 0
! mobility setNid 0
! id setDefaultID 0
! hello setStaticEnabled true
! hello_setInterfaceInfo 0 [java::new drcl.inet.data.InterfaceInfo 0 -8]

# mobility setTopologyParameters 2000.0 0.0 2000.0 0.0 300.0 300.0 10.0
# mobility setPosition 0.0 0.0 0.0

! mac disable_PSM
! arp setBypassARP true

! queue setMode "packet"
! queue setCapacity 40

# Enable Debug of node 0
#! ospfmdr setDebugEnabled true
#! ospfmdr setDebugEnabledAt true 0

cd ..

puts "create node 1"
set node1 [mkdir drcl.comp.Component n1]
cd n1

mkdir drcl.inet.mac.LL       ll
mkdir drcl.inet.mac.ARP      arp
mkdir drcl.inet.core.queue.PriorityQueue queue
mkdir drcl.inet.mac.Mac_802_11    mac
mkdir drcl.inet.mac.WirelessPhy phy
mkdir drcl.inet.mac.FreeSpaceModel propagation
mkdir drcl.inet.mac.MobilityModel mobility
mkdir drcl.inet.core.Hello hello
mkdir drcl.inet.protocol.ospfmdl.OSPF ospfmdl
set PD [mkdir drcl.inet.core.PktDispatcher pktdispatcher]
set RT [mkdir drcl.inet.core.RT rt]
set ID [mkdir drcl.inet.core.Identity id]
$PD bind $RT
$PD bind $ID

! queue setLevels 2
! queue setClassifier [java::new drcl.inet.mac.MacPktClassifier]

#enable route_back flag at PktDispatcher
! pktdispatcher setRouteBackEnabled true

puts "connect components in node 1"
connect phy/.mobility@ -and mobility/.query@
connect phy/.propagation@ -and propagation/.query@
connect mac/down@ -and phy/up@
connect mac/up@ -and queue/output@
connect ll/.mac@ -and mac/.linklayer@
connect ll/down@ -and queue/up@
connect ll/.arp@ -and arp/.arp@
connect -c pktdispatcher/0@down -and ll/up@
connect -c ospfmdl/down@ -and pktdispatcher/698@up
connect ospfmdl/.service_rt@ -and rt/.service_rt@
connect ospfmdl/.service_id@ -and id/.service_id@
connect hello/.service_if@ -and hello/.service_if@
#connect mac/.linkbroken@ -and ospfmdl/.linkbroken@

! arp setAddresses 1 1
! ll setAddresses 1 1
! phy setNid 1
! mac setMacAddress 1
! mobility setNid 1
! id setDefaultID 1
! hello setStaticEnabled true
! hello/.setInterfaceInfo 0 [java::new drcl.inet.data.InterfaceInfo 1 -8]

# mobility setTopologyParameters 2000.0 0.0 2000.0 0.0 300.0 300.0 10.0
! mobility setPosition 0.0 200.0 0.0 0.0

! macro disable_PSM
! arp setBypassARP true

! queue setMode "packet"
! queue setCapacity 40
cd ..

# create node 2
puts "create node 2"
set node2 [mkdir drcl.comp.Component n2]
cd n2

mkdir drcl.inet.mac.LL    ll
mkdir drcl.inet.mac.ARP   arp
mkdir drcl.inet.core.Queue.PreeemptPriorityQueue  queue
mkdir drcl.inet.mac.Mac_802_11    mac
mkdir drcl.inet.mac.WirelessPhy    phy
mkdir drcl.inet.mac.FreeSpaceModel  propagation
mkdir drcl.inet.mac.MobilityModel  mobility
mkdir drcl.inet.core.Hello    hello
mkdir drcl.inet.protocol.ospfmdr.OSPF  ospfmdr
set PD [mkdir drcl.inet.core.PktDispatcher      pktdispatcher]
set RT [mkdir drcl.inet.core.RT                 rt           ]
set ID [mkdir drcl.inet.core.Identity           id           ]
$PD bind $RT
$PD bind $ID

#trace the different wireless layers, receive packet
#setflag trace true ospf/down@
#setflag trace true pktdispatcher/698@up
#setflag trace true pktdispatcher/0@down
#setflag trace true ll/up@
#setflag trace true ll/.mac@
#*setflag trace true mac/.linklayer@
#*setflag trace true mac/down@
#setflag trace true phy/up@

! queue setLevels 2
! queue setClassifier [java::new drcl.inet.mac.MacPktClassifier]

#enable route_back flag at PktDispatcher
! pktdispatcher setRouteBackEnabled true

puts "connect components in node 2"
connect phy/.mobility@    -and mobility/.query@
connect phy/.propagation@ -and propagation/.query@
connect mac/down@ -and phy/up@
connect mac/up@ -and queue/output@
connect ll/.mac@ -and mac/.linklayer@
connect ll/down@ -and queue/up@
connect ll/.arp@ -and arp.arp@
connect -c pktdispatcher/0@down -and ll/up@
connect -c ospfmdr/down@ -and pktdispatcher/698@up
connect ospfmdr/.service_rt@ -and rt/.service_rt@
connect ospfmdr/.service_id@ -and id/.service_id@
connect ospfmdr/.service_if@ -and hello/.service_if@
connect hello/.service_id@ -and id/.service_id@
#connect mac/.linkbroken@ -and ospfmdr/.linkbroken@

! arp setAddresses 2 2
! ll setAddresses 2 2
! phy setNid 2
! mac setMacAddress 2
! mobility setNid 2
! id setDefaultID 2
! hello setStaticEnabled true
! hello _setInterfaceInfo 0 [java::new drcl.inet.data.InterfaceInfo 2 -8]

# mobility setTopologyParameters 2000.0 0.0 2000.0 0.0 300.0 300.0 10.0
! mobility setPosition 0.0 400.0 0.0 0.0

! mac disable_PSM
! arp setBypassARP true

! queue setMode "packet"
! queue setCapacity 40

cd ..

# create node 3
puts "create node 3"
set node3 [mkdir drcl.comp.Component n3]
cd n3

mkdir drcl.inet.mac.LL   ll
mkdir drcl.inet.mac.ARP  arp
mkdir drcl.inet.core.queue.PreemptPriorityQueue queue
mkdir drcl.inet.mac.Mac_802_11  mac
mkdir drcl.inet.mac.WirelessPhy phy
mkdir drcl.inet.mac.FreeSpaceModel propagation
mkdir drcl.inet.mac.MobilityModel mobility
mkdir drcl.inet.core.Hello hello
mkdir drcl.inet.protocol.ospfmdr.OSPF ospfmdr
set PD [mkdir drcl.inet.core.PktDispatcher pktdispatcher]
set RT [mkdir drcl.inet.core.RT rt]
set ID [mkdir drcl.inet.core.Identity id]
$PD bind $RT
$PD bind $ID

#trace the different wireless layers, send packet
#setflag trace true ospfmdr/down@
#setflag trace true pktdispatcher/698@up
#setflag trace true pktdispatcher/0@down
#setflag trace true ll/up@
#setflag trace true ll/down@
#setflag trace true queue/up@
#setflag trace true queue/output@
#setflag trace true mac/up@
#*setflag trace true mac/down@
#*setflag trace true phy/up@

! queue setLevels 2
! queue setClassifier [java::new drcl.inet.mac.MacPktClassifier]

#enable route_back flag at PktDispatcher
! pktdispatcher setRouteBackEnabled true

puts "connect components in node 3"
connect phy/.mobility@ -and mobility/.query@
connect phy/.propagation@ -and propagation/.query@
connect mac/down@ -and phy/up@
connect mac/up@ -and queue/output@
connect ll/.mac@ -and mac/.linklayer@
connect ll/down@ -and queue/up@
connect ll/arp@ -and arp/.arp@
connect -c pktdispatcher/0@down -and ll/up@
connect -c ospfmdr/down@ -and pktdispatcher/698@up
connect ospfmdr/service_rt@ -and rt/service_rt@
connect ospfmdr/service_id@ -and id/service_id@
connect ospfmdr/service_if@ -and hello/service_if@
connect hello/service_id@ -and id/service_id@
#connect mac/.linkbroken@ -and ospfmdr/.linkbroken@

! arp setAddresses 3 3
! ll setAddresses 3 3
! phy setNid 3
! mac setMacAddress 3
! mobility setNid 3
! id setDefaultID 3
! hello setStaticEnabled true
! hello_setInterfaceInfo 0 [java::new drcl.inet.data.InterfaceInfo 3 -8]

maxX maxY minX minY dX dY dZ
! mobility setTopologyParameters 2000.0 0.0 2000.0 0.0 300.0 300.0 10.0
! mobility setPosition 0.0 600.0 0.0 0.0

! mac disable_PSM
! arp setBypassARP true

! queue setMode "packet"
! queue setCapacity 40

cd ..

