Evaluating Multihop Mobile Wireless Networks with Controllable Node Sparsity or Density

by

Keyvan Amiri

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy

Approved, Thesis Committee:

David B. Johnson, Chair
Professor of Computer Science and Electrical and Computer Engineering

Richard G. Baraniuk
Professor of Electrical and Computer Engineering

Christopher Jermaine
Associate Professor of Computer Science

Houston, Texas

December, 2015
ABSTRACT

Evaluating Multihop Mobile Wireless Networks with Controllable Node Sparsity or Density

by

Keyvan Amiri

Simulation is the most widely used tool for evaluating the performance of multihop mobile wireless networks, yet such simulation has so far been limited due to the lack of sufficient wireless mobility models for creating a wide range of different types of network scenarios of mobile nodes moving about for use in protocol simulation. For example, the very commonly used Random Waypoint mobility model can only effectively be used in scenarios with relatively high node density, as attempting to generate sparser scenarios (e.g., trying the same number of nodes in larger and larger spaces) results in scenarios in which the network is frequently or always partitioned, with no possible multihop wireless path between many different pairs of nodes.

In this thesis, I present the design and evaluation of the Random Controlled Sparse (RCS) mobility model, a new dynamic, tunable mobility model that can be controlled to generate a wide range of mobile scenarios with varying levels of node sparsity or density while avoiding network partitions. The model requires only a small set of parameters to define the desired behavior of the scenarios being generated. In generating a scenario, RCS itself internally operates as a separate discrete event simulator, utilizing highly efficient graph and computational geometry algorithms to control the desired sparse behavior and manage the constraints between the motions.
of different nodes. To further improve the performance and scalability of the model, I have also parallelized certain key parts of the scenario generation in the model.

To show the performance of the model in generating scenarios, I have evaluated the running time of the model across a wide range of number of nodes and node densities. I also present an evaluation of the scenarios generated, in terms of metrics such as the average number of neighbors of a node and the average minimum possible path length (hop count) existing between each pair of nodes, demonstrating the range of scenarios that RCS is able to produce. To show the usefulness of the model in revealing protocol behavior, I show the performance of DSDV, a common multihop wireless ad hoc network routing protocol, across a wide range of sparse and dense network scenarios. These results demonstrate that different degrees of node sparsity or density sometimes have surprising effects on protocol performance. Simulations such as these, revealing these types of results, have not generally been possible before due to the lack of suitable mobility models.

Finally, to more fully show the use of the RCS model in evaluating real protocols, I present the design and evaluation of LAMP, the Local-Approximation Multicast Protocol, a new on-demand multicast routing protocol I have designed for mobile wireless ad hoc networks that delivers high performance in both sparse as well as dense scenarios. LAMP maintains high performance by utilizing link-layer unicast transmissions, based on a new algorithm in which each node computes a local approximation of the globally optimal multicast forwarding tree to the receivers. LAMP also introduces a new distributed protocol optimization known as anticipatory forwarding, to further improve both overhead and packet delivery latency when this local approximation deviates from the globally optimal tree. I have evaluated LAMP through detailed ns-2 simulations using scenarios from the RCS model as well as the Random Waypoint model, and compared it with ODMRP and ADMR, two existing on-demand multicasting protocols that have previously been shown to perform well.
Acknowledgments

I would like to sincerely thank my PhD advisor, Professor David B. Johnson, for all his amazing guidance and help throughout my years of graduate studies at Rice University. As both my Masters and PhD advisor, Dave has provided me with extremely detailed advice through every single piece of my research work, and mentorship to become a well-trained engineer and scientist. But what I learned from him goes well beyond the technical contributions of this thesis. I learned how to become a critical thinker and how important minor details are in different stages of research development. Through his guidance, I learned how to succinctly and effectively present my work and communicate, which I’m confident I will be able to use for the rest of my career.

I would also like to thank the rest of my thesis committee, Professor Richard Baraniuk and Professor Christopher Jermaine. Through their valuable insight and feedback, I was able to improve the presentation of my work and fine-tune different aspects of my thesis.

Finally, I would like to thank my loving family. My mom, Zahra Shadmani, who has always dedicated herself to support me at every single stage of my life; my dad, Mansour Amiri, who has consistently pushed me to better myself as a person; they both have provided me with all the support and unconditional love I could ever wish for. And finally, I’m greatly indebted to my brother, Kiarash Amiri, who has been like a life mentor to me. He has helped me through many obstacles in life and given me great advice that has helped me immensely with my career.
## Contents

