Diploma Thesis

Survey of Mobile Ad-hoc Routing Algorithms

Marc Schiely

Dept. of Computer Science
Swiss Federal Institute of Technology (ETH) Zurich
Winter 2003 / 2004

Prof. Dr. Roger Wattenhofer
Distributed Computing Group
Advisor: Regina Bischoff

Abstract

Mobile ad-hoc networks are becoming more and more important. They will become more popular because they have interesting aspects. It is easy and inexpensive to build such networks because no other infrastructure than the mobile devices is needed.

One of the remaining problems in mobile ad-hoc networks is to find a routing algorithm which performs good in all common sorts of networks. This thesis analyzes three link reversal routing algorithms on how the algorithms behave on different classes of networks. Therefor a simulation was made. The framework which was implemented for simulating mobile routing algorithms is presented in this thesis. It is shown how the framework can be used for simulating other mobile routing algorithms.

The results from the simulation are not very spectacular. Unfortunately no noticeable dependence exists between network density, network mobility and the performance of the algorithms. What we expected was, that there exist network scenarios where an algorithm which is bad in one scenario is best in this scenarios.

Also the simulation on different classes of networks does not show any unexpected results. There is no noticeable difference between the performance of the algorithms on different classes of networks. It is always the same of the analyzed algorithms which is best for all four scenarios.

Contents

Chapter 1

Introduction

Since a few years the new technology of wireless networks is becoming famous. Universities and companies have built wireless LANs to allow members to join the wired network from anywhere in the transmission area of the network.

Two concepts of wireless networks exist: Networks with a dedicated access point and networks without dedicated hardware. The former one is more expensive to build, because a wired access point is needed to connect wireless devices to the network. Therefor it is not that flexible as the latter one is. The second method are so-called ad-hoc networks. In such a network each communication partner is equivalent. There is absolutely no need of a wired device. This makes it very cost efficient to install such a net. The only expenses come from buying wireless devices which are also needed in wireless networks with access points.

In this diploma thesis we will only focus on ad-hoc networks. One open problem in ad-hoc networks is the problem of efficient routing. Because of the mobility of the communication partners and because there is no dedicated hardware which is fixed, we can not maintain and use global routing tables. Instead all communication nodes have to cooperate to send a message from one node to another.

Through the movement of the communication nodes, nodes may leave or join the network. Also it will happen that existing links between two nodes break down due to weak radio signals and movements or that new links will arise.

Many routing algorithms for mobile ad-hoc networks have been proposed and have been enhanced [Das et al.] [Vincent D. Park, M. Scott Corson] [Busch et al.]. But it is not clear how different algorithms are behaving in different environments. An algorithm may be best in a special network but worst in another. The goal of this diploma thesis is to classify network and mobility scenarios and to analyze how different algorithms behave in these scenarios. From this classification a new algorithm should be proposed which should behave best in all classes of nets which are being analyzed in this thesis.

For classifying the nets, a simulation framework has been used. The framework was written by Aaron Zollinger in the Distributed Computing Group at ETH Zurich. The concept of mobility has been added to the framework and new algorithms have been implemented for testing. The framework is described in chapter 2.

The simulation itself was executed on different networks with different parameters. Chapter 3 describes the parameters which were defined for the simulation. These parameters define the classes of networks we analyzed.

The results of the simulation are presented in chapter 4. The classification is shown and possible other scenarios are described.

Chapter 5 finally shows what future work on this subject could be. Also a critical analysis of my work is presented there.

Chapter 2

Implementation of Simulation Framework

2.1 The Existing Simulation Framework

2.1.1 Concept

For simulating routing algorithms Aaron Zollinger from the Distributed Computing Group at ETH Zurich wrote a framework in java. The framework allows two different simulation modes: one for visualizing the algorithm and one just for writing the simulation results into a text file. The visualization mode consists of a java GUI which allows the user to simply view and debug the routing procedure.

Different routing algorithms can be plugged into the system and can be compared. For doing fair comparisons the network on which the algorithms are simulated is generated in a separate program. Before running the algorithms the network generator is executed. It generates a network for a given network density and writes it to a file. The generator distributes nodes randomly in the network such that the given density constraint is met.

Afterwards the network is read in from the file for simulation. Then all algorithms can be run on the same network and results can be compared.

2.1.2 Simulating an Algorithm

To simulate a new algorithm in the framework the following requirements have to be met: The routing algorithm must implement the interface *RoutingAlgorithm*. The most important method in the interface is the method $receive(Message)$. It is called when a node receives a message. There it is defined how the node has to proceed when it receives a message. The next host to send the message to is determined and the message is sent out to the node. Table 2.1 shows the interface *RoutingAlgorithm*.

First the net can be generated using the following command:

$$NetGen < NrOfNetworks >< sizeX >< sizeY >< hostDensity >$$

Interface *RoutingAlgorithm*
public void $setNetwork(Network)$ public void $init()$ public void $abort()$ public void $receive(Message)$ public int $getNumOfSentMessages()$ public void $draw(Graphics, scaleFactor)$ public Graph $getGraph()$

Table 2.1: The interface *RoutingAlgorithm*

The generated file has to be renamed to "graph.net", then it can be used in the SimViz class.

Afterwards the algorithm can be simulated and debugged using the visualizer with the following command:

$$SimViz < algorithm1 > [< algorithm2 > [< algorithm3 > [...]]]$$

For simulating algorithms without visualizing them the following command has to be used:

$$Sim < algorithm1 > [< algorithm2 > [< algorithm3 > [...]]]$$

2.2 Concept of Mobility for Simulation

The presented framework worked just for algorithms where the nodes are expected to stay at the same position. What we wanted was to simulate and evaluate algorithms on mobile networks. Therefor we enhanced the framework with a concept of mobility. This concept is presented in the following sections.

2.2.1 Mobility Models

It is obvious that the same routing algorithm behaves different on different mobility models [Camp et al., 2002]. An algorithm which saves all routes in a table works best in static models but has no chance in very mobile nets. On the other hand an algorithm which explores the whole net at each routing request may work best in very mobile networks but is very bad in static nets due to the large overhead.

We wanted to analyze how the same algorithm performs on different mobility models. We wanted that the mobility model can easily be plugged into the framework. Therefor we wrote an interface which is used for realizing the mobility. The interface $MobilityModel$ is shown in table 2.2.

Interface *MobilityModel*
public Point2D.Double $getNewPosition(Host, sizeX, sizeY)$

Table 2.2: The interface *MobilityModel*

2.2.2 Mobility Generator

The same concept as for the generation of networks has been applied for generating mobility files. Before simulating an algorithm, a network and a mobility for the network must be generated. A separate program is used to achieve this. The program generates a random distributed network with a defined density as the normal *NetGen* class does. The program takes this network as input and uses the mobility model to simulate the moves of nodes. If a node is chosen to move then the new position is evaluated. Each move is saved in an array. After the maximum number of steps (can be defined in the class) the moves are written to a file. Afterwards we have a file which defines in which time step which node moves where.

The new position is evaluated using the mobility model (Also see Interface *MobilityModel*). The mobility has to define in which direction and which amount a node has to move.

2.2.3 Simulating Mobile Routing Algorithms

For simulating in visualization mode the following command has to be used:

$$SimVizMobile < netInputFile >< algorithm1 >$$

$$[< algorithm2 > [< algorithm3 > [...]]]$$

The simulation framework reads the network from the *netInputFile* parameter and reads the corresponding mobility array. In each simulation step the message is sent one hop. After that, all moves for the current step are read from the array and the corresponding nodes move. Figure 2.1 shows the schema of the mobile simulation framework. It shows the how the different classes work together.