# create node 4
puts "create node 4"
set node4 [mkdir drcl.comp.Component n4]
cd n4

mkdir drcl.inet.mac.LL ll
mkdir drcl.inet.mac.ARP arp
mkdir drcl.inet.core.queue.PreemptPriorityQueue queue
mkdir drcl.inet.mac.Mac_802_11 mac
mkdir drcl.inet.mac.WirelessPhy phy
mkdir drcl.inet.mac.FreeSpaceModel propagation
mkdir drcl.inet.mac.MobilityModel mobility
mkdir drcl.inet.core.Hello hello
mkdir drcl.inet.protocol.ospfmdr.OSPF ospfmdr
set PD [mkdir drcl.inet.core.PktDispatcher pktdispatcher]
set RT [mkdir drcl.inet.core.RT rt]
set ID [mkdir drcl.inet.core.Identity id]
SPD bind SRT
SPD bind SID

! queue setLevels 2
! queue setClassifier [java::new drcl.inet.mac.MacPktClassifier]

#enable route_back flag at PktDispatcher
! pktdispatcher setRouteBackEnabled true

puts "connect components in node 4"
connect phy/.mobility@ -and mobility/.query@
connect phy/.propagation@ -and propagation/.query@
connect mac/down@ -and phy/up@
connect mac/up@ -and queue/output@
connect ll/.mac@ -and mac/.linklayer@
connect ll/down@ -and queue/up@
connect ll/arp@ -and arp/.arp@
connect -c pktdispatcher/0/down -and ll/up@
connect -c ospfmdr/down@ -and pktdispatcher/698/up
connect ospfmdr/.service_rt@ -and rt/service_rt@
connect ospfmdr/.service_id@ -and id/.service_id@
connect ospfmdr/.service_if@ -and hello/service_if@
connect hello/.service_id@ -and id/service_id@
#connect mac/.linkbroken@ -and ospfmdr/.linkbroken@

! arp setAddresses 4 4
! ll setAddresses 4 4
! phy setNid 4
! mac setMacAddress 4
! mobility setNid 4
! id setDefaultID 4
! hello setStaticEnabled true
! hello_setInterfaceInfo 0 [java::new drcl.inet.data.InterfaceInfo 4 -8]

# mobility setTopologyParameters 2000.0 0.0 2000.0 0.0 300.0 300.0 10.0
# mobility setPosition 0.0 100.0 200.0 0.0

! mac disable_PSM
! arp setBypassARP true

! queue setMode "packet"
! queue setCapacity 40

cd ..

# create node 5
puts "create node 5"
set node5 [mkdir drcl.comp.Component n5]
cd n5

mkdir drcl.inet.mac.LL ll
mkdir drcl.inet.mac.ARP arp
mkdir drcl.inet.core.queue.PriorityQueue queue
mkdir drcl.inet.mac.Mac_802_11 mac
mkdir drcl.inet.mac.WirelessPhy phy
mkdir drcl.inet.mac.FreeSpaceModel propagation
mkdir drcl.inet.mac.MobilityModel mobility
mkdir drcl.inet.core.Hello hello
mkdir drcl.inet.protocol.ospfmdr.OSPf ospfmdr
set PD [mkdir drcl.inet.core.PktDispatcher pktdispatcher]
set RT [mkdir drcl.inet.core.RT rt]
set ID [mkdir drcl.inet.core.Identity id]
SPD bind SRT
SPD bind SID

! queue setLevels 2
! queue setClassifier [java::new drcl.inet.mac.MacPktClassifier]

#enable route_back flag at PktDispatcher
! pktdispatcher setRouteBackEnabled true

puts "connect components in node 5"
connect phy/.mobility@ -and mobility/.query@
connect phy/.propagation@ -and propagation/.query@
connect mac/down@ -and phy/up@
connect mac/up@ -and queue/output@
connect ll/.mac@ -and mac/.linklayer@
connect ll/down@ -and queue/up@
connect ll/.arp@ -and arp/.arp@
connect -c pktdispatcher/0@down -and ll/up@
connect -c ospfmdr/down@ -and pktdispatcher/698@up
connect ospfmdr/.service_rt@ -and rt/.service_rt@
connect ospfmdr/.service_id@ -and id/.service_id@
connect ospfmdr/.service_if@ -and hello/.service_if@
connect hello/.service_id@ -and id/.service_id@
#connect mac/.linkbroken@ -and ospfmdr/.linkbroken@

! arp setAddresses 5 5
! ll setAddresses 5 5
! phy setNid 5
! mac setMacAddress 5
! mobility setNid 5
! id setDefaultID 5
! hello setStaticEnabled true
! hello_setInterfaceInfo 0 [java::new drcl.inet.data.InterfaceInfo 5 -8]

# maxX maxY minX minY dX dY dZ
! mobility setTopologyParameters 2000.0 0.0 2000.0 0.0 300.0 300.0 10.0
! mobility setPosition 0.0 300.0 200.0 0.0

! mac disable_PSM
! arp setBypassARP true

! queue setMode "packet"
! queue setCapacity 40

cd ..

puts "create channel"

mkdir drc1.inet.mac.Channel channel
mkdir drc1.inet.mac.NodePositionTracker tracker
# maxX minX maxY minY dX dY
! tracker setGrid 2000.0 0.0 2000.0 0.0 300.0 300.0

connect channel/.tracker@ -and tracker/.channel@
connect /example/n0/mobility/.report@ -and /example/tracker/.node@
connect /example/n1/mobility/.report@ -and /example/tracker/.node@
connect /example/n2/mobility/.report@ -and /example/tracker/.node@
connect /example/n3/mobility/.report@ -and /example/tracker/.node@
connect /example/n4/mobility/.report@ -and /example/tracker/.node@
connect /example/n5/mobility/.report@ -and /example/tracker/.node@

! channel setCapacity 6

connect n0/phy/down@ -to channel/.node@
! channel attachPort 0 [/example/n0/phy getPort .channel]

connect n1/phy/down@ -to channel/.node@
! channel attachPort 1 [! /example/n1/phy getPort .channel]
connect n2/phy/down@ -to channel/.node@
! channel attachPort 2 [! /example/n2/phy getPort .channel]
connect n3/phy/down@ -to channel/.node@
! channel attachPort 3 [! /example/n3/phy getPort .channel]
connect n4/phy/down@ -to channel/.node@
! channel attachPort 4 [! /example/n4/phy getPort .channel]
connect n5/phy/down@ -to channel/.node@
! channel attachPort 5 [! /example/n5/phy getPort .channel]

++++++++++++++++++++++Source and sink+++++++++++++++++++++++++++++++++++++
pus "setup source and sink..."
set _dst 3
set _src 0

# Set up traffic source/sink:
# send a packet of size 512 bytes every .1 second
set packetSize_ 1500
set period_ .01
set traffic [java::new drcl.net.traffic.traffic_PacketTrain $packetSize_ $period_]
mkdir [java::call drcl.inet.InetUtil createTrafficSource $traffic 0 _dst 0] n$_src/cbr
connect -c n$_src/cbr/down@ -and n$_src/pktdispatcher/17@up

pus "Set up TrafficMonitor & Plotter..."
set plot_ [mkdir drcl.comp.tool.Plotter .plot]
set tm_ [mkdir drcl.net.tool.TrafficMonitor .tm]
set cfile_ [mkdir drcl.comp.io.FileComponent .cfile]
$cfile_ open "bytecount.txt"
connect -c n3/pktdispatcher/17@up -to $tm_/in@
connect -c $tm_/bytecount@ -to $plot_/0@1
connect -c $tm_/bytecount@ -to $cfile_/in@

++++++++++++++++++++++Start simulation+++++++++++++++++++++++
pus "simulation begins..."
set sim [attach_simulator .]
#setflag debug true -at {DEBUG_SAMPLE DEBUG_NEIGHBOR DEBUG_SEND DEBUG_SPF
DEBUG_REFRESH DEBUG_LSA DEBUG_ACK DEBUG_TIMEOUT DEBUG_DETAIL} n3/ospfmdr
#setflag debug true -at {DEBUG_NEIGHBOR DEBUG_SEND} n3/ospfmdr
run .

#script {rt n0/rt} -at 45 -on $sim
script {cat n0/rt} -at 45 -on $sim
script {cat n*/ospfmdr} -at 45 -on $sim
script {stop n1} -at 50 -on $sim
script {rt n0/rt} -at 56 -on $sim
script {cat n0/rt} -at 56 -on $sim
script {rt n0/rt} -at 100 -on $sim
$sim stopAt 100
Appendix B
Installation of simulation tools in a windows environment

1. download j-sim from http://www.j-sim.org , unpack it to e.g. c:\java\jsim-1.3
download j2sdk1.4.x from java.sun.com, install it to e.g. c:\java\j2sdk1.4.2
download ant from ant.apache.org, unpack it to e.g. c:\java\ant-1.6.2

2. edit the setcpath.bat, obs that there maybe a lot of java.exe in the system, like c:\windows\system32\java.exe, and the version maybe lower than 1.4.x, therefor we shall set the right java at the first place of the path variable.

@echo off
echo "for jdk, java version should be 1.4.x or over"
set JAVA_HOME=c:/java/j2sdk1.4.2
set PATH=%PATH%;%JAVA_HOME%\bin
echo "for jsim"
set J_SIM=c:/java/jsim-1.3
set script=%J_SIM%\jars/tcl.zip;%J_SIM%\jars/jython.jar
set classpath=.;%CLASSPATH%;%J_SIM%\classes;%script%
echo "for ant"
set ANT_HOME=c:/java/ant-1.6.2
set PATH=%PATH%;%ANT_HOME%\bin

3. run the setcpath.bat everytime before starting jsim.

Instead of running the bat file, one can also change the environment variable in Control panel->System->Advanced->Environment variable. Remember that bat file is valid only for the doswindow where bat is run, while the other method will be valid until you remove them from the control panel etc.