Abstract  
Acknowledgments  
List of Illustrations  

1 Introduction  

2 Related Work  

2.1 Trace-Based Mobility Models  
2.2 Synthetic Mobility Models  
2.3 Other Related Work  

3 RCS Mobility Model Design  

3.1 The Failure of Current Mobility Models  
3.2 RCS Model Overview  
3.3 Dividing Space into Wedges  
3.4 Density Metric and Outer Wedge Boundary  
3.5 Selecting a Wedge  
3.5.1 Rationale for the Wedge Selection Algorithm  
3.5.2 Probability Distribution of Wedge Selection  
3.6 Additional Model Features  
3.6.1 Dealing with Node Vibration  
3.6.2 Encouraging Additional Topology Changes  
3.7 Chapter Summary  


4 RCS Implementation Challenges

4.1 General RCS Implementation Structure ........................................... 27
4.2 Scheduler and Description of Events ................................................. 28
4.3 Intersection Algorithm ...................................................................... 32
  4.3.1 Intersection of a Line Segment and Circle .................................... 35
  4.3.2 Checking the Inclusion of a Point on a Segment ............................ 36
  4.3.3 Intersection of a Circular Arc and a Circle .................................. 36
  4.3.4 Checking the Inclusion of a Point on a Circular Arc ..................... 37
  4.3.5 Checking whether a Point Lies within a Sector ............................ 39
4.4 Floating Point Considerations ............................................................ 44
  4.4.1 Comparison Between Two Floating point Numbers ....................... 44
  4.4.2 Loss of Significance ..................................................................... 45
4.5 Chapter Summary .............................................................................. 47

5 Evaluation of the Model and Implementation ........................................ 48

5.1 Evaluation of the Model in terms of Mobility Behavior and Revealing
  Protocol Behavior .............................................................................. 48
  5.1.1 Network Topology Results .......................................................... 51
  5.1.2 Example Routing Protocol Performance Evaluation ..................... 54
5.2 Asymptotic Complexity of the Sequential Version .............................. 57
5.3 Experimental Evaluation of the Sequential Version ............................ 59
5.4 Parallel RCS Implementation ............................................................. 61
  5.4.1 Parallel Computation of the Vibration Checks .............................. 62
  5.4.2 Parallel Computation of Floyd-Warshall Algorithm ....................... 64
5.5 Experimental Evaluation of the Parallel Version ............................... 66
5.6 Chapter Summary .............................................................................. 68

6 Protocol Design and Performance Study using RCS ............................... 70

6.1 Overview of Ad Hoc Network Multicast Protocols ............................... 72
6.2 LAMP Protocol Design .............................................. 73
6.3 A Local Approximation of the Optimal Multicast Tree .......... 76
   6.3.1 Upper Bound Analysis ......................................... 77
   6.3.2 Solving the Local Approximation Problem ................. 80
6.4 Maintaining Unicast Routing Information .......................... 82
   6.4.1 UID Mechanism ................................................. 84
   6.4.2 Adding Loop-free Routes ...................................... 85
6.5 Distributed Optimization ............................................ 91
6.6 Evaluation Methodology ............................................. 93
6.7 Simulation Results .................................................. 94
6.8 Chapter Summary ................................................... 101

7 Conclusion .................................................................... 105

Bibliography .................................................................. 107
Illustrations

3.1 Snapshots of Random Waypoint model for 670x670 m² .......... 13
3.2 Snapshots of Random Waypoint model for 2500x2500 m² .......... 13
3.3 Snapshots of a sparse scenario generated by the RCS mobility model 13
3.4 Partitioning by wedges vs. partitioning by grid cells ............... 15
3.5 Wedge boundaries adjacent to A, and the intersection of other nodes’
radio ranges with this wedge ......................................... 17
3.6 Probability density functions of wedge selection for various density
control parameter values .............................................. 20
3.7 Node C currently acts as a bridge between nodes A and B and can
only “vibrate” in the tiny intersection region, shown as shaded, in
order to avoid partitions .............................................. 23
3.8 A spends a long cumulative time of being leaf as a result of network
becoming static; it gets into leaf-mode by pairing up with a random
node D. Two nodes then start moving towards each other ........... 25

4.1 A general intersection region shape between an arbitrary circle and
sector ................................................................. 34
4.2 A general intersection region shape between an arbitrary circle and
sector ................................................................. 41
4.3 A general intersection region shape between an arbitrary circle and
sector ................................................................. 42
5.1 Partitioning time metrics for Random Waypoint model of 50 nodes as a function of simulation area size; The x-axis is the side length of a square-sized simulation area ........................................ 51
5.2 Average node degree vs. simulation time ........................................ 52
5.3 Average path length vs. simulation time ........................................ 53
5.4 First set of performance graphs for DSDV; generated by RCS for different density levels, and RWP for different area sizes .............. 56
5.4 Second set of performance graphs for DSDV; generated by RCS for different density levels, and RWP for different area sizes .............. 57
5.5 Running time of the sequential scenario generator for a simulation time of 1000 seconds, across different range of density levels and number of nodes ......................................................... 61
5.6 The execution path of the scenario generator before, while, and after performing vibration checks in parallel ........................................ 65
5.7 Row-wise decomposition of the distance matrix to be computed by separate threads concurrently; the kth row is the additional local data needed by each thread .............................................. 66
5.8 OpenMP-based implementation of parallel Floyd-Warshall algorithm 67
5.9 Running time of the parallel scenario generator for a simulation time of 1000 seconds, across different range of density levels and number of nodes ................................................. 68