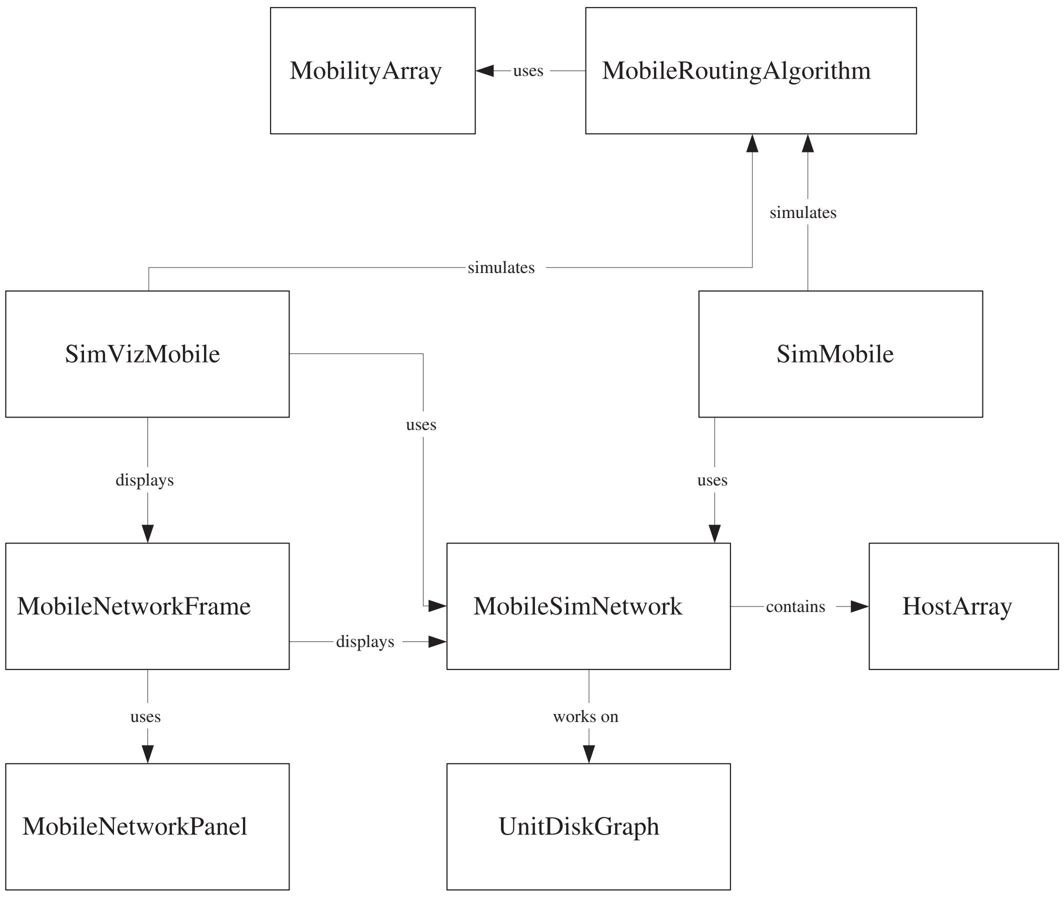

Figure 2.1: Schema of simulation framework

Chapter 3

Simulation Setup

3.1 Model for Simulation

The following model has been used for the simulation of the algorithms:

- The first algorithm reads the network and the mobility array
- It initializes all nodes and tables, such as initial heights or distances
- The algorithm starts and sends a route request over one hop
- All moves of nodes for the current time step are done
- All distance tables and heights which are affected through the moves are updated
- The algorithm sends the route request to the next node (one hop)
- Next moves and updates
- Until the algorithm reaches the destination or reaches an upper limit
- A new sender for the route request is chosen randomly
- 50 consecutive route requests are done before going to the next algorithm
- The network and mobility array are again read from the files
- Do the same for all algorithms

3.2 Simulated Algorithms

3.2.1 Full Link Reversal

The idea behind link reversal algorithms is that every node which is not the destination, directs the links in direction of the destination. When this is done, every message will be routed along the directed edges. If a route from the source to the destination exists, then the message will get to the destination.

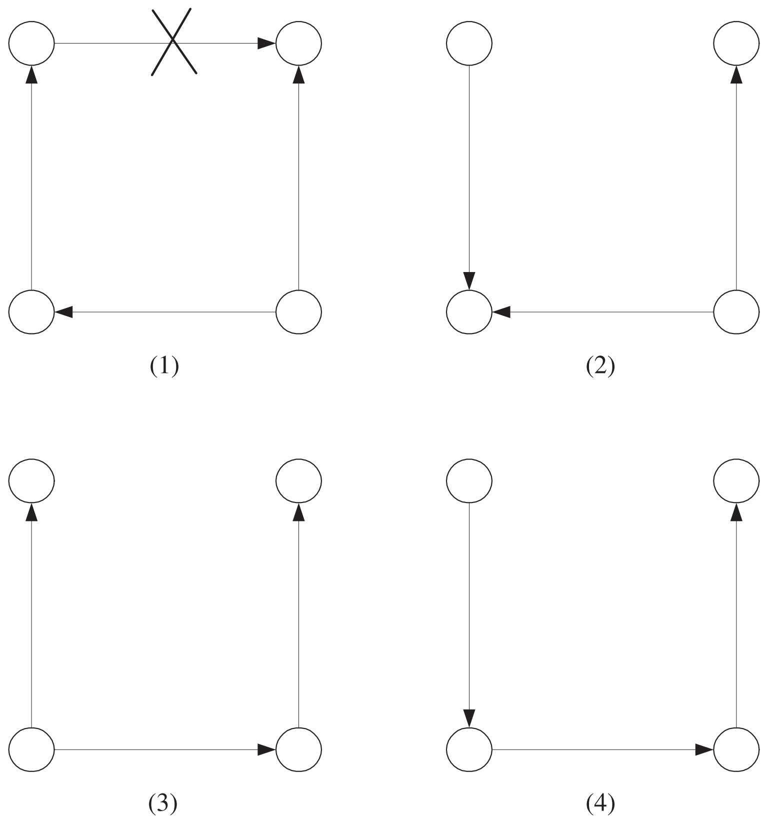

Figure 3.1: Full Link Reversal

Figure 3.1 shows an example of how full link reversal works. We assume that initially all links are directed in direction of the destination (1). If a link breaks then it may happen that a node has no more outgoing links (1). Such a node is called a sink. To re-initialize the network the following procedure is executed: as long as sinks exist each sink reverses all of its links. The only sink that remains sink is the destination.

The algorithm was implemented using heights. At the beginning each node has an initial height. If a route request arises then the destination sets its height to 0. If a node becomes a sink it rises its height to the height of the highest neighbor + 1. This corresponds to reversing all links.

It was shown that the algorithm always terminates and generates no loops [Busch et al.].

3.2.2 Partial Link Reversal

The partial link reversal algorithm is a specialization of the full link reversal. The concept is the same but the idea is, that unnecessary reversals are avoided by smarter reversals [Busch et al.].

Each node maintains a table. A link is inserted into the table if it is reversed.