The result should be something like these:
C:\java\jsim-1.3>set j_sim
J_SIM=c:/java/jsim-1.3

C:\java\jsim-1.3>set java_home
JAVA_HOME=c:/java/j2sdk1.4.2

C:\java\jsim-1.3>set script
script=c:/java/jsim-1.3\jars/tcl.zip;c:/java/jsim-1.3\jars/jython.jar

C:\java\jsim-1.3>set classpath
classpath=.;c:/java/jsim-1.3\classes;c:/java/jsim-1.3\jars/tcl.zip;c:/java/jsim-1.3\jars/jython.jar

C:\java\jsim-1.3>set ant_home
ANT_HOME=c:/java/ant-1.6.2

4. compile the jsim, by running
C:\java\jsim-1.3>ant compile
Buildfile: build.xml

init:
[echo] 17-august-2004 11:35 PM

compile:
[javac] Compiling 514 source files to C:\java\jsim-1.3\classes
[copy] Copying 1 file to C:\java\jsim-1.3\classes\drl\ruv

BUILD SUCCESSFUL
Total time: 8 seconds

To remove all the compiled files, run ant clean
5. Run the jsim and get fun
C:\java\jsim-1.3\script\test>java drcl.ruv.System include.tcl

6. Something wrong?
if you get this exception:
Exception in thread "main" java.lang.UnsupportedClassVersionError: drcl/ruv/Syst	em (Unsupported major.minor version 48.0)

This means that the java version is lower than 1.4.x. To check the real version, you can type command
java -version

IF you search for java.exe, you may find quite a bunch, therefor it'll be necessary to get the right one in the
environment variable 'PATH'

7. Install netbeans
D:\ling\hf\tools\netbeans>netbeans-3_6-windows.exe -is:javahome c:\java\j2sdk1.4

Appendix C
The walk-through process of making extension to OSPF implementation
Suppose that $JSIM_HOME is the path where J-Sim is installed.

Phase 1: Preparation
Make a copy of $JSIM_HOME\src\drcl\inet\protocol\ospf, and place it in the same directory, rename it to be
$JSIM_HOME\src\drcl\inet\protocol\ospfmdr.
Create another new directory $JSIM_HOME\script\drcl\inet\ospfmdr for storing OSPF-MDR related tcl scripts.
Change the package line in every java file in the ospfmdr directory from
package drcl.inet.protocol.ospf;
to
package drcl.inet.protocol.ospfmdr;

Modify an existing ospf script test4.tcl to let it use the ospfmdr package, replace tekst “ospf.OSPF” with
“ospfmdr.OSPF”, test the script. The script should return the same result as with ospf package.

Modify an existing olsr script scenario1_6nodes.OLSR.tcl to let it use the ospfmdr package, replace tekst
“olsr.OLSR” to “ospfmdr.OSPF”, and then replace tekst “olsr” with “ospfmdr”, test the script. It will return
several errors when running the script and the next phase will focus on making the script to work.

Phase 2: Extention

Interface
OSPF_Interface class
OSPF class: adds type IF_MANET=5

Neighbor
OSPF_Neighbor class

Hello
OSPF_Hello class, OSPFMDR_LLS class, OSPFMDR_LLS_TLV class

Sending/Receiving Hello
OSPF class: _start( ), timeout( ), ospf_send_hello( ), dataArriveAtDownPort( ), Process_Hello_MANET( )
OSPF_Hello class
OSPF_Neighbor class
OSPF_Interface class

Interface State Machine
OSPF_Interface class MDRNeighborChange( )

Neighbor State Machine
OSPF class: _nbr_hello_received( ), _nbr_twoway_received( )

Tips on how to stop a component in J-Sim, adopted from the J-Sim discussion forum.
If the event source implements the _stop() interface, then you can
simply "stop" it at specific time as:

```
script {stop n1/source} -at <time> -on $sim
```

Another trick is to disable the transport or network layer component.
For example, the following command disables the network layer
component "pd". The result is that "pd" will not respond to any
incoming and outgoing packets and thus the entire node appears to be
"stopped":

```
setflag component false n1/csl/pd
```

Or the following command disables "udp" so that all the applications
above it appear to be "stopped":

```
setflag component false n1/udp
```

Appendix D
Javadoc for OSPF package of the OSPF-MDR implementation

**Class OSPF**

java.lang.Object
   ↳ drcl.DrlObj
      ↳ drcl.comp.Component
         ↳ drcl.net.Module
             ↳ drcl.inet.Protocol
                 ↳ drcl.inet.protocol.Routing
                    ↳ drcl.inet.protocol.ospfmdr.OSPF

All Implemented Interfaces:
drcl.comp.ActiveComponent, java.lang.Cloneable, drcl.inet.InetConstants, drcl.ObjectCloneable,

Direct Known Subclasses:
   OSPF_QoS

public class OSPF
extends drcl.inet.protocol.Routing
implements drcl.inet.protocol.UnicastRouting, drcl.comp.ActiveComponent
The package, drcl.inet.protocol.OSPF, implements the link-state routing protocol, OSPF v2, Open Shortest Path
First routing protocol and follow the RFC2328. In this implementation, we refered much from OSPF code in
GNU Zebra project, especially zebra 0.85.
The general data structure can refer to figure 9 and section 5 RFC2328 (If not specially mentioned, the following
sections, figures, and etc. can be found in RFC 2328)
See Also:
   OSPF_Area, OSPF_DBdesc, OSPF_Packet, OSPF_Interface, OSPF_LS_Database, OSPF_LSA,
   OSPF_LSA_Header, OSPF_LSack, OSPF_LSrequest, OSPF_LUpdate, OSPF_Neighbor,
   OSPF_TimeOut_EVT, Serialized Form

Nested Class Summary

Nested classes inherited from class drcl.comp.Component
| Field Summary |
|--------------|----------------------------------|
| protected java.util.Vector | **area_list** | The list of OSPF areas to which this router is attached |
| protected static int | **DATABASE** | A constant indicating a database description packet. |
| static boolean | **debug** |
| static int | **DEBUG ACK** | Debug level of handling LSA acknowledgement packets. |
| static int | **DEBUG DETAIL** | Debug level of detailed messages for other levels. |
| static int | **DEBUG LSA** | Debug level of handling LSA update packets. |
| static int | **DEBUG NEIGHBOR** | Debug level of neighbor state changes. |
| static int | **DEBUG REFRESH** | Debug level of LSA refresh events. |
| static int | **DEBUG SAMPLE** | Debug level to enable sampled debug messages. |
| static int | **DEBUG SEND** | Debug level of sending ospf packets. |
| static int | **DEBUG SPF** | Debug level of calculating SPF tree and installing route entries. |
| static int | **DEBUG TIMEOUT** | Debug level of handling timeout events. |
| static boolean | **debug2** |
| protected static int | **HELLO** | A constant indicating a hello packet. |
| protected static int | **LS_ACK** | A constant indicating a link state acknowledge packet. |
| protected static int | **LS_REQUEST** | A constant indicating a link state request packet. |
| protected static int | **LS_UPDATE** | A constant indicating a link state update packet. |
| protected java.util.Vector | **ospf_if_list** | The list of OSPF Interfaces to which this router is attached |
| protected int | **router_id** | OSPF router ID of this node |
| static int | **UDP PORT** | OSPF-MDR: Borrow the UDP port used by OLSR, assigned by IANA. |

**Fields inherited from class drcl.inet.protocol.Routing**

queryPort, rtconfigPort
Fields inherited from class `drcl.net.Module`
- `downPort`, `PortGroup_DOWN`, `PortGroup_UP`, `timerPort`, `upPort`

Fields inherited from class `drcl.comp.Component`
- `FLAG_COMPONENT_NOTIFICATION`, `FLAG_DEBUG_ENABLED`, `FLAG_DIRECT_OUTPUT_ENABLED`, `FLAG_ENABLED`, `FLAG_ERROR_ENABLED`, `FLAG_EVENT_ENABLED`, `FLAG_GARBAGE_DISPLAY_ENABLED`, `FLAG_GARBAGE_ENABLED`, `FLAG_PORT_NOTIFICATION`, `FLAG_STARTED`, `FLAG_STOPPED`, `FLAG_TRACE_ENABLED`, `FLAG_UNDEFINED_START`, `id`, `infoPort`, `locks`, `name`, `parent`, `PortGroup_DEFAULT_GROUP`, `PortGroup_EVENT`, `PortGroup_SERVICE`, `Root`, `Trace_DATA`, `Trace_SEND`

Fields inherited from interface `drcl.inet.InetConstants`

Constructor Summary
- `OSPF()` Constructor.