6.1 Sender node S wants to multicast packets to a multicast group containing receivers R1 and R2 .................................................. 74
6.2 Multicast data packet forwarding in LAMP ................................. 75
6.3 Adding link C–E would create a loop ........................................... 86
6.4 Anticipatory forwarding of overheard packets ............................... 92
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5</td>
<td>Packet Delivery Ratio for multicast protocols for sparse and dense scenarios generated by RCS</td>
<td>96</td>
</tr>
<tr>
<td>6.5</td>
<td>Packet Delivery Ratio for multicast protocols for sparse and dense scenarios generated by RWP (different area sizes)</td>
<td>97</td>
</tr>
<tr>
<td>6.6</td>
<td>Packet Delivery Latency for multicast protocols for sparse and dense scenarios generated by RCS</td>
<td>99</td>
</tr>
<tr>
<td>6.6</td>
<td>Packet Delivery Latency for multicast protocols for sparse and dense scenarios generated by RWP</td>
<td>100</td>
</tr>
<tr>
<td>6.7</td>
<td>Total routing packets for multicast protocols for sparse and dense scenarios generated by RCS</td>
<td>102</td>
</tr>
<tr>
<td>6.7</td>
<td>Total routing packets for multicast protocols for sparse and dense scenarios generated by RWP</td>
<td>103</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Simulation is the most common tool for evaluating the performance of multihop mobile wireless networks. For example, simulation allows easy evaluation of new protocol designs and can quickly compare different design alternatives. Although protocol experimentation in a testbed of real hardware is also a valuable evaluation technique, the expense and complexity of fielding large amounts of equipment may be prohibitive, and testbed evaluation cannot generally produce the kind of detailed repeatable experiments, possible via simulation, that are necessary for specific performance comparisons.

However, multihop mobile wireless simulation has so far been limited due to a lack of relevant wireless mobility models to create different scenarios of mobile nodes moving about. For example, using real-word traces, the performance study would become very limited; traces are very expensive and difficult to generate both because of the logistics equipment they require as well as the inaccuracy of such collected traces (Chapter 2). More importantly, such traces do not represent the actual node movement participating in an ad hoc network. The movement of users in these network is influenced by the desire of each user to remain near at least one access point, as opposed to having the motivation to maintain a non-partitioned network with other mobile users. Using the current synthetic models, on the other hand, has its own limitations as well. For instance, with the widely used Random Waypoint mobility model, it is not possible to generate scenarios with sparse distributions of
mobile nodes, and can only effectively be used in scenarios with relatively high node density. Placing the same number of nodes in larger and larger spaces (or fewer and fewer nodes in the same size area) results in scenarios where the network is frequently or always partitioned, with no possible multihop path between many different pairs of nodes. As such, the results from evaluating, for example, ad hoc network routing protocols in such scenarios does not generate generally useful results; although in some real applications, the mobile users may become partitioned, in many real-world applications of ad hoc networks, such as in military or emergency disaster relief efforts, the users will have a reason to cooperate and try to avoid such partitions to keep the network functioning and serving its purpose.

In my thesis, I present the design and evaluation of a new dynamic, tunable mobility model, called the Random Controlled Sparse (RCS) mobility model, a new dynamic mobility model that can be controlled to generate a wide range of mobile scenarios with varying levels of node density while avoiding network partitions. My goal is to be able to create mobile scenarios that expose previously unexplored areas of wireless protocol performance, particularly for multihop routing protocols. The RCS model internally operates as a discrete event simulator, utilizing highly efficient graph and computational geometry algorithms to control the desired sparse behavior and manage the constraints between the motions of different nodes. Different complex components of the model is parallelized, despite being an inherently sequential model. This enables the model to generate even large-scale scenarios within a short amount of time (scenarios of constantly moving nodes with an average speed of 10 m/s consisting of 200 nodes, are generated in less than 6 minutes on a 12-core Intel Xeon machine running at 2.50 GHz). The internal algorithms are designed to be robust in the presence of imprecise machine arithmetic operations, especially since we are dealing
with computational geometry algorithms, where inaccurate machine calculations can hinder the algorithm’s ability to produce valid results.

I evaluate the model itself as well as its implementation performance in generating mobile scenarios. Then, I demonstrate the sometimes surprising performance results that different degrees of node density have on example ad hoc network routing protocols; simulations such as these, revealing these types of results, have not generally been possible before due to the lack of suitable mobility models. In particular, protocol performance in sparse networks significantly drops in comparison with the dense ones.