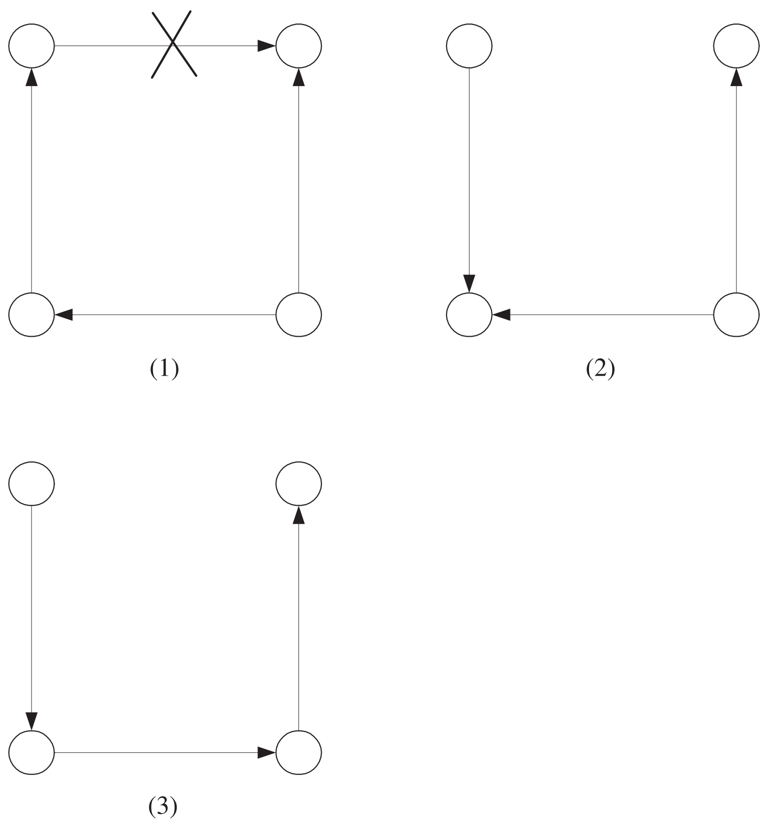

Figure 3.2: Partial Link Reversal

If all links for a node were reversed then the table is emptied and the reversal can start again. As you can see in figure 3.2 this simple change in the full link reversal algorithm enhances it.

3.2.3 Lazy Link Reversal

With full and partial link reversal many reversals happen at points which do not affect the routing request. We tried to avoid such unnecessary reversals by a lazy link reversal routing algorithm.

We define the lazy link reversal as following:

- A routing request is sent along the nodes as long as it can be sent
- When the request comes to a sink, then a full link reversal is done in the whole network

We will see that the overhead is lower than it is with full link reversal.

3.2.4 Distance Vector Routing

The classic distance vector routing is simulated for comparing the other algorithms with. Each node has an entry with the distance to the destination. The distance

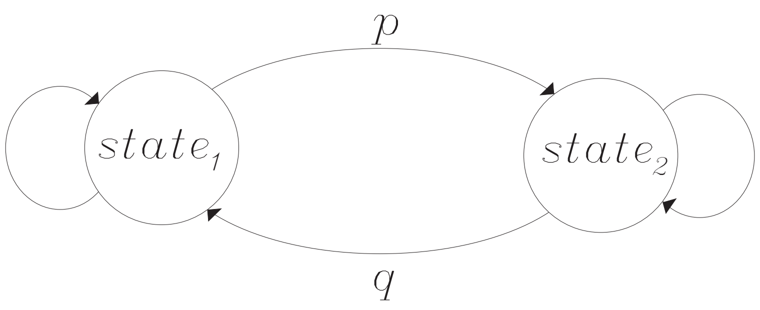

Figure 3.3: Mobility model for simulation

is computed in the following way: the destination has distance 0. For a node n take the neighbor with the lowest distance to the destination and add 1 to it. This gives the new shortest distance for n to the destination.

In a mobile network updating the distances is a costly task. Each node which updates its distance vector has to inform its neighbors. This generates a very high overhead.

3.3 Implemented Mobility Model

After the first test we saw that even a simple mobility model with few moves generates an amount of data which is not easy to handle. Therefor we decided to use a very simple mobility model (see figure 3.3).

Each node is in one of two states:

- $state_1$: the node will not move in the coming round
- $state_2$: the node will move to a new position in the coming round

If the node is in $state_1$ then it will move with probability p from $state_1$ to $state_2$. With probability $r = 1 - p$ the node will stay in its state $state_1$.

If the node is in $state_2$ then it will move with probability q from $state_2$ to $state_1$. It will stay with probability $s = 1 - q$ in $state_2$.

We defined that the following condition shall always hold:

$$p << q$$

.

3.4 Parameters

3.4.1 Net Parameters

The network we used was defined by two main parameters: density and mobility. In our simulation we used unit disk graphs. Therefor the density defines how many nodes are in one unit disk. The density may vary in the whole graph from unit disk to unit disk. Also if nodes are moving the density in disks changes.

Networks with low densities for example are sensor networks where the sensor devices are mobile but have a low transmission radius. Because of the low radius only few nodes are in a unit disk.

In the contrary there exist networks with high densities, for example networks in conference rooms where many people communicate with mobile devices which have a high transmission radius.

For our simulation we used the following density values:

$$\rho = 11 \text{ for low density networks}$$

$$\rho = 20 \text{ for middle density networks}$$

$$\rho = 30 \text{ for middle density networks}$$

$$\rho = 40 \text{ for dense networks}$$

3.4.2 Mobility Parameters

For changing the mobility in the network we used two different parameters:

- The probability to move from $state_1$ to $state_2$ (p), where $state_1$ means no movement and $state_2$ means move in the next step
- The maximum distance to move (ϵ)

As mentioned above we chose $p << q$ (where q is the probability to change from $state_2$ to $state_1$. We set q constant as

$$q = 0.9$$

.

The probability p was set as following:

$$p = 0.02 \text{ for low mobility}$$

$$p = 0.1 \text{ for high mobility}$$

The probability to stay in $state_1$ and $state_2$ is $1 - p$ and $1 - q$ respectively.

The maximum distance factor to move (*epsilon*) was set to the following values:

$$\epsilon = 0.05 \text{ for low mobility}$$

$$\epsilon = 0.1 \text{ for middle mobility}$$

$$\epsilon = 1.0 \text{ for high mobility}$$

The maximum distance to move was calculated the following way:

$$d_{max} = \epsilon * r_{UDG}$$

where r_{UDG} is the radius of the unit disks in the graph.

The direction in which a node moves was chosen randomly (uniformly distributed).

Chapter 4

Simulation Results

4.1 Measured Magnitudes

We were interested in the following two magnitudes.

4.1.1 Path Length

The quality of a routing algorithm first is defined by the path length of the routes which are generated. The goal of each routing algorithm is to use as few nodes as possible for a routing request. If paths are shorter then the energy consumption is lower and fewer collisions happen.

4.1.2 Overhead

The same as for the path length holds for the overhead. In mobile networks it is important that the power consumption is as low as possible. Also the network traffic should be as low as possible such that few collisions happen. Therefor it is essential that as few overhead messages as possible are sent over the net.

In the simulation we measured the overhead which was generated for updating routing tables (heights and distances). We wanted an algorithm that not only provides short routes but also uses as few overhead messages as possible.

4.2 Classification of Scenarios

For comparing how the different algorithms behave on different networks we tried to classify different networks.

4.2.1 $Class_1$: Very Mobile and Dense Networks

This class of networks is defined through the following parameter settings:

$$p = 0.1$$

$$\epsilon = 1.0$$

$$\rho >= 30$$

Where p is the probability to move a certain node, ϵ is the maximum distance factor to move and ρ is the density of the network.

An example for this class of networks is a network with very mobile devices like cars which are moving in a city and are communicating in an ad-hoc manner.

4.2.2 $Class_2$: Very Mobile and Sparse Networks

$Class_2$ networks are defined through the following parameters:

$$p = 0.1$$

$$\epsilon = 1.0$$

$$\rho < 30$$

For this class of networks we can use the same example as for $class_1$ networks. A $class_2$ network may be an ad-hoc network of cars where the density of cars is low.

4.2.3 $Class_3$: Dense Networks with Low Mobility

We can use the following parameters to define this class of networks:

$$p = 0.02$$

$$\epsilon <= 0.1$$

$$\rho >= 30$$

An example for this class of networks would be a full conference room where all members are using a mobile communication device. The members of the conference just move little while the conference is being held.