Method Summary
- `protected void _resume()` Resumes the active component OLSR.
- `void _start()`
- `protected void _stop()` Stops the active component OSPF
- `void dataArriveAtDownPort(java.lang.Object data_, drcl.comp.Port downPort_)` Handle data arriving at the down port.
- `protected void delayed_acknowledge(OSPF_LSA lsa)` Delayed acknowledgement Include multiple acks in one packet.
- `protected void direct_acknowledge(OSPF_LSA lsa, OSPF_Interface oif, int now_, boolean clone_)` Direct acknowledgement
- `void duplicate(java.lang.Object source_)`
- `protected long[] getAllAddresses()` OSPF-MDR Returns an array containing the addresses
- `java.lang.String[] getDebugLevelNames()`
<table>
<thead>
<tr>
<th>java.lang.String</th>
<th></th>
<th><code>getName()</code></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>protected int</td>
<td><code>inactivity_timer</code></td>
<td><code>OSPF Neighbor nbr</code></td>
<td></td>
</tr>
<tr>
<td>java.lang.String</td>
<td></td>
<td><code>info()</code></td>
<td></td>
</tr>
<tr>
<td>protected boolean</td>
<td><code>isSameNL</code></td>
<td><code>java.util.Vector NL1, java.util.Vector NL2</code></td>
<td></td>
</tr>
<tr>
<td>protected void</td>
<td><code>lsa_receive</code></td>
<td><code>OSPF LSA lsa, OSPF Neighbor from</code></td>
<td>Process every LSA received in LS update packet.</td>
</tr>
<tr>
<td>protected void</td>
<td><code>neighborDownEventHandler</code></td>
<td><code>int ifindex_, drcl.inet.data.NetAddress neighbor_, drcl.comp.Port inPort_</code></td>
<td></td>
</tr>
<tr>
<td>protected void</td>
<td><code>neighborUpEventHandler</code></td>
<td><code>int ifindex_, drcl.inet.data.NetAddress neighbor_, drcl.comp.Port inPort_</code></td>
<td>Called back when a neighbor up event is received.</td>
</tr>
<tr>
<td>protected void</td>
<td><code>ospf_add_lsd MANET</code></td>
<td><code>Router LSA lsa, OSPF Interface oif</code></td>
<td></td>
</tr>
<tr>
<td>protected void</td>
<td><code>ospf_area rt install</code></td>
<td><code>OSPF Area area</code></td>
<td>Install area's vertex_list to routing table.</td>
</tr>
<tr>
<td>protected void</td>
<td><code>ospf_dbdesc_sequnum init</code></td>
<td><code>OSPF Neighbor nbr</code></td>
<td></td>
</tr>
<tr>
<td>protected <code>OSPF Interface</code></td>
<td><code>ospf_if_lookup_by_addr</code></td>
<td><code>long addr</code></td>
<td></td>
</tr>
<tr>
<td>protected void</td>
<td><code>ospf_install_candidate</code></td>
<td><code>drcl.util.queue.TreeMapQueue candidate, OSPF SPF vertex w</code></td>
<td>Install vertex w into candidate list the candidate list is sorted from the min cost to max cost.</td>
</tr>
<tr>
<td>protected void</td>
<td><code>ospf_lsa_flood</code></td>
<td><code>OSPF LSA lsa</code></td>
<td>flood ospf_lsa within appropriate scope.</td>
</tr>
<tr>
<td>protected void</td>
<td><code>ospf_lsa_flood</code></td>
<td><code>OSPF LSA lsa, int scope_type</code></td>
<td>flood ospf_lsa within appropriate scope.</td>
</tr>
<tr>
<td>protected int</td>
<td><code>ospf_lsa_has_link</code></td>
<td><code>OSPF LSA w, OSPF LSA v</code></td>
<td>Test whether there is a link in LSA w pointer to vertex v returns 1 if a link exists in w -&gt; v.</td>
</tr>
<tr>
<td>protected void</td>
<td><code>ospf_lsdb install</code></td>
<td><code>OSPF LSA new lsa, OSPF Area area</code></td>
<td>When installing more recent LSA, must detach less recent database copy from LS-list of neighbors, and attach new one.</td>
</tr>
<tr>
<td>protected void</td>
<td><code>ospf_lsdb install</code></td>
<td><code>OSPF LSA new lsa, OSPF LSA old lsa</code></td>
<td></td>
</tr>
<tr>
<td>protected void</td>
<td><code>ospf_lsdb install</code></td>
<td><code>OSPF LSA new lsa, OSPF LSA old lsa, int scope_type</code></td>
<td>Install a new LSA into the database.</td>
</tr>
<tr>
<td>protected void</td>
<td><code>ospf_lsdb install</code></td>
<td><code>OSPF LSA new lsa, OSPF LSA old lsa, OSPF Area area</code></td>
<td>Replaces old_lsa with new_lsa and removes old_lsa from the retranslists in area.</td>
</tr>
<tr>
<td>protected <code>OSPF LSA</code></td>
<td><code>ospf_lsdb lookup</code></td>
<td><code>int type, int id, int advrtr, java.lang.Object scope</code></td>
<td>ordinary lookup function.</td>
</tr>
</tbody>
</table>
protected   Router LSA   ospf_make_router_lsa(OSPF Area area)

protected   void   ospf_message_broadcast(int type, java.lang.Object body, int body_size, OSPF_Interface oif)

protected   void   ospf_message_broadcast(OSPF_MESSAGE msg)
   Send the packet to every neighbor connected to this interfac.

protected   void   ospf_message_send(int type, java.lang.Object body, int body_size, long dst, OSPF_Interface oif)
   send the pkt to specific neighbor on specific if OSPF-MDR: modified to be a method that deals with both legacy and MANET interfaces.

protected   void   ospf_message_send(OSPF_MESSAGE msg)
   send the pkt to specific neighbor on specific if

protected   void   ospf_nexthop_calculation(OSPF_Area area, OSPF_SPF_vertex v, OSPF_SPF_vertex w)
   Calculate nexthop from root to vertex W.

protected   void   ospf_possible_ack(OSPF_LSA received, int ismore_recent, OSPF_Interface oif, int impliedack_flag, int duplicate_flag, int now_)
   Possible ack action mentioned in sec 13 (5)(e) and (7)(b)

protected   void   ospf_route_add(long dst, OSPF_SPF_vertex new_en, OSPF_Area area, int now_)
   Add a routing entry into RoutingTable and synchronize with drcl.inet.core.RT

protected   void   ospf_router_lsa_flood(OSPF_Area area, boolean check_)

protected   int   ospf_send_dbdesc(OSPF_Neighbor nbr)
   Send the database description to the peer ref: 10.8 OSPF-MDR: can not send all the dbdesc at one time, therefore a random timer is needed The random timeout timer is set in ospf_message_send() method

protected   int   ospf_send_hello(OSPF_Interface oif)
   Sends hello pkts out at each functioning interface.

protected   int   ospf_send_lsreq(OSPF_Neighbor nbr)
   Send the LS request pkt to its peer.

protected   int   ospf_send_single_hello(OSPF_Interface oif, long dst)
   used when receiving EVENT_IF_NEIGHBOR_UP, send only one hello packet to ensure two-way connection

void   ospf_set_area_id(int id_)
   xxx: Specify the area id for this router (Needed to be modified)

protected   void   ospf_spf_init(OSPF Area area)
   Initialization of calculating SPF

protected
static   OSPF_SPF_vertex   ospf_vertex_lookup(drcl.util.queue.TreeMapQueue list, long id)
   Test whether a vertex exists in the list xxx: a hashtable would help the performance

protected
static   OSPF_SPF_vertex   ospf_vertex_lookup(java.util.Vector list, long id)
   Test whether a vertex exists in the list xxx: a hashtable would help the performance

protected   void   process(java.lang.Object data_, drcl.comp.Port inPort_)

void   reset()

protected   void   timeout(java.lang.Object evt_)
   Handling timeout event.
protected void transit_vertex_rtable_install(OSPF_SPF_vertex v, 
    OSPF_Area area, int now_)

    Hold the intermediate results on area routing table Install a new entry into routing table

Methods inherited from class drcl.inet.protocol.Routing

addRTEntry, addRTEntry, addRTEntry, addRTEntry, addRTEntry, addRTEntry, addRTEntry, addRTEntry, 
graftRTEntry, graftRTEntry, graftRTEntry, graftRTEntry, pruneRTEntry, pruneRTEntry, pruneRTEntry, 
removeAllRTEntries, removeRTEntry, removeRTEntry, replaceRTEntry, replaceRTEntry, 
retrieveAllRTEntries, retrieveBestRTEntryDest, retrieveRTEntry, retrieveRTEntryDest, 
retrieveRemoteSrc

Methods inherited from class drcl.inet.Protocol

broadcast, broadcast, broadcast, broadcast, broadcast, 
createConfigSwitchPort, createIDChangedEventPort, 
createIFEventPort, createIFQueryPort, 
createLinkBrokenEventPort, createMcastHostEventPort, createMcastQueryPort, 
createMulticastRCThangedEventPort, createPktArrivalEventPort, 
createRTServicePort, createUcastQueryPort, createUnicastRCThangedEventPort, 
createVIFEEventPort, forward, forward, forward, forward, forward, forward, forward, forward, 
idAddedEventHandler, idRemovedEventHandler, LinkBrokenEventHandler, 
mcastHostJoinEventHandler, mcastHostLeaveEventHandler, pktArrivalHandler, 
processOther, routeQueryHandler, routeQueryHandler, rtAddedEventHandler, 
rtModifiedEventHandler, rtRemovedEventHandler, vNeighborDownEventHandler, 
vNeighborUpEventHandler

Methods inherited from class drcl.net.Module

cancelTimeout, dataArriveAtUpPort, deliver, removeDefaultDownPort, 
removeDefaultUpPort, removeTimerPort, setTimeout, setTimeoutAt

Methods inherited from class drcl.comp.Component

addComponent, addComponent, addEventPort, addEventPort, addForkPort, 
addPort, addPort, addPort, addPort, addPort, addPort, addPort, addPort, 
addServerPort, addServerPort, cancelFork, componentAdded, componentRemoved, 
connect, containsComponent, containsComponent, containsPort, debug, 
disconnectAll, disconnectAllPeers, disconnectAllPorts, drop, drop, error, 
error, expose, exposeEventPorts, exposePort, exposePort, exposePort, 
exposePort, exposePort, findAvailable, findAvailable, findAvailable, 
finishing, fork, forkAt, getAllComponents, getAllPorts, 
getAllWiresInside, getAllWiresInsideOut, getAllWiresOut, getComponent, 
addComponentFlag, getComponentFlag, getContract, getContractHT, 
getDebugFlagsInBinary, getForkManager, getID, getParent, 
getPort, getPort, getRoot, getRuntime, getTime, iduplicate, 
isAncestorOf, isComponentNotificationEnabled, isContainer, 
isDebugEnabled, isDebugEnabledAt, isDebugEnabled, 
isDebugEnabledAt, isDirectlyRelatedTo, isDirectOutputEnabled, isEnabled, 
isErrorNoticeEnabled, isEventExportEnabled, isGarbageDisplayEnabled, 
isGarbageEnabled, isPortNotificationEnabled, isPortRemovable, isStarted, 
isStopped, isTraceEnabled, lock, notify, notifyAll, operate, portAdded, 
portRemoved, reboot, removeAll, removeAllComponents, removeAllPorts, 
removeAllPorts, removeComponent, removeComponent, removePort, removePort, 
removePort, resume, run, sduplicate, send, sendAt, setComponentFlag, 
setComponentFlag, setComponentNotificationEnabled, setContract, 
setDebugEnabled, setDebugEnabled, setDebugEnabledAt, setDebugEnabledAt, 
setDebugEnabledAt, setDirectOutputEnabled, setDirectOutputEnabled, 
setEnabled, setErrorNoticeEnabled, setErrorNoticeEnabled,
Methods inherited from class drcl.DrcObj

clone

Methods inherited from class java.lang.Object
equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail
debug

public static boolean debug

debug2

public static boolean debug2

HELLO

protected static final int HELLO
A constant indicating a hello packet.
See Also:
Constant Field Values