Finally, to more fully show the use of the RCS model in evaluating real protocols, I present the design and evaluation of the Local-Approximation Multicast Protocol (LAMP), that performs well in both sparse and dense scenarios. I have evaluated LAMP through detailed ns-2 simulations of connected mobile ad hoc networks with dense and with sparse average node densities, generated by RCS. Compared with ODMRP [23] and ADMR [17], two existing on-demand multicasting protocols that have previously been shown to perform well [17], LAMP generally equaled or outperformed both protocols for dense as well as sparse scenarios on all metrics. For example, in sparse scenarios with continuously moving nodes, LAMP achieved a packet delivery ratio (PDR) of over 99%, while ODMRP achieved only around 90–95% PDR and ADMR achieved only 65–85% PDR; in dense scenarios with 3 groups, LAMP’s overhead was about 3–4 times lower than for ODMRP and ADMR. The understanding and evaluation of LAMP and other protocols across such a range of types of wireless multihop network topologies, including dense as well as sparse topologies, has not previously been possible before the development of the Random Controlled Sparse (RCS) mobility model.
Chapter 2

Related Work

Various mobility models have been proposed in the context of mobile wireless networks. Current models can be divided into two general categories; traced-based and synthetic mobility models.

2.1 Trace-Based Mobility Models

In trace-based model, the mobility track of actual mobile devices are replayed. These traces can be obtained in several ways. Global Positioning System (GPS) devices captures the location information of nodes by processing satellite data. The challenges of this method lie in the difficulties of both collecting the traces and evaluating wireless networks using these traces.

Regarding the trace collection phase, GPS devices, for instance, work only in outdoor environments, and are prone to approximation errors in the presence of obstacles such as buildings, leaves of trees. Additionally, to capture location information of mobile devices, they need to be equipped with special software that are constantly in communication with access points (AP)s. Thus, they could potentially pose practical limitation related to the battery life of mobile devices.

Alternatively, these traces could be captured by relying on the information provided by GSM Base Station (BS) [27] or Wi-Fi access points (AP) [22]. The mobility traces in this case are collected by processing either the GSM or Wi-Fi beacons. By
measuring the received signal strength from AP or BS, the mobile device can infer its approximate location. However, this method has its own difficulties and challenges as well.

One of these issues is the low accuracy of location information obtained from the traces. This is because the location obtained by this method is often estimated from the location of which access point each mobile user is associated to [4, 11, 25, 13]. The accuracy could be low due to the fact that each mobile user could be anywhere within the coverage range of each associated AP. When replaying the mobility trace and simulating wireless propagation of packets for the protocol being studied, small errors in location that are part of the trace may have a large effect on which packets are received or not.

The other challenge lies in the nature of wireless signals themselves that can experience multipath fading both in indoor and outdoor environments. For example, the signals from the APs can be obstructed by buildings and trees; as such, a wrong estimate could be produced when signal strength is considered as a way of measuring distance from an AP. Also, mobile devices have different criteria in choosing APs, and the geographically closest AP may not be the desired chosen one; additionally, users can change their associations at different rates resulting in some mobile devices potentially being far from their associated AP.

Indoor localization becomes even more challenging due to the existence of various obstacles such as walls and human beings in a more compact space. GPS which is widely used in outdoor environments does not work indoors due to the obstructed line of sight between satellites and objects inside a building. Various indoor localization technologies have been proposed which are overviewed in several articles [10]. Several emission sources have been proposed within this context. In Infrared (IR) positioning
systems, infrared signals are emitted from wired or wireless devices to detect the location of the objects, which require line of sight communication. For instance, Active Badge system [37], can localize devices in a room by having each user carry an active badge, and periodically transmitting IR signals, which are then received by a set of fixed sensors installed in the room. These sensors can infer the location of the user by communicating the location information each one received from the user.

Ultrasound signals have also been used as another way of locating users inside a room. In AT&T’s Active Bat [38], for example, a set of receivers are mounted across the ceiling at predefined locations. A short pulse of ultrasonic signal is then emitted from the transmitter attached to each object (a Bat). Using the time-of-flight lateration technique to multiple receivers, the location of the object can then be estimated within 3 cm. Cricket [29] provides the localization at each object locally; Objects receive the ultrasound signals emitted from a set of emitters attached to the walls or ceiling, and perform position triangulation locally, with a position estimation accuracy of 10 cm. These techniques requires a large number of receivers across the ceiling, which are expensive to install, and their placements need accurate alignment.