4.2.4 $Class_4$: Sparse Networks with Low Mobility

Finally $class_4$ networks are defined through the following parameters:

$$p = 0.02$$

$$\epsilon <= 0.1$$

$$\rho < 30$$

Sensor networks are a good example for $class_4$ networks. They are distributed in a large area with few nodes between and they are not very mobile.

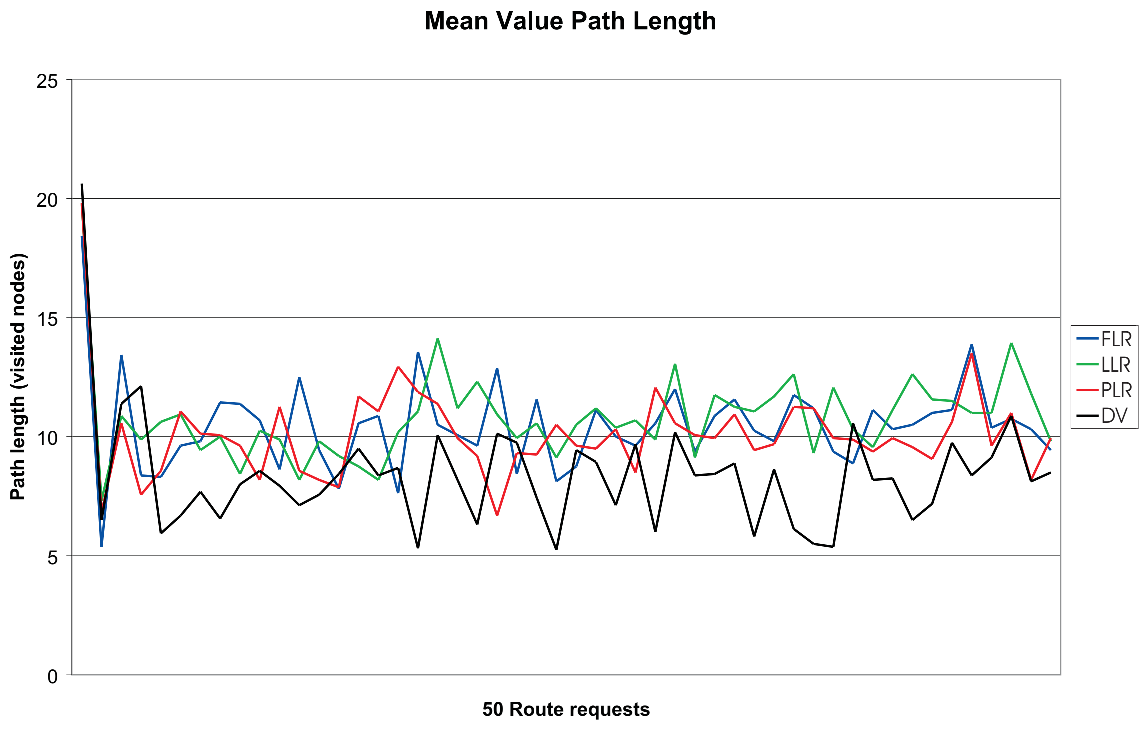

Figure 4.1: Comparison of path length for routing algorithms on $class_1$ networks

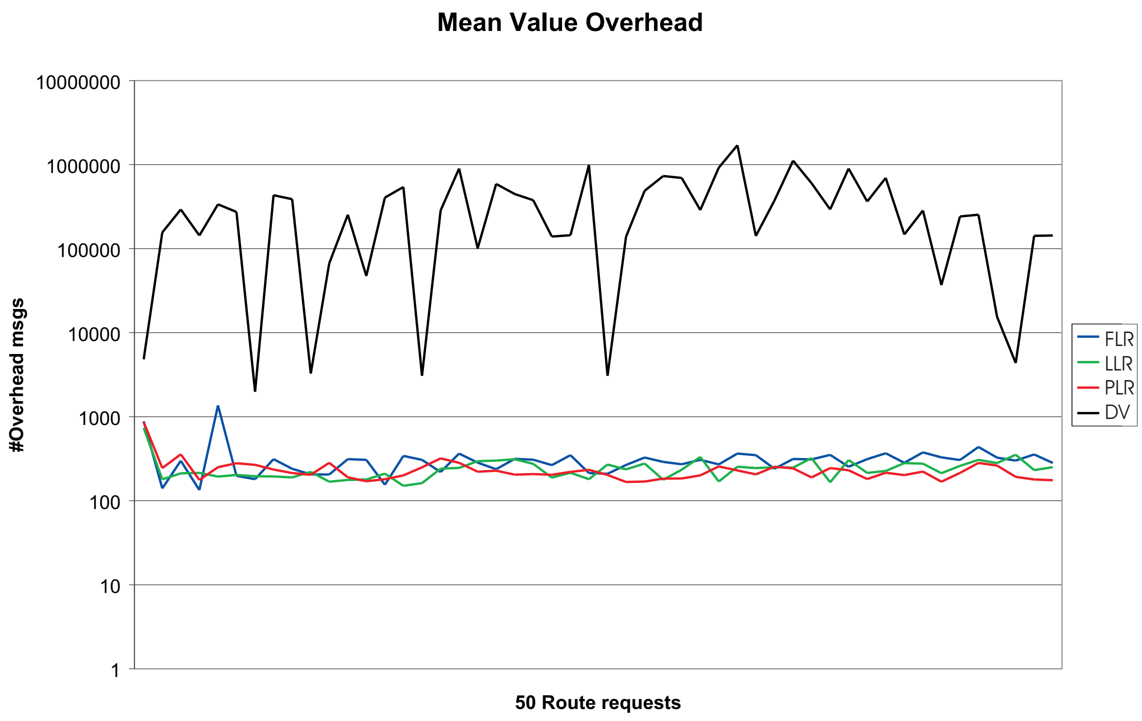

Figure 4.2: Comparison of overhead size for routing algorithms on $class_1$ networks

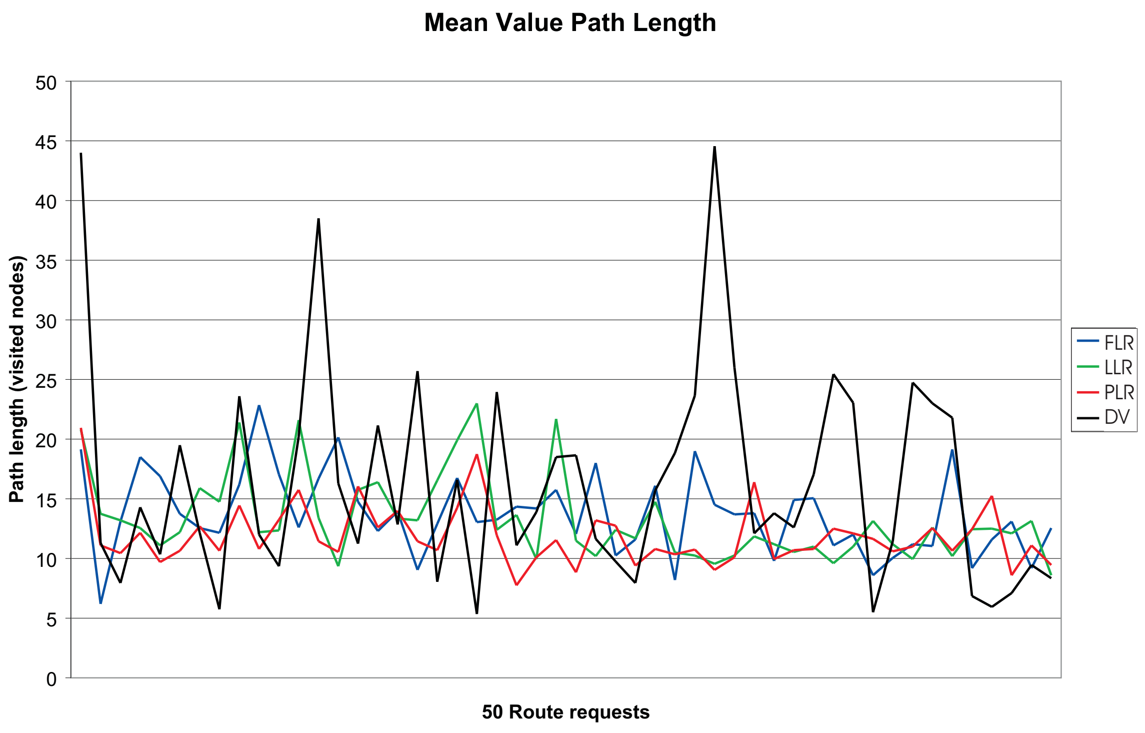

Figure 4.3: Comparison of path length for routing algorithms on $class_2$ networks

4.3 Behavior of Routing Algorithms on Different Classes of Networks

4.3.1 $Class_1$ Networks

Dense Networks with high mobility.