DATABASE

protected static final int DATABASE
A constant indicating a database description packet.
See Also:
Constant Field Values

LS_REQUEST

protected static final int LS_REQUEST
A constant indicating a link state request packet.
See Also:
Constant Field Values

LS_UPDATE

protected static final int LS_UPDATE
A constant indicating a link state update packet.
See Also:
Constant Field Values

LS_ACK

protected static final int LS_ACK
A constant indicating a link state acknowledge packet.
See Also:
Constant Field Values
router_id

protected int router_id
OSPF router ID of this node

area_list

protected java.util.Vector area_list
The list of OSPF areas to which this router is attached

ospf_if_list

protected java.util.Vector ospf_if_list
The list of OSPF Interfaces to which this router is attached

UDP_PORT

public static final int UDP_PORT
OSPF-MDR: Borrow the UDP port used by OLSR, assigned by IANA.
See Also:
Constant Field Values

DEBUG_SAMPLE

public static final int DEBUG_SAMPLE
Debug level to enable sampled debug messages.
See Also:
Constant Field Values

DEBUG_NEIGHBOR

public static final int DEBUG_NEIGHBOR
Debug level of neighbor state changes.
See Also:
Constant Field Values

DEBUG_SEND

public static final int DEBUG_SEND
Debug level of sending ospf packets.
See Also:
Constant Field Values

DEBUG_SPF

public static final int DEBUG_SPF
Debug level of calculating SPF tree and installing route entries.
See Also:
Constant Field Values

DEBUG_REFRESH

public static final int DEBUG_REFRESH
Debug level of LSA refresh events.
See Also:
Constant Field Values

DEBUG_LSA

public static final int DEBUG_LSA
Debug level of handling LSA update packets.
See Also:
Constant Field Values

**DEBUG_ACK**

public static final int **DEBUG_ACK**
Debug level of handling LSA acknowledgement packets.
See Also:
Constant Field Values

**DEBUG_TIMEOUT**

public static final int **DEBUG_TIMEOUT**
Debug level of handling timeout events.
See Also:
Constant Field Values

**DEBUG-detail**

public static final int **DEBUG-detail**
Debug level of detailed messages for other levels.
See Also:
Constant Field Values

## Constructor Detail

**OSPF**

public **OSPF**( )
Constructor.

## Method Detail

**getName**

public java.lang.String **getName**( )

**getDebugLevelNames**

public java.lang.String[] **getDebugLevelNames**( )

**_start**

public void **_start**( )

**_stop**

protected void **_stop**( )
Stops the active component OSPF

**_resume**

protected void **_resume**( )
Resumes the active component OLSR.

**info**

public java.lang.String **info**( )

**reset**

public void **reset**( )
public void **duplicate**(java.lang.Object source_)

**Specify by:**
duplicate in interface drcl.ObjectDuplicable

---

protected void **timeout**(java.lang.Object evt_)
Handling timeout event. According to the timeout event which can be obtained from evt_, there are different handling functions.

**Parameters:**
evt_ - event object got from the timer port

---

protected void **process**(java.lang.Object data_,
   drcl.comp.Port inPort_)

---

public void **dataArriveAtDownPort**(java.lang.Object data_,
   drcl.comp.Port downPort_)
Handle data arriving at the down port. According to the different packet types (OSPF Hello pkt, Database description pkt, Link State update pkt, LS request pkt, or LS ack pkt), different corresponding methods can handle the packet. Note for OSPF_Hello pkt will be received only once. The other hello maintaince is done by drcl.inet.core.Hello

**Parameters:**
data_ - message body arriving at the down port.
downPort_ - down port at which messages arrive.

---

protected void **neighborUpEventHandler**(int ifindex_,
   drcl.inet.data.NetAddress neighbor_,
   drcl.comp.Port inPort_)
Called back when a neighbor up event is received. Create a OSPF_Neighbor data associated with OSPF_Interface data in this router.

**Parameters:**
ifindex_ - index of the interface.
See Also:
Hello

---

protected void **neighborDownEventHandler**(int ifindex_,
   drcl.inet.data.NetAddress neighbor_,
   drcl.comp.Port inPort_)

---

protected boolean **isSameNL**(java.util.Vector NL1,
   java.util.Vector NL2)

---

protected int **inactivity_timer**(OSPF_Neighbor nbr)

---

Isa_receive
protected void lsa_receive(OSPF_LSA lsa, 
    OSPF_Neighbor from)
Process every LSA received in LS update packet. Maintain the Link State database stored in the router. Compare 
the existing LS in the database to the received one. Update the database if the received is more updated. 
Broadcast to its neighbor if necessary. Finally ack the LS to its originator. ref: section 13

ospf_make_router_lsa

protected void ospf_make_router_lsa(OSPF_Area area)

ospf_add_lsd MANET

protected void ospf_add_lsd MANET(Router_LSA lsa, 
    OSPF_Interface oif)

direct_acknowledge

protected void direct_acknowledge(OSPF_LSA lsa, 
    OSPF_Interface oif, 
    int now_, 
    boolean clone_)
Direct acknowledgement

delayed_acknowledge

protected void delayed_acknowledge(OSPF_LSA lsa)
Delayed acknowledgement Include multiple acks in one packet. Hold these acks until the timer timeout. ref: sec. 
13.5

ospf_possible_ack

protected void ospf_possible_ack(OSPF_LSA received, 
    int ismore_recent, 
    OSPF_Interface oif, 
    int impliedack_flag, 
    int duplicate_flag, 
    int now_)
Possible ack action mentioned in sec 13 (5)(e) and (7)(b)

ospf_message_send

protected void ospf_message_send(int type, 
    java.lang.Object body, 
    int body_size, 
    long dst, 
    OSPF_Interface oif)
send the pkt to specific neighbor on specific if OSPF-MDR: modified to be a method that deals with both legacy 
and MANET interfaces. For legacy, call the real send method; for MANET, start a timeout timer and add the 
message to the queue

ospf_message_send

protected void ospf_message_send(OSPF_MESSAGE msg)
send the pkt to specific neighbor on specific if

ospf_message_broadcast

protected void ospf_message_broadcast(int type, 
    java.lang.Object body,
int body_size,
OSPF_Interface oif)

protected void ospf_message.broadcast(OSPF_MESSAGE msg)
Send the packet to every neighbor connected to this interface. For sending packets downward, we call the
forward(), defined in Protocol.java void forward(PacketBody p_, long src_, long dest_, int dest_uhp_, boolean
routerAlert_, int TTL, int ToS, int link_id) route-lookup forwarding routerAlert is set to be true, in which the
packet is sent in only one hop. In this case, whatever dst is irrelevant

ospf_send_hello

protected int ospf_send_hello(OSPF_Interface oif)
Sends hello pkts out at each functioning interface. (The method is not used anymore, we are using
drl.net.core.Hello as underlying Hello service provider) Ref: sec. 9.5

ospf_send_single_hello

protected int ospf_send_single_hello(OSPF_Interface oif,
long dst)
used when receiving EVENT_IF_NEIGHBOR_UP, send only one hello packet to ensure two-way connection

ospf_send_dbdesc

protected int ospf_send_dbdesc(OSPF_Neighbor nbr)
Send the database description to the peer ref: 10.8 OSPF-MDR: can not send all the dbdesc at one time, therefore
a random timer is needed The random timeout timer is set in ospf_message_send() method

ospf_send_lsreq

protected int ospf_send_lsreq(OSPF_Neighbor nbr)
Send the LS request pkt to its peer. Ref: sec. 10.9

ospf_router_lsa_flood

protected void ospf_router_lsa_flood(OSPF_Area area,
boolean check_ )

ospf_lsa_flood

protected void ospf_lsa_flood(OSPF_LSA lsa)
flood ospf_lsa within appropriate scope

ospf_lsa_flood

protected void ospf_lsa_flood(OSPF_LSA lsa,
int scope_type)
flood ospf_lsa within appropriate scope

ospf_lsdcb_lookup

protected OSPF_LSA ospf_lsdcb_lookup(int type,
int id,
int advrtr,
java.lang.Object scope)
ordinary lookup function

ospf_lsdcb_install
protected void `ospf_lsdb_install`\(\text{OSPF\_LSA}\ new\_lsa,\ \text{OSPF\_Area}\ area\)\)

When installing more recent LSA, must detach less recent database copy from LS-lists of neighbors, and attach new one. ref. sec 13.2

`ospf_lsdb_install`

protected void `ospf_lsdb_install`\(\text{OSPF\_LSA}\ new\_lsa,\ \text{OSPF\_LSA}\ old\_lsa,\ \text{OSPF\_Area}\ area\)\)

Replaces old\_lsa with new\_lsa and removes old\_lsa from the retranslists in area. Recalculates SPF if necessary (see 13.2). This is the main method of `ospf_lsdb_install`, and it calls `ls\_db.ospf_lsdb_replace` to replace the lsa in the database it calls also `lsa_change` later to update the rt when lsa is not the same