Radio signals can also be used for indoor localization. Using Radio Frequency Identification (RFID) technology, each user carries an RFID tag which acts as a transceiver for transmitting different information. For example, in WhereNet [39], such information is received by a number of deployed location antennas mounted on the ceiling, which in turn forward the information to a location processor for inferring the location of the user. Microsoft’s RADAR [2] is another famous indoor localization technique which uses radio signals. In RADAR, the signal strength from different access points is surveyed and recorded at multiple receiver location cells. The user’s location can then be triangulated by mapping the observed signal strength and
matching it with one of those predefined samples. The reported accuracy of RADAR systems is about 2.9-4.3 m (at the 50th percentile). Bluetooth radio signals have also enabled indoor localization used in position systems such as Topaz [34].

All these localization techniques will still incur some error range (higher than GPS-level accuracy) which could be quite significant in terms of the radio behavior. This can in turn make them impractical to be used in trace-based models for evaluating wireless protocol performance. Also, regardless of the technique used to collect the traces, trace-based models have limitations in terms of the scope of evaluation. This limitation is the result of the limited number of traces that can be obtained from an environment. The evaluation is thus only valid within a limited set of scenarios, that are the specific set of conditions (such as the time of the day) under which the traces were recorded. Furthermore, traces cannot be tweaked or aggregated to reconstruct other traces; hence can only be used as they are, in their limited quantity.

2.2 Synthetic Mobility Models

The other category of mobility models is synthetic models which attempt to capture the real mobilities by generating random movements, rather than depending on real mobility traces. The models under this category differ in terms of how much abstraction is used to generate the movements. One type of models under this category are domain-specific models. These models attempt to mimic what real behavior of mobile user would be by capturing the details of movement in each specific environment. For example, in Freeway and Manhattan mobility models [3] try to mimic the movement in an urban area; motion is restricted to pathways defined by grids of horizontal and vertical streets. In obstacle mobility models [15, 16], obstacles of a real terrain are placed into the model, and the movements trajectories and radio propagation is af-
ected by them. Disaster area mobility models [1, 35] provides a model that mimics movement of agents in disaster area scenarios. These moving agents are typically vehicles and rescue agents.

One major limitation is domain-specific models is that they require detailed analysis of mobility data about users in an environment, and can thus be quite complex. Such analysis requires capturing detailed mobility patterns of for instance vehicles or pedestrians during different time intervals of the day. It’s very difficult to verify the level of realism achieved by these models since collecting such high-detailed traces that captures microscopic behavior of users which is very difficult if not infeasible. Moreover, these models are tuned to specific environments during specific time of the day; hence a new model needs to be developed for each unique environment, or even the same environment but different time periods.

The other subtype of synthetic models that are more abstract are random-based models. These models abstractly model general mobility by random generation of node movement, rather than basing it on detailed parameter settings of environments. They are thus not limited to any specific environments or circumstances, and so are more generally and easily usable, but may not capture essential attributes of node mobility that may be necessary in some simulation-based evaluations.

The most commonly used model of this category (especially in performance evaluation of ad hoc networks) is Random Waypoint model [7]. In this model, nodes randomly select destinations (known as waypoint) in a simulation field, and start moving towards the waypoint with a random speed. Upon reaching the destination, they pause for a random duration known as 'pause time'. They repeat this process over the simulation lifetime. The two other variants of this model are Random Walk and Random Direction [30] models. In Random Walk model, node pick new a ran-
dom speed and direction at each time interval, which mimics the *Brownian Motion* in physics. Random direction model, on the other hand, was proposed to address the non-uniform distribution of nodes across the simulation field. Rather than choosing a random waypoint, a uniformly random direction is picked by each node; the node travels along the chosen direction until it reaches the boundary of the simulation field; which then pauses for a random time and follows the same process again. There are other flavors of Random Waypoint model, that try to put either temporal or spatial dependencies on movement of nodes to represent some more fined-tuned behavior of mobile users. For the temporal dependent models, the movement of nodes is affected by their previous movement. An example is the Gauss-Markov model [24] in which the velocity of a mobile node is correlated over different time slots, and is modeled as a Gauss-Markov stochastic process. Smooth Random Mobility model [5] is another example which takes a more detailed approach regarding the temporal dependency of velocity over different time slots. The speeds are changed smoothly according to the node acceleration/deceleration value.

In spatially correlated mobility models of this group, the mobility of a node is affected by the motion of its surrounding nodes. The *reference point group mobility model* (RPGM) [12] is proposed to model the movement of mobile nodes following group leaders. Each group of nodes has a group leader or a reference point, whose motion determines the movement trend of all its group members. The movement of group members deviates from this reference point by a randomly generated vector of a maximum radius added to the position of the reference point. There are several proposed variants of this model, such as *column model, pursue model, and nomadic-community* models [32] that describe movement of nodes either in a group or following a special target.
Regardless of how the movement is generated (using trace-based or synthetic models), existing models fail to take node connectivity or partition into consideration, making them effectively unusable in many types of simulation-based evaluations for scenarios with sparse node mobility. Unlike these existing models, the new random controlled sparse mobility model (RCS) presented here is able to generate mobility scenarios over a wide range of node sparsity or density while avoiding network partitions. Furthermore, with RCS, even in extremely sparse scenarios, the individual nodes continue to actively move, creating dynamic network topology changes over time. The RCS mobility model enables new, previously unexplored avenues of protocol evaluation in such sparse scenarios, revealing protocol behaviors not seen before.