From figures 4.1 and 4.2 we can see that DV routing gives the shortest routes in $class_1$ networks. The other three algorithms deliver more or less equivalent good routes. But the overhead which is generated is, as expected, the highest with DV routing. Then we have full-, lazy- and partial link reversal in this order as expected. Please pay attention that the scale for the overhead is logarithmic. Therefor the overhead for DV routing is very big compared to the other algorithms.

As a result in $class_1$ networks the partial link reversal algorithm should be preferred over full- and lazy link reversal.

4.3.2 $Class_2$ Networks

Sparse Networks with high mobility.

We see in figures 4.3 and 4.4 that DV routing delivers much worse routes than the link reversal algorithms do. This may be due to the following effect: if the network is sparse then few routes exist which minimize the distance. So if a link on such a route breaks then the message has to be routed on another route. Then

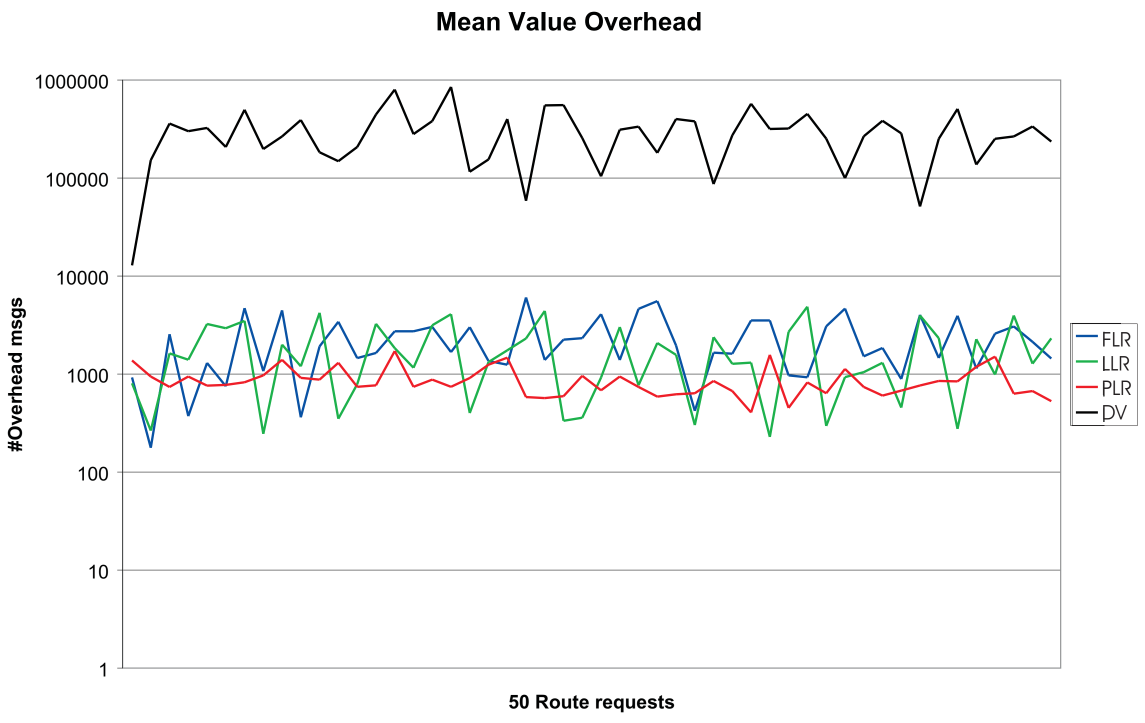

Figure 4.4: Comparison of overhead size for routing algorithms on $class_2$ networks

indirections may arise. Link reversal algorithms in contrary are not fixed on few paths. They can take any path and therefor may have successful routed while DV routing still is searching a path.

The overhead size again is as expected: DV routing very high, full-, lazy- and partial link reversal in this order.

4.3.3 $Class_3$ Networks

Dense networks with low mobility.

The same observations as for $class_1$ networks apply here. DV routing is best, the other three algorithms are more or less equivalent good. Also see figures 4.5 and 4.6.

The overhead again is as expected.

4.3.4 $Class_4$ Networks

Sparse networks with low mobility.

For this class of networks DV routing is the best choice (see figures 4.7 and 4.8). In contrary to the statement that was made for $class_2$ networks, DV routing outperforms the other algorithms because very few moves are made in this class of networks. Therefor DV routing mostly can take the first route that was computed and succeeds.

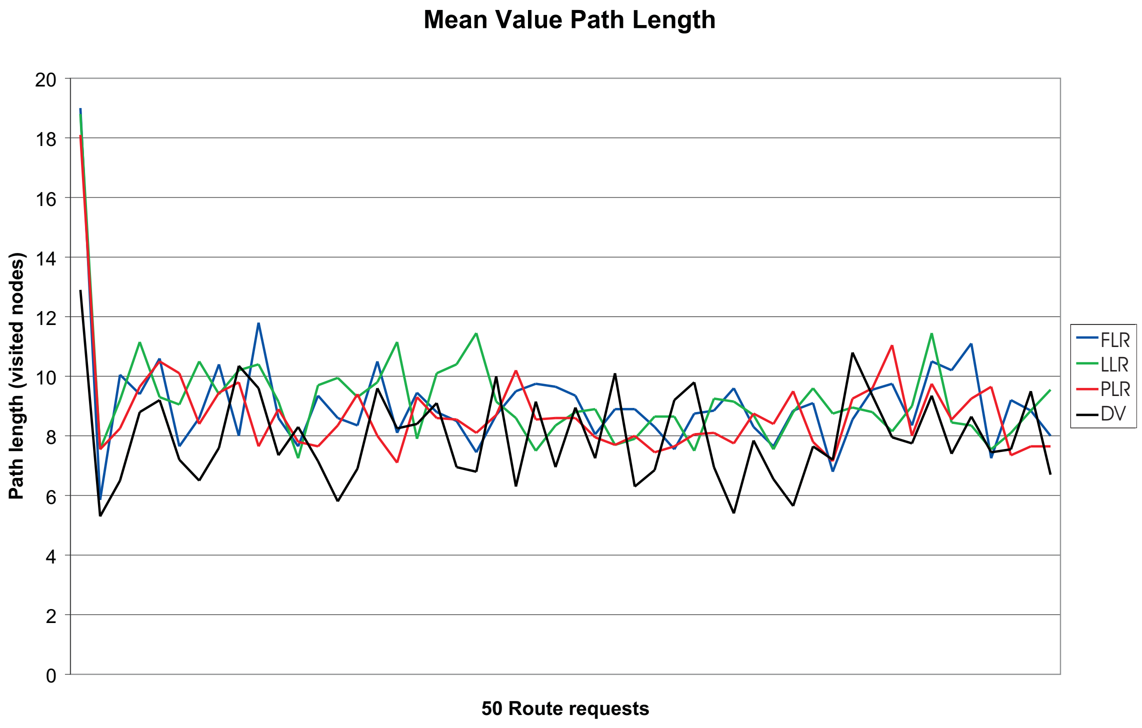

Figure 4.5: Comparison of path length for routing algorithms on $class_3$ networks