`ospf_lsdb_install`

protected void `ospf_lsdb_install`\(\text{OSPF\_LSA}\ new\_lsa,\ \text{OSPF\_LSA}\ old\_lsa\)\)

`ospf_lsdb_install`

protected void `ospf_lsdb_install`\(\text{OSPF\_LSA}\ new\_lsa,\ \text{OSPF\_LSA}\ old\_lsa,\ \text{int}\ scope\_type\)\)

Install a new LSA into the database

`ospf_if_lookup_by_addr`

protected `OSPf\_Interface` `ospf_if_lookup_by_addr`\(\text{long}\ addr\)\)

`ospf_set_area_id`

public void `ospf_set_area_id`\(\text{int}\ id\)\)

xxx: Specify the area id for this router (Needed to be modified)

`ospf_dbdesc_seqnum_init`

protected void `ospf_dbdesc_seqnum_init`\(\text{OSPf\_Neighbor}\ nbr\)\)

`ospf_spf_init`

protected void `ospf_spf_init`\(\text{OSPf\_Area}\ area\)\)

Initialization of calculating SPF

`ospf_nexthop_calculation`

protected void `ospf_nexthop_calculation`\(\text{OSPf\_Area}\ area,\ \text{OSPf\_SPF\_vertex}\ v,\ \text{OSPf\_SPF\_vertex}\ w\)\)

Calculate nexthop from root to vertex W. Ref: sec. 16.1.1

`ospf_area_rt_install`

protected void `ospf_area_rt_install`\(\text{OSPf\_Area}\ area\)\)

Install area’s vertex\_list to routing table

`transit_vertex_rtable_install`

protected void `transit_vertex_rtable_install`\(\text{OSPf\_SPF\_vertex}\ v,\ \text{OSPf\_Area}\ area\)\)

29
int now_)
Hold the intermediate results on area routing table Install a new entry into routing table

ospf_route_add

protected void ospf_route_add(long dst,
                             OSPF_SPF_vertex new_en,
                             OSPF_Area area,
                             int now_)
Add a routing entry into RoutingTable and synchronize with drcl.inet.core.RT

ospf_vertex_lookup

protected static OSPF_SPF_vertex ospf_vertex_lookup(java.util.Vector list,
                                                   long id)
Test whether a vertex exists in the list xxx: a hashtable would help the performance

ospf_vertex_lookup

protected static OSPF_SPF_vertex ospf_vertex_lookup(drcl.util.queue.TreeMapQueue list,
                                                   long id)
Test whether a vertex exists in the list xxx: a hashtable would help the performance

ospf_install_candidate

protected void ospf_install_candidate(drcl.util.queue.TreeMapQueue candidate,
                                      OSPF_SPF_vertex w)
Install vertex w into candidate list the candidate list is sorted from the min cost to max cost

ospf_lsa_has_link

protected int ospf_lsa_has_link(OSPF_LSA w,
                                OSPF_LSA v)
Test whether there is a link in LSA w pointer to vertex v returns 1 if a link exists in w --> v

getAllAddresses

protected long[] getAllAddresses()
OSPF-MDR Returns an array containing the addresses
Returns:
an long[] array.

Package | Class | Tree | Deprecated | Index | Help
---|---|---|---|---|---
PREV CLASS | NEXT CLASS | FRAMES | NO FRAMES | All
SUMMARY: NESTED | FIELD | CONSTR | METHOD | DETAIL: FIELD | CONSTR | METHOD

Class OSPF_Interface

java.lang.Object
   \_drcl.inet.protocol.ospfmdr.OSPF_Interface

public class OSPF_Interface
extends java.lang.Object
An OSPF interface is the connection between a router and a network. One or more router adjacencies may
develop over an interface. A router's LSAs reflect the state of its interfaces and their associated adjacencies.
The OSPF interface is extended to support MANET. A new interface type is added and several new variables and parameters are added.

See Also:
OSPF, OSPF Neighbor, OSPF_TimeOut_EVT

### Field Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected</td>
<td>OSPF_TimeOut_EVT</td>
<td>Ack_Delay_Reach_EVT</td>
</tr>
<tr>
<td>protected</td>
<td>OSPF Area</td>
<td>area</td>
</tr>
<tr>
<td>protected</td>
<td>int</td>
<td>bdr</td>
</tr>
<tr>
<td>protected</td>
<td>long</td>
<td>bmdr_parent</td>
</tr>
<tr>
<td>protected</td>
<td>java.util.Vector</td>
<td>DNL</td>
</tr>
<tr>
<td>protected</td>
<td>int</td>
<td>dr</td>
</tr>
<tr>
<td>protected</td>
<td>OSPF_TimeOut_EVT</td>
<td>Hello_TimeOut_EVT</td>
</tr>
<tr>
<td>protected</td>
<td>java.util.Vector</td>
<td>HNL</td>
</tr>
<tr>
<td>protected</td>
<td>int</td>
<td>hsn</td>
</tr>
<tr>
<td>protected</td>
<td>int</td>
<td>if_id</td>
</tr>
<tr>
<td>protected</td>
<td>int</td>
<td>ifmtu</td>
</tr>
<tr>
<td>protected</td>
<td>long</td>
<td>ip_if_addr</td>
</tr>
<tr>
<td>protected</td>
<td>long</td>
<td>lladdr</td>
</tr>
<tr>
<td>protected</td>
<td>java.util.Vector</td>
<td>LNL</td>
</tr>
<tr>
<td>protected</td>
<td>java.util.Vector</td>
<td>lsa_delayed_ack</td>
</tr>
<tr>
<td>double</td>
<td>MAXJITTER</td>
<td>OSPF-MDR: The maximum value used for jitter, used in diverse timeout event</td>
</tr>
<tr>
<td>protected</td>
<td>int</td>
<td>mdr_level</td>
</tr>
<tr>
<td>protected</td>
<td>long</td>
<td>mdr_parent</td>
</tr>
<tr>
<td>protected</td>
<td>java.util.Vector</td>
<td>neighbor_list</td>
</tr>
<tr>
<td>protected</td>
<td>int</td>
<td>ospf_stat_delayed_lsack</td>
</tr>
<tr>
<td>protected</td>
<td>int</td>
<td>ospf_stat_dr_election</td>
</tr>
</tbody>
</table>
protected java.util.Vector **RNL**

protected int **rtr_id**

protected int **rtr_pri**

protected int **rxmt interval**

    The number of seconds between LSA retransmissions, for
    adjacencies belonging to this interface.

protected int **state**

protected int **transdelay**

    I/F transmission delay, the approximate delay over this link

protected int **type**

    the type info from interface, default value: IF_PTOP

### Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>protected OSPF_Interface()</code></td>
<td>Constructor OSPF-MDR: new data items are initialized</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>protected void add_new_neighbor(OSPF_Neighbor nbr)</code></td>
<td>Add one new neighbor to this interface OSPF-MDR: modified to only add neighbor when the neighbor is not on the list</td>
</tr>
<tr>
<td><code>void BackupMDRSelection(OSPF_Neighbor currentNode, OSPF_Neighbor Rmax, java.util.ArrayList Rmax1hopNeighbors, int[][] NCM, int[][] LCM)</code></td>
<td></td>
</tr>
<tr>
<td><code>protected void clear()</code></td>
<td>Disable all timer</td>
</tr>
<tr>
<td><code>int compareTriplet(OSPF_Neighbor node1, OSPF_Neighbor node2)</code></td>
<td></td>
</tr>
<tr>
<td><code>int[][] computeLinkCostMatrix(OSPF_Neighbor node, int[][] NCM)</code></td>
<td></td>
</tr>
<tr>
<td><code>int[][] computeLinkCostMatrix2(OSPF_Neighbor node, int[][] NCM, int[][] LCM)</code></td>
<td></td>
</tr>
<tr>
<td><code>int[][] computeNCM()</code></td>
<td>5.1 Phase 1: creating the Neighbor Connectivity Matrix * Reported Neighbor List (RNL) The Reported Neighbor List for the neighbor, which is updated when a Hello is received from the neighbor that contains an RNL TLV.</td>
</tr>
<tr>
<td><code>OSPF_Neighbor computeRmax()</code></td>
<td></td>
</tr>
<tr>
<td><code>protected int get_cost()</code></td>
<td></td>
</tr>
<tr>
<td><code>java.util.Vector getAdjNeighbors()</code></td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OSPF Neighbor</td>
<td>getMinimumHops2Unlabeled()</td>
</tr>
<tr>
<td>protected int if_is_pointopoint()</td>
<td>Check whether this interface is the type of point-to-point</td>
</tr>
<tr>
<td>java.lang.String info</td>
<td>getMinimumHops2Unlabeled()</td>
</tr>
<tr>
<td>protected int Interface_Down()</td>
<td></td>
</tr>
<tr>
<td>protected int Interface_Up()</td>
<td></td>
</tr>
<tr>
<td>protected void MDRNeighborChange()</td>
<td>OSPF-MDR: replace the OSPF’s neighbor_change method in the MANET interface</td>
</tr>
<tr>
<td>void MDRParentSelection(OSPF_Neighbor current, OSPF_Neighbor Rmax)</td>
<td></td>
</tr>
<tr>
<td>int[][] MDRSelection(OSPF_Neighbor current, OSPF_Neighbor Rmax, java.util.ArrayList Rmax1hopNeighbors, int[][] NCM)</td>
<td></td>
</tr>
<tr>
<td>protected int neighbor_change()</td>
<td>One of the neighbors associated with this interface change its state</td>
</tr>
<tr>
<td>protected int ospf_interface_count_full_nbr()</td>
<td>count number of full neighbor adjacent to this interface</td>
</tr>
<tr>
<td>static OSPF Interface ospf_interface_create(int if_index, int type_, int mtu_, OSPF_Area area, drcl.comp.Port out_)</td>
<td>Constructor: Create new ospf6 interface structure. In our implementation, this function is called when OSPF receives NEIGHBOR_UP event so that the state of interface is UP</td>
</tr>
<tr>
<td>protected OSPF Neighbor ospf_nbr_lookup_by_routerid(long id)</td>
<td>Lookup the neighbor associated with this interface with router id OSPF-MDR: used in Process_Hello_MANET</td>
</tr>
<tr>
<td>protected void remove_neighbor(OSPF_Neighbor nbr)</td>
<td>Remove one new neighbor to this interface</td>
</tr>
<tr>
<td>void reset()</td>
<td></td>
</tr>
<tr>
<td>protected void set_cost(int cost_)</td>
<td></td>
</tr>
<tr>
<td>java.lang.String toString()</td>
<td></td>
</tr>
</tbody>
</table>