2.3 Other Related Work

There has been some work in the literature for the relationship between the density of the network and the connectivity of the network graph. One notable example of such work is [6], in which a minimum range is sought in order to create an almost surely $k$-connected network from a random uniform distribution of the nodes. This paper, however, derives an analytical solution for relating the minimum node degree (rather than the connectivity of the network graph) to the minimum required radio range. This analytical solution is merely just a rough approximation of the connectivity of the network graph. As witnessed by the simulation results in the paper, this approximation does not represent the actual connectivity of the network graph very well.

Xue and Kumar [42] derived an analytical solution for the minimum number of neighbors each node should have in order to make the resulting network graph connected. The analytical solution however is an asymptotic bound which can only
be useful when the number of nodes approaches infinity, and cannot give an accurate estimate for realistic networks of limited number of nodes.
Chapter 3

RCS Mobility Model Design

The Random Controlled Sparse mobility model (RCS) is a new dynamic, tunable mobility model that can be controlled to generate a wide range of mobile scenarios with varying levels of node sparsity or density while avoiding network partitions. The model requires only a small set of parameters to define the desired behavior of the scenarios being generated. This chapter details the design of the RCS model.

3.1 The Failure of Current Mobility Models

Existing mobility models fail to produce connected scenarios as the network topology becomes sparser. For example, Figures 3.1 to 3.3 show snapshots of the network topology at 3 different times in a 1000 s simulation: time 0, time 500 s, and time 1000 s. Figure 3.1 corresponds to a commonly used dense scenario size of 670 by 670 used in the simulation of mobile ad hoc networks. This average node density is known to have very low probability of partition, although the hop-count distances between nodes are generally very short, mostly with only 1 or 2 between any pair of nodes. In an attempt to create sparser networks, we increased the simulation area to 2500 by 2500, again with 50 nodes. Figures 3.2a to 3.2c show the configuration of nodes at the same 3 times. The network now, although sparse, there are large number of partitions in the network. In contrast, Figures 3.3a to 3.3c show the network topology from RCS model using a density control parameter setting of 0.1 (explained in...
Figure 3.1 Snapshots of Random Waypoint model for 670x670 m²

Figure 3.2 Snapshots of Random Waypoint model for 2500x2500 m²

Figure 3.3 Snapshots of a sparse scenario generated by the RCS mobility model
Section 3.2), resulting in a very sparse average node density; although sparse, the network topology still continues to change and remains dynamic.

### 3.2 RCS Model Overview

This section provides an overview of the new Random Controlled Sparse mobility model (RCS). In the remainder of this chapter, I then detail different aspects of the model’s design in the following sections.

Initially, nodes are distributed according to an input *density control parameter*, chosen from \([0, 1]\), that governs how much node sparsity/density is desired in the generated mobility scenario. This distribution approximates the overall spatial distribution of nodes over the simulation time. Each mobile node then independently picks a random waypoint, similar to Random Waypoint mobility, but limited to inside a region of simulation space according to the density control parameter. The node then travels towards this chosen waypoint with a uniformly random speed. The node either reaches its chosen waypoint or stops earlier in case any further movement causes a partition. This process is repeated over the scenario simulation time being generated. The following steps summarizes the operation of RCS, with each step detailed in the following sections:

1. Divide space around the node into equi-angular regions referred to as *wedges*.
2. Calculate a *density metric* for each wedge.
3. Select a wedge according to this metric and the desired input density control parameter.
4. Choose a uniformly random waypoint \((x, y)\) within that wedge.
5. Start traveling towards the chosen waypoint until reaching it or until any further movement would cause a partition.

### 3.3 Dividing Space into Wedges

Before a node picks a new waypoint, it divides the space around itself into sectors, called *wedges*, of equal angle $\theta$; in our implementation, $\theta = 30$, resulting in 12 wedges around the node, starting at the positive $x$-axis. This partitioning by wedges has several key properties that other partitionings lack. First, this partitioning means that a node remains within the area of the desired density throughout its motion toward its waypoint, as opposed to other partitioning, e.g., by grid cells.

For instance, Figure 3.4 shows the difference in the behavior of wedge partitioning vs. partitioning by grid cells, in the exact same scenario. Node $A$ here first selects the area of the right density in which to choose a random waypoint; the location of chosen here is the same in both cases, chosen within the wedge (or grid cell) that contains of the desired density. Using wedges, all motion of node $A$ towards its chosen waypoint...
remains within the area of the desired density, regardless of whether the node reaches the waypoint or not, but using grid cells, the motion of node A passes through other grid cells of differing densities.