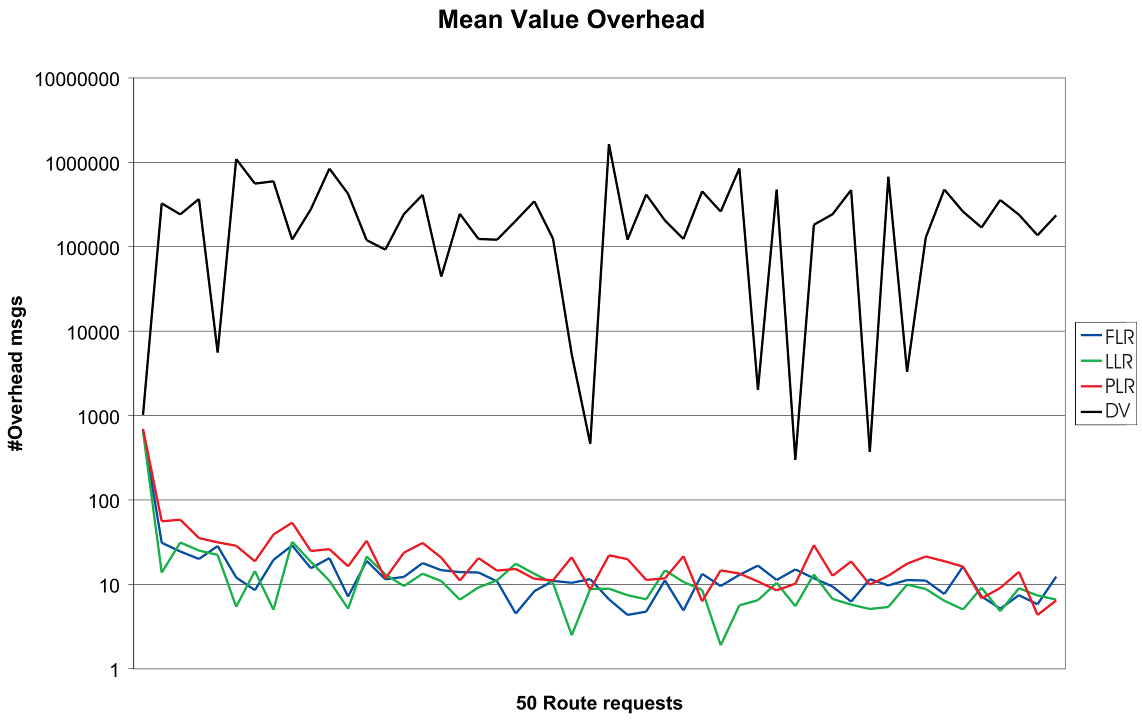

Figure 4.6: Comparison of overhead size for routing algorithms on $class_3$ networks

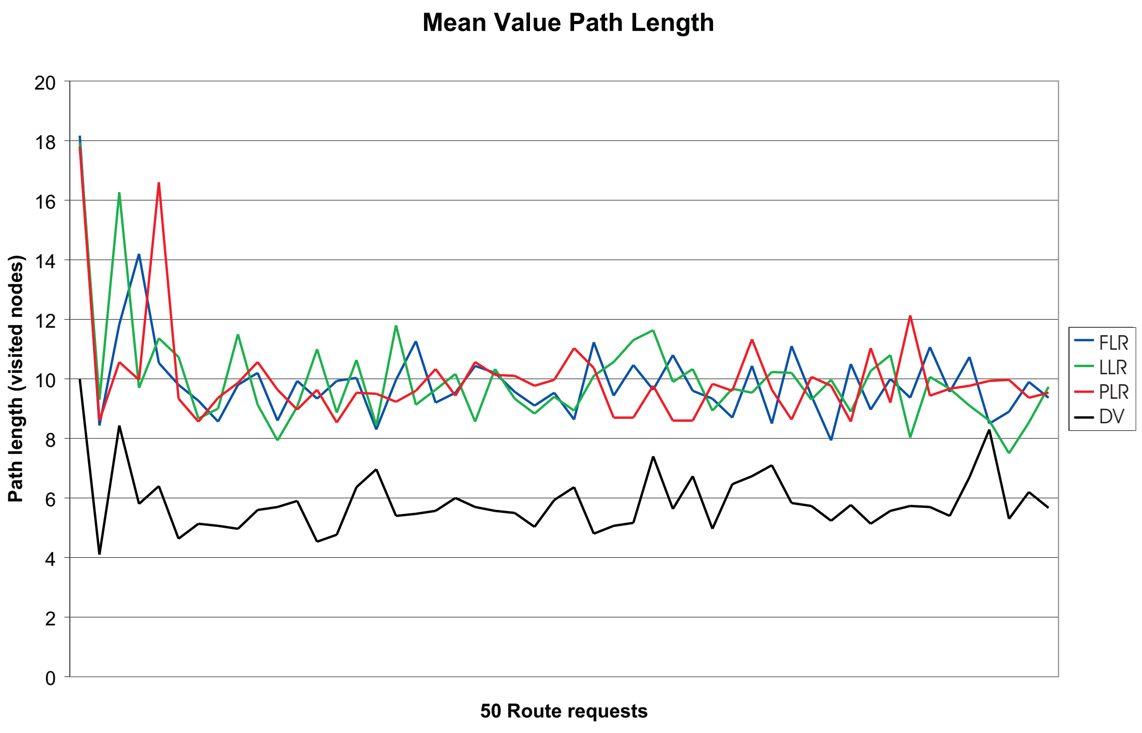

Figure 4.7: Comparison of path length for routing algorithms on $class_4$ networks

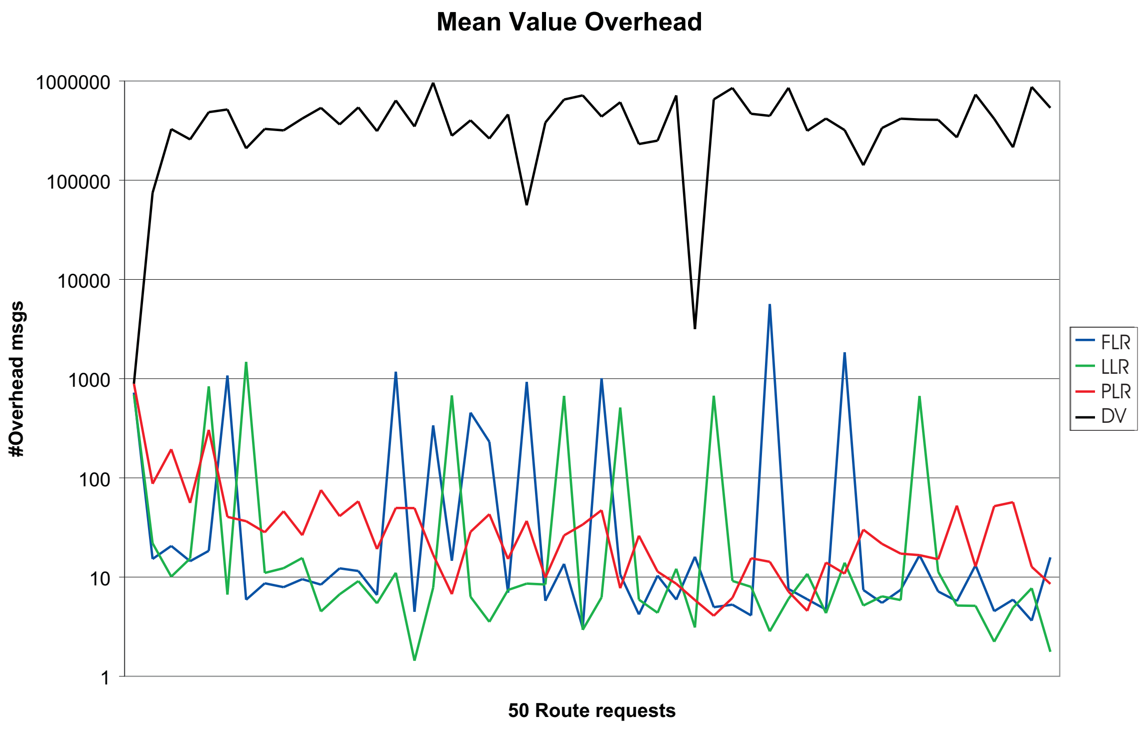

Figure 4.8: Comparison of overhead size for routing algorithms on $class_4$ networks