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait

Field Detail

if_id
protected int if_id

area

protected OSPF_Area area
A router interface belongs to one and only one area

type

protected int type
the type info from interface, default value: IF_PTOP

state

protected int state

lladdr

protected long lladdr

MAXJITTER

public final double MAXJITTER
OSPF-MDR: The maximum value used for jitter, used in diverse timeout event
See Also:
Constant Field Values
transdelay

protected int transdelay
I/F transmission delay, the approximate delay over this link

Hello_TimeOut_EVT

protected OSPF_TimeOut_EVT Hello_TimeOut_EVT

Ack_Delay_Reach_EVT

protected OSPF_TimeOut_EVT Ack_Delay_Reach_EVT

neighbor_list

protected java.util.Vector neighbor_list
list of ospf neighbor over this interface

rxmt_interval

protected int rxmt_interval
The number of seconds between LSA retransmissions, for adjacencies belonging to this interface. Also used when retransmitting Database Description and Link State Request Packets.

lsa_delayed_ack

protected java.util.Vector lsa_delayed_ack
LSAs to Delayed Acknowledge

ifmtu

protected int ifmtu
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ospf_stat_dr_election</td>
<td>protected int</td>
</tr>
<tr>
<td>ospf_stat_delayed_lsack</td>
<td>protected int</td>
</tr>
<tr>
<td>rtr_pri</td>
<td>protected int</td>
</tr>
<tr>
<td>ip_if_addr</td>
<td>protected long</td>
</tr>
<tr>
<td>dr</td>
<td>protected int</td>
</tr>
<tr>
<td>bdr</td>
<td>protected int</td>
</tr>
<tr>
<td>rtr_id</td>
<td>protected int</td>
</tr>
<tr>
<td>mdr_level</td>
<td>protected int</td>
</tr>
<tr>
<td>mdr_parent</td>
<td>protected long</td>
</tr>
<tr>
<td>bmdr_parent</td>
<td>protected long</td>
</tr>
<tr>
<td>hsn</td>
<td>protected int</td>
</tr>
<tr>
<td>LNL</td>
<td>protected java.util.Vector</td>
</tr>
<tr>
<td>HNL</td>
<td>protected java.util.Vector</td>
</tr>
<tr>
<td>RNL</td>
<td>protected java.util.Vector</td>
</tr>
<tr>
<td>DNL</td>
<td></td>
</tr>
</tbody>
</table>
protected java.util.Vector DNL.

**Constructor Detail**

OSPF_Interface

protected **OSPF_Interface()**  
Constructor OSPF-MDR: new data items are initialized

**Method Detail**

get_cost

protected int **get_cost()**

set_cost

protected void **set_cost(int cost_)**

toString

public java.lang.String **toString()**

info

public java.lang.String **info(java.lang.String prefix_)**

reset

public void **reset()**

clear

protected void **clear()**  
Disable all timer

ospf_interface_create

public static **OSPF_Interface ospf_interface_create(int if_index,**
  int type_,
  int mtu_,
  **OSPF_Area area_**,  
  **rcl.comp.Port out_)**

Constructor: Create new ospf6 interface structure In our implementation, this function is called when OSPF receives NEIGHBOR_UP event so that the state of interface is UP

ospf_interface_count_full_nbr

protected int **ospf_interface_count_full_nbr()**  
count number of full neighbor adjacent to this interface

Interface_Up

protected int **Interface_Up()**  
Never called in this implemented since we use CSL hello service Interface_UP event is trigger be NEIGHBOR_UP event

Interface_Down

protected int **Interface_Down()**

neighbor_change
protected int neighbor_change()
One of the neighbors associated with this interface change its state

OSPFRouter ospf_nbr_lookup_by_routerid

protected OSPF_Neighbor ospf_nbr_lookup_by_routerid(long id)
Lookup the neighbor associated with this interface with router id OSPF-MDR: used in Process_Hello_MANET
Parameters:
  id - the router id
Returns:
an OSPF_Neighbor object if exists, otherwise null

if_is_pointopoint

protected int if_is_pointopoint()
Check whether this interface is the type of point-to-point

add_new_neighbor

protected void add_new_neighbor(OSPF_Neighbor nbr)
Add one new neighbor to this interface OSPF-MDR: modified to only add neighbor when the neighbor is not on the list

remove_neighbor

protected void remove_neighbor(OSPF_Neighbor nbr)
Remove one new neighbor to this interface

MDRNeighborChange

protected void MDRNeighborChange()
OSPF-MDR: replace the OSPF's neighbor_change method in the MANET interface

computeNCM

public int[][] computeNCM()
5.1 Phase 1: creating the Neighbor Connectivity Matrix * Reported Neighbor List (RNL) The Reported Neighbor List for the neighbor, which is updated when a Hello is received from the neighbor that contains an RNL TLV. The Reported Neighbor Lists for all neighbors represent the 2-hop neighbor information. Report2Hop: For neighbor j, a single-bit variable is equal to 1 if a full Hello (which contains a full Reported Neighbor List) has been received from it (the neighbor j).
Returns:
assign a value of 0 or 1 for each pair of (bidirectional) neighbors NCM is symmetric, therefor NCM(j,k) is always equal to NCM(k,j) (1.1) If Report2Hop=1 for both neighbors j and k: NCM(j,k)=NCM(k,j)=1 only if j belongs to the RNL of k and k belongs to the RNL of j (1.2) If Report2Hop=1 for neighbor j and is 0 for k: NCM(j,k)=NCM(k,j)=1 only if k belongs to the RNL of j (1.3) If Report2Hop=0 for both neighbors j and k: NCM(j,k)=NCM(j,k)=0

MDRSelection

public int[][] MDRSelection(OSPF_Neighbor currentNode,
OSPF_Neighbor Rmax,
java.util.ArrayList Rmax1hopNeighbors,
int[][] NCM)

BackupMDRSelection

public void BackupMDRSelection(OSPF_Neighbor currentNode,
OSPF_Neighbor Rmax,
java.util.ArrayList Rmax1hopNeighbors,
public void MDRParentSelection(OSPF_Neighbor currentNode, OSPF_Neighbor Rmax)

calculateLinkCostMatrix

calculateLinkCostMatrix(int[][] NCM)

calculateLinkCostMatrix2

calculateLinkCostMatrix2(OSPF_Neighbor node, int[][] NCM)

calculateRmax

calculateRmax(OSPF_Neighbor node)

calculateTriplet

calculateTriplet(OSPF_Neighbor node1, OSPF_Neighbor node2)

getMinimumHops2Unlabeled

getMinimumHops2Unlabeled()

getAdjNeighbors

getAdjNeighbors()

Class OSPF_Neighbor

java.lang.Object

drcl.inet.protocol.ospfmdr.OSPF_Neighbor

public class OSPF_Neighbor extends java.lang.Object

The information exchanged with other adjacent nodes is described by a OSPF_Neighbor data structure, which is bounded to a specific OSPF router interface. Ref: sec.10

See Also:
OSPF, OSPF_Interface, OSPF_DBdesc, OSPF_LSA, OSPF_TimeOut_EVT

Constructor Summary
<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected void clear()</td>
</tr>
<tr>
<td>java.lang.String info(java.lang.String prefix_)</td>
</tr>
<tr>
<td>protected void ospf_add_request(OSPF_LSA lsa)</td>
</tr>
<tr>
<td>protected void ospf_add_retrans(OSPF_LSA lsa)</td>
</tr>
<tr>
<td>protected void ospf_add_summary(OSPF_LSA lsa)</td>
</tr>
<tr>
<td>protected OSPF_LSA ospf_lookup_request(OSPF_LSA lsa)</td>
</tr>
<tr>
<td>protected OSPF_LSA ospf_lookup_retrans(OSPF_LSA_Header h)</td>
</tr>
<tr>
<td>protected void ospf_remove_summary_all()</td>
</tr>
<tr>
<td>protected void ospf_remove_summary(OSPF_LSA lsa)</td>
</tr>
<tr>
<td>protected void prepare_neighbor_lsdb(int now_)</td>
</tr>
<tr>
<td>void reset()</td>
</tr>
<tr>
<td>java.lang.String toString()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods inherited from class java.lang.Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSPF_Neighbor()</td>
</tr>
<tr>
<td>Constructor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OSPF_Neighbor(int src, OSPF_Interface oif)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static function used to create an OSPF_Neighbor entity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
</tr>
</tbody>
</table>

| 39 |
protected void clear()
Clear all the timer associated with this nbr. This Function is called when the router detect the nbr is down.