Second, the narrow wedge angle provides density-based clustering of space around each node. The wedges are small enough so that they provide a good degree of homogeneity within each. Consequently, the density metric defined for each wedge can be assigned to all the points inside the wedge.

Third both the shape of the wedge and the narrow angle allows the waypoint choices to be nearly independent of the orientation of the coordinate axes. In general, coordinate axis orientation is arbitrary and should not affect node behavior. The choice of waypoints should not depend on, for example, the direction of North or East, and nodes should exhibit similar behavior regardless of the orientation of these axes.

### 3.4 Density Metric and Outer Wedge Boundary

The wedge defined in the previous section is a closed shape, enabling us to define the area over which to pick a random waypoint. It also affects the calculation of the density metric for each wedge. The outer boundary is defined as a circular arc, to remove the dependency of the wedge shape on the orientation of the coordinate axes. The radius of this circle is defined to cover the farthest point that would still be within the wireless radio range of at least one other mobile node. For instance, in the example of Figure 3.5, the farthest point from node A inside the wedge is the intersection point of the wedge and radio range circle around node D, which is farther from A.
The selection of wedges is based upon the density metric associated with each wedge. This metric indicates how occupied each wedge is, which is used as the measure of sparsity/density. Intuitively, this metric represents the number of links a node have if the node were to be placed at a random location inside the wedge. As such we define the \textit{density metric} $\hat{D}$ in terms of the area of a wedge and its overlaps with other radio range circles as: $\hat{D}_{W_i} = \frac{\sum_{j \neq k} A(W_i \cap C_j)}{A(W_i)}$ where $W_i$ is the wedge, $C_j$ is the radio range circle around node $j$, $\cap$ denotes the geometric intersection, $A(\ldots)$ denotes the area of each shape, and $k$ is is the index of the node being considered. Essentially, the contributions of the coverage percentage of each one of the nodes to this wedge are summed up. Each contribution is calculated by finding the intersection area of each wedge with the radio range circle around each node. Section 4.3 describes a robust algorithm that finds the area of intersection efficiently. The above density metric effectively approximates the expected number of neighbors a node have at a random point within the wedge. By defining it so, not only we consider the contributions
of nodes that lie inside the wedge, we also take into account nodes that are outside
the region that have potential of becoming neighbors of this node. For example, in
Figure 3.5, the intersection of the radio range circles of nodes $B, C, D,$ and $E$ with
the wedge around node $A$ is shown. Even a node far away from the wedge like $E$ can
have a small contribution to the total overlap.

### 3.5 Selecting a Wedge

After identifying and computing each wedge density metric, a wedge is selected ac-
cording to the input density control parameter, chosen from $[0, 1]$, indicating the
degree of desired sparsity/density.

#### 3.5.1 Rationale for the Wedge Selection Algorithm

If in selecting wedges, nodes were to always pick the densest (or the sparsest) choice,
 extreme scenarios (very dense/sparse) are generated. In the sparse case, however,
 nodes stretch out in the simulation space so much that the topology effectively be-
comes rigid and no more motion is possible. In this case, nodes lose their degree of
 freedom to move since the topology needs to stay connected. On the other hand, if
 nodes constantly select the densest wedge to pick a waypoint in, the topology com-
presses, as most of the motion happens towards the middle of the network. In this
 case, the topology compacts to the point that almost all the nodes are within the
 wireless range of one another. Thus, we need to pick a middle ground that the selec-
tion of both sparse and dense wedges are possible in one scenario, according to the
density control parameter. In doing so, we can view the density control parameter as
 a sliding scale, whose value controls the average dense/sparse waypoint choices by the
 nodes. The value of this parameter is used to generate the probability distribution of
the wedge selection. (Although for simplicity we used a single parameter to generate this distribution, the user can also explicitly set the individual probability values of the wedges if desired.) The value of this parameter controls the overall density/sparsity of the scenarios generated, while still allowing some dense (sparse) choices in an overall sparse (dense) scenario. In the next section, I explain the probability distribution of the wedge selection according to this parameter.