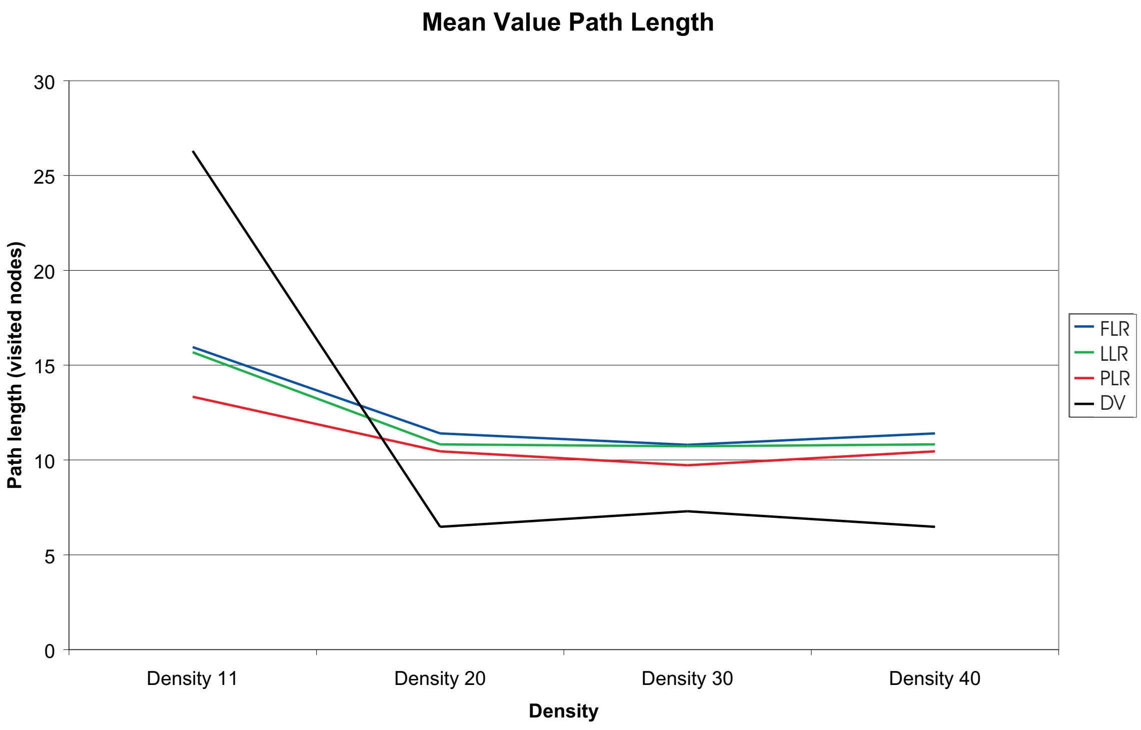

Figure 4.9: Comparison of path length for routing algorithms on networks with different densities

The overhead again is as expected.

4.4 Dependence of Routing Algorithms on Simulation Parameters

4.4.1 Density

As it is shown in figures 4.9 and 4.10 DV routing is worst at low densities as stated above. Obviously there is no other dependence of link reversal algorithms on network density for the simulated densities.

Interestingly the overhead sinks with increasing density for the link reversal algorithms. This may be due to the fact that more routes exist in denser networks and therefor fewer reversals have to be done.

4.4.2 Mobility

As it is shown in figures 4.11 and 4.12 there seems to be no unexpected results. The path length and the overhead both become lower with lower mobility.

We can see from these figures, that no link reversal algorithm becomes better compared to the others with different mobilities.

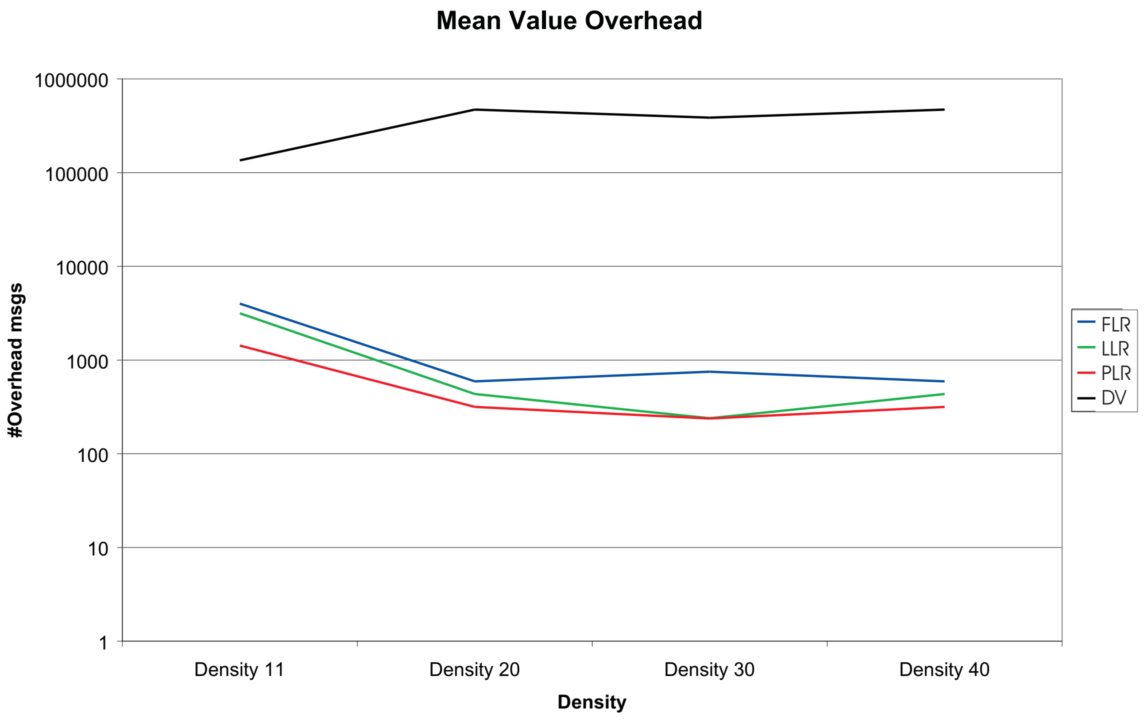

Figure 4.10: Comparison of overhead size for routing algorithms on networks with different densities