protected void prepare_neighbor_lsdbs(int now_)
When the neighbor reaches state 'Exchange', we have to list ls summary list for it. Prepare for dd exchange

protected void ospf_add_summary(OSPF_LSA lsa)
add lsa to summary list of neighbor

protected void ospf_remove_summary(OSPF_LSA lsa)
remove lsa from summary list of neighbor

protected void ospf_remove_summary_all()
remove all lsa from retrans list of neighbor

protected void ospf_add_request(OSPF_LSA lsa)
add lsa to summary list of neighbor

protected OSPF_LSA ospf_lookup_retrans(OSPF_LSA_Header h)
lookup lsa on retrans list of neighbor

protected void ospf_add_retrans(OSPF_LSA lsa)
add lsa to retrans list of neighbor

protected OSPF_LSA ospf_lookup_request(OSPF_LSA lsa)
Lookup lsa on request list of neighbor. This lookup is different from others, because this lookup is to find the same LSA instance of different memory space

public void reset()

toString

public java.lang.String toString()

info

public java.lang.String info(java.lang.String prefix_)
### Class OSPF_Hello

```java
java.lang.Object
   └── drcl.DrclObj
      └── drcl.inet.protocol.ospfmdr.OSPF_Hello
```

All Implemented Interfaces:

```java
public class OSPF_Hello extends drcl.DrclObj

See Also:
Serialized Form
```

#### Field Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected  int</td>
<td><code>bdr</code></td>
</tr>
<tr>
<td>protected  int</td>
<td><code>dr</code></td>
</tr>
<tr>
<td>protected  double</td>
<td><code>hello_interval</code></td>
</tr>
<tr>
<td>protected  OSPFMDR_LLS</td>
<td><code>lls</code></td>
</tr>
<tr>
<td>protected  int</td>
<td><code>mdr_level</code></td>
</tr>
<tr>
<td>protected  java.util.Vector</td>
<td><code>neighbor_id_list</code></td>
</tr>
<tr>
<td>protected  int</td>
<td><code>neighbor_no</code></td>
</tr>
<tr>
<td>protected  int</td>
<td><code>options</code></td>
</tr>
<tr>
<td>protected  boolean</td>
<td><code>options_D</code></td>
</tr>
<tr>
<td>protected  boolean</td>
<td><code>options_L</code></td>
</tr>
<tr>
<td>protected  double</td>
<td><code>router_dead_interval</code></td>
</tr>
<tr>
<td>protected  int</td>
<td><code>rtr_pri</code></td>
</tr>
</tbody>
</table>

#### Method Summary
### Field Detail

**hello_interval**

protected double **hello_interval**

**router_dead_interval**

protected double **router_dead_interval**

**options**

protected int **options**

**neighbor_no**

protected int **neighbor_no**

**neighbor_id_list**

protected java.util.Vector **neighbor_id_list**

**dr**

protected int **dr**

**bdr**

protected int **bdr**

---

Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

---

```java
protected void add_neighbor_id(int id)

java.lang.Object clone()

void duplicate(java.lang.Object source_)

protected double get_hello_interval()

protected java.util.Vector get(LLS_TLV_values(int lls_type)

protected double get_router_dead_interval()

protected void set_neighbor_id(java.util.Vector list)

protected int size()

return the packet length of Hello Body (in byte), which excludes the OSPF header length Network Mask: 4, HelloInterval:2, Options:1, Rtr Pri:1, RouterDeadInterval: 4 Designated Router: 4, Backup: 4, the sum of the above items is assigned to FIX HELLO LEN = 20, Neighbor: each 4 byte

java.lang.String toString()
```
protected int rtr_pri
protected int mdr_level
protected boolean options_L
protected boolean options_D
protected OSPFMDR_LLS lls

Method Detail

clone
public java.lang.Object clone()
duplicate
public void duplicate(java.lang.Object source_)
get_hello_interval
protected double get_hello_interval()
get_router_dead_interval
protected double get_router_dead_interval()
set_neighbor_id
protected void set_neighbor_id(java.util.Vector list)
add_neighbor_id
protected void add_neighbor_id(int id)
size
protected int size()

return the packet length of Hello Body (in byte) , which excludes the OSPF header length Network Mask: 4, HelloInterval:2, Options:1, Rtr Pri:1, RouterDeadInterval: 4 Designated Router: 4, Backup: 4, the sum of the above items is assigned to FIX_HELLO_LEN = 20, Neighbor: each 4 byte

get_LLS_TLV_values
protected java.util.Vector get_LLS_TLV_values(int lls_type)
toString
public java.lang.String toString()
public class OSPFMDR_LLS extends drcl.DrclObj
OSPF Hello LLS (Local Link Signaling)
ref: A.2.2.6 ref: OSPF_Hello.java for LLS TLV types
See Also:
Serialized Form

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected void add_lls_tlv(OSPFMDR_LLS_TLV lls_tlv)</td>
<td>add_lls_tlv()</td>
<td>Add LLS TLV to OSPFMDR_LLS object</td>
</tr>
<tr>
<td>java.lang.Object clone()</td>
<td>clone()</td>
<td>Clone OSPFMDR_LLS object</td>
</tr>
<tr>
<td>void duplicate(java.lang.Object source_)</td>
<td>duplicate()</td>
<td>Duplicate OSPFMDR_LLS object</td>
</tr>
<tr>
<td>protected int get_lls_data_length()</td>
<td>get_lls_data_length()</td>
<td>Get LLS Data Length of OSPFMDR_LLS object</td>
</tr>
<tr>
<td>protected java.util.Vector get_lls_tlv_values(int tlv_type)</td>
<td>get_lls_tlv_values()</td>
<td>Get LLS TLV Values by Type</td>
</tr>
<tr>
<td>protected java.util.Vector get_lls_tlvs()</td>
<td>get_lls_tlvs()</td>
<td>Get LLS TLVs of OSPFMDR_LLS object</td>
</tr>
<tr>
<td>protected void set_lls_data_length()</td>
<td>set_lls_data_length()</td>
<td>Set LLS Data Length of OSPFMDR_LLS object</td>
</tr>
<tr>
<td>protected void set_lls_tlvs(java.util.Vector lls_tlvs)</td>
<td>set_lls_tlvs()</td>
<td>Set LLS TLVs of OSPFMDR_LLS object</td>
</tr>
<tr>
<td>protected int size()</td>
<td>size()</td>
<td>Get size of OSPFMDR_LLS object</td>
</tr>
<tr>
<td>java.lang.String toString()</td>
<td>toString()</td>
<td>Convert OSPFMDR_LLS object to String</td>
</tr>
<tr>
<td>protected void update_tlv(int type, int len, java.util.Vector tlv_values)</td>
<td>update_tlv()</td>
<td>Update LLS TLV with type, length, and values</td>
</tr>
</tbody>
</table>

Methods inherited from class java.lang.Object
equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait
Method Detail

clone

public java.lang.Object clone()

duplicate

public void duplicate(java.lang.Object source_)

get_lls_data_length

protected int get_lls_data_length()

set_lls_data_length

protected void set_lls_data_length()

get_lls_tlv_values

protected java.util.Vector get_lls_tlv_values()

set_lls_tlv_values

protected void set_lls_tlv_values(java.util.Vector lls_tlvs)

get_lls_tlvs

protected java.util.Vector get_lls_tlvs()

set_lls_tlvs

protected void set_lls_tlvs(java.util.Vector lls_tlvs)

add_lls_tlv

protected void add_lls_tlv(OSPFMDR_LLS_TLV lls_tlv)

update_tlv

protected void update_tlv(int type,
                        int len,
                        java.util.Vector tlv_values)

size

protected int size()

toString

public java.lang.String toString()
public class OSPFMDR_LLS_TLV
extends java.lang.Object

OSPF Hello LLS Type Length Value The LLS TLV types Heard Neighbor List TLV - 11 Reported Neighbor List TLV - 12 Lost Neighbor List TLV - 13 Hello Sequence TLV - 14 MDR TLV - 15 Dependent Neighbor List TLV - 16
ref: A.2.2.6 ref: OSPF.java for LLS TLV types

### Method Summary

<table>
<thead>
<tr>
<th>Method (return type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected void</td>
<td>add_tlv_value(java.lang.Object tlv_value)</td>
</tr>
<tr>
<td>java.lang.Object</td>
<td>clone()</td>
</tr>
<tr>
<td>void</td>
<td>duplicate(java.lang.Object source_)</td>
</tr>
<tr>
<td>protected int</td>
<td>get_tlv_length()</td>
</tr>
<tr>
<td>protected int</td>
<td>get_tlv_type()</td>
</tr>
<tr>
<td>protected java.util.Vector</td>
<td>get_tlv_values()</td>
</tr>
<tr>
<td>protected void</td>
<td>set_tlv_length()</td>
</tr>
<tr>
<td>protected void</td>
<td>set_tlv_type(int tlv_type)</td>
</tr>
<tr>
<td>protected void</td>
<td>set_tlv_values(java.util.Vector tlv_values)</td>
</tr>
<tr>
<td>protected int</td>
<td>size()</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>toString()</td>
</tr>
</tbody>
</table>

### Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

### Method Detail

clonen

public java.lang.Object clone()

duplicate

public void duplicate(java.lang.Object source_)

get_tlv_type

protected int get_tlv_type()

set_tlv_type

protected void set_tlv_type(int tlv_type)
### Field Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected</td>
<td>OSPF_MESSAGE_BROADCAST</td>
</tr>
<tr>
<td>static int</td>
<td></td>
</tr>
<tr>
<td>protected</td>
<td>OSPF_MESSAGE_SEND</td>
</tr>
<tr>
<td>static int</td>
<td></td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>toString()</td>
</tr>
</tbody>
</table>
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait,
wait

Field Detail
OSPF_MESSAGE_SEND

protected static final int OSPF_MESSAGE_SEND
See Also:
Constant Field Values

OSPF_MESSAGE_BROADCAST

protected static final int OSPF_MESSAGE_BROADCAST
See Also:
Constant Field Values

Method Detail
toString

public java.lang.String toString()