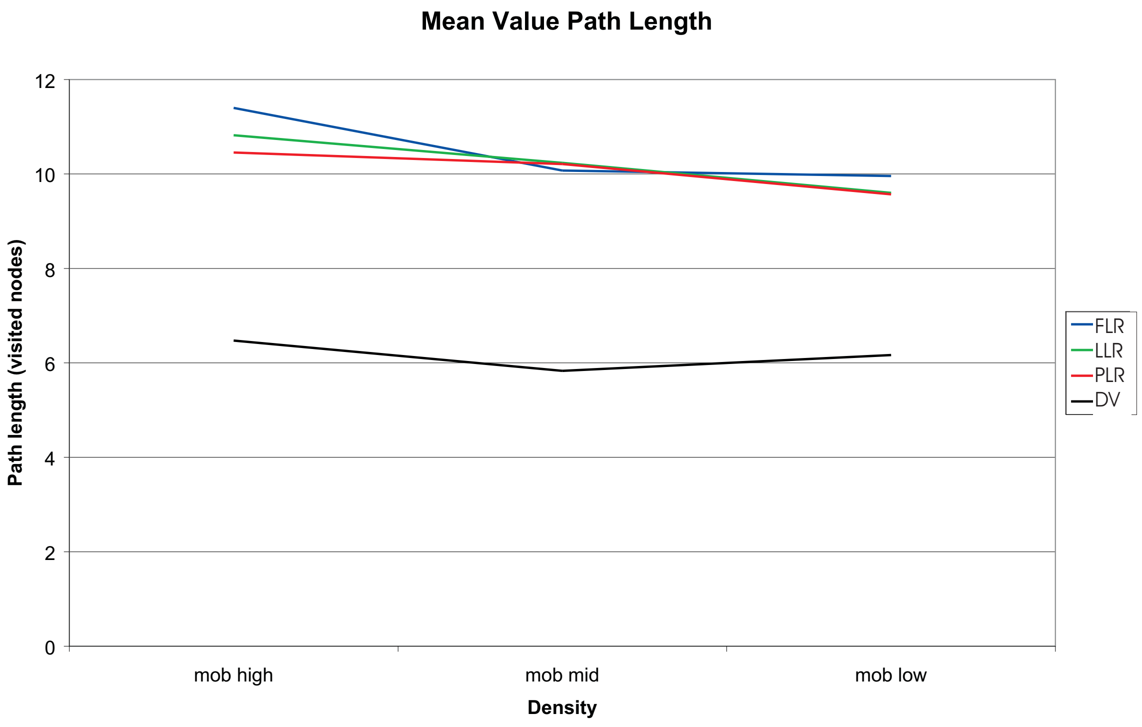

Figure 4.11: Comparison of path length for routing algorithms on networks with different mobilities

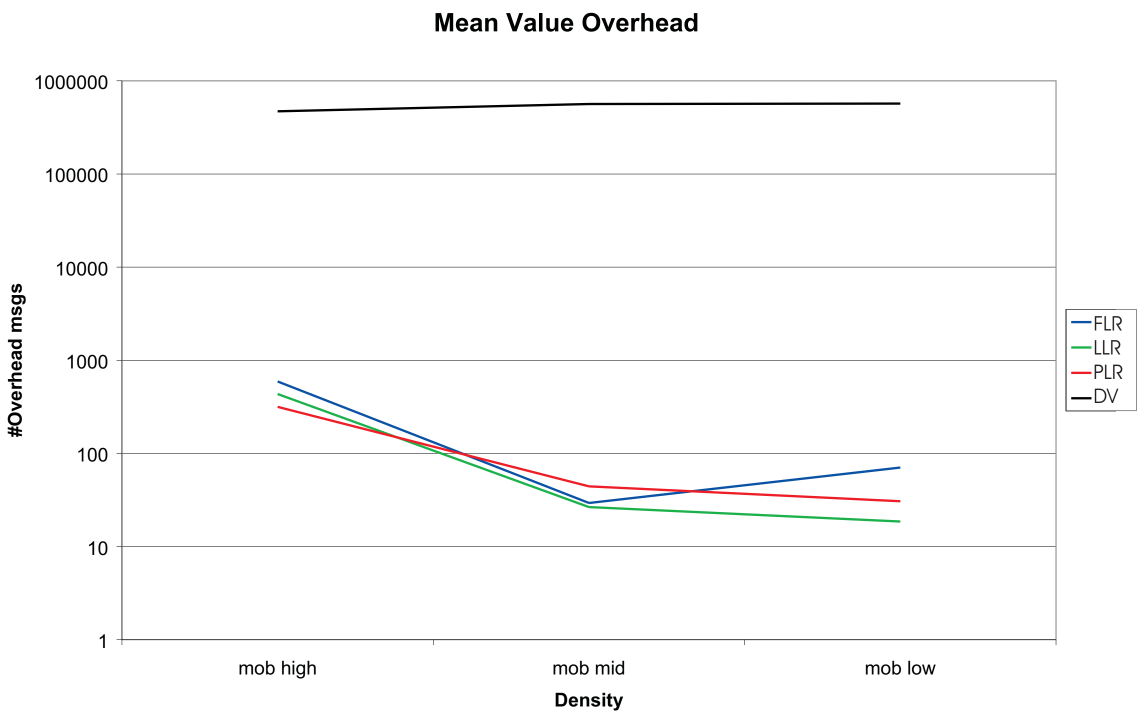

Figure 4.12: Comparison of overhead size for routing algorithms on networks with different mobilities

Chapter 5

Conclusions

5.1 Future Work

Many interesting points remain open. Unfortunately I had not the time to simulate and analyze more algorithms. It would be interesting how other algorithms than link reversal algorithms would behave in the same environments.

If a dependence on either mobility or other network properties could be found one could try to combine different algorithms to form a new one which is best, independent of these properties.

5.2 Own Work

The diploma thesis was announced as a theoretical analysis of different routing algorithms. At the beginning of my work I tried to prove that link reversal algorithms are not as bad as it is presented in [Busch et al.]. Unfortunately I did not succeed. I came to a point where too much parameters influenced the routing protocols. So I was not able to proceed.

Then we decided to do the comparison by simulation. I first enhanced the framework as it is described in chapter 2. After implementing the framework and the algorithms, I prepared the simulation, executed it and analyzed the results.

Looking back on my work I am not very happy with the results I found. Unfortunately I planned my time wrong such that I was not able to analyze the most interesting questions. I needed too much time on the analysis at the beginning. And after that I spent too much time on implementing and testing the framework.

For the next time I will try to plan the whole work at the start of the project and set milestones.

Many thanks to my advisor Regina and to Roger for supporting me.

Bibliography

[Broch et al.] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, Jorjeta Jetcheva: A Performance Comparison of Multi-HopWireless Ad Hoc Network Routing Protocols, Pittsburgh PA.

[Busch et al.] Costas Busch, Srikanth Surapaneni, Srikanta Tirthapura: Analysis of Link Reversal Routing Algorithms for Mobile Ad Hoc Networks, Troy NY.

[Camp et al., 2002] Tracy Camp, Jeff Boleng, Vanessa Davies: A Survey of Mobility Models for Ad Hoc Network Research, Golden CO, September 10, 2002.

[Das et al.] Samir R. Das, Charles E. Perkins, Elizabeth M. Royer: Performance Evaluation of Two On-demand Routing Protocols for Ad Hoc Networks, San Antonio TX.

[Jie Wu, Fei Dai] Jie Wu and Fei Dai: An Extended Link Reversal Protocol in Dynamic Networks, Boca Raton FL.

[Per Johansson et al.] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, Mikael Degermark: Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-hoc Networks, Stockholm.

[Piyush Gupta, P. R. Kumar] Piyush Gupta and P. R. Kumar: A System and Traffic Dependent Adaptive Routing Algorithm for Ad Hoc Networks.

[Shah-An Yang, John S. Baras] Shah-An Yang and John S. Baras: TORA, Verification, Proofs and Model Checking, Maryland.

[Vincent D. Park, M. Scott Corson] Vincent D. Park, M. Scott Corson: A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks, Maryland.

[Vincent D. Park, Joseph P. Macker] Vincent D. Park and Joseph P. Macker: Anycast Routing for Mobile Services, Washington DC.