3.5.2 Probability Distribution of Wedge Selection

For simplicity of the user interface and easier understanding of the generated scenarios’ behavior, two parameters control for the probability density: (i) a single density control parameter used to adjust the degree each node exhibits sparse/dense behavior and (ii) variance control parameter that controls the amount of variability in the selection of wedges. The model also allows the individual selection of the wedge probabilities, if the user wishes. As discussed, in sparse scenarios, each individual node should be capable of making dense choices from time to time, and vice versa. Figure 3.6 illustrates an example probability density function of wedge selection for 3 example density control parameter values, and a variance parameter value of 5. The x-axis corresponds to the wedge number in increasing order of wedge densities from left to right, and the y-axis shows their probability value. For ease of explanation, we have shown the probability values for 6 wedges rather than 12 wedges to make the graph less cluttered. As will be explained below, the probability distribution function is a continuous function which is mapped to 12 discrete individual wedges. Above the probability density functions, the value of the probability for the corresponding wedge is shown; these values correspond to the 3 parameter choices of their associated color.
For example, the red solid curve represents an example of a sparse parameter setting, where the value of the density control parameter is 0.1. As is shown, there is a high probability of choosing the sparest wedge (85.45%), and the probability starts dropping as the wedges become denser; however, the probability starts growing back towards the densest wedges, and it comes to the value of 9.49% at the densest wedge. The situation for a density control parameter if 0.9 is the opposite of above (the black dash-dot curve), which corresponds to a dense parameter setting. In this scenario, most of the probability mass is towards the densest wedges, with some small amount towards the sparse end. 0.5 (the blue dashed curve) corresponds to a network where the probability of wedge selection for both the densest and sparsest wedges is equal (47.47% for both the sparsest and densest wedge).
To generate such probability distribution, a mixture of 2 truncated normal distributions is used, one corresponding to the sparse, and the other one to the dense selection. The reason for using normal distribution is that the probability of mean exceeds other values, and the density function smoothly decreases. This translates to the smooth reduction in probability of selecting wedges in both the sparse, and the dense end of the spectrum. However, the range of values needs to be bounded since our range of interest is \([0, 1]\), hence truncated normal distribution is used for each end. The truncated normal probability density function is denoted by \(\psi(\mu, \sigma, a, b; x)\), where \(\mu\) and \(\sigma\) are the mean and variance of the generating normal PDF, and \(a\) and \(b\) specify the truncated interval. The PDF of the wedge selection is then derived by:

\[
\omega(\sigma, a, b; x) = (b - a) \cdot ((1 - d) \cdot \psi(a, \sigma, a, b; a + (b - a)x) + d \cdot \psi(b, \sigma, a, b; a + (b - a)x)) \quad 0 \leq x \leq 1
\]

(3.1)
in which, the PDF of a truncated normal distribution is obtained from:

\[
\psi(\mu, \sigma, a, b; x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x - \mu)^2}{2\sigma^2}} \Phi\left(\frac{b - \mu}{\sigma}\right) - \Phi\left(\frac{a - \mu}{\sigma}\right) \quad a \leq x \leq b
\]

where \(\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\) is the PDF of the standard normal distribution, and \(\Phi(.)\) is its cumulative distribution function. In Formula (3.1), \(d\) and \(\sigma\) are the density control and variance control parameters, respectively. Also, as (3.1) suggests, the PDF of wedge selection is a mixture of two probabilities, weighted by the value of the density control parameter. The density control parameters controls the likelihood of choosing either of the sparse or dense component distributions. Each one of these components is formed by a truncated normal distribution, one corresponding to the sparse wedge selection with mean value being the left endpoint \((a)\), and the other
one corresponding to the dense wedge selection with mean value equal to the right endpoint \( b \) of the interval. Essentially, wedge selection random variable ranges over the interval \([0, 1]\), which is scaled to the truncated interval of \([a, b]\). The scaling factor of \((b - a)\) guarantees that the PDF integrates to 1. To define the truncated interval \([a, b]\), it’s highly desirable to have the sparse (dense) distribution component to have the majority of its contribution at the sparse (dense) end, and have close to zero contribution towards the dense (sparse) end of spectrum. To this end, the truncation interval should extend at least five standards deviations to each corresponding side of the mean value. In the implementation, I picked \( \sigma = 5, \ a = 50, \ b = 150 \) that meets these criteria.

### 3.6 Additional Model Features

In this section, I discuss some of the additional features required to make the model work in practice, as well as allowing the generation of richer mobility scenarios.

#### 3.6.1 Dealing with Node Vibration

As described, if any node’s further motion toward its current waypoint would partition the network, the node stops and picks another waypoint. However, if a node acts as the sole bridge between clusters of nodes, and if the intersection of the radio ranges around these clusters is very small, the node essentially can make only very tiny movements in any direction, in the extreme case just “vibrating” in place. Node vibration is defined such that the movements made during vibration are much too small to cause network topology changes, but such movements still are all generated (according to the wedge densities and the density control parameter, as described in Section 3.5), causing the scenario generation and the later network protocol simulation
Figure 3.7  Node C currently acts as a bridge between nodes A and B and can only “vibrate” in the tiny intersection region, shown as shaded, in order to avoid partitions on that scenario to both run extremely slowly, and the resulting scenario file and network protocol simulation trace file to both be extremely large.

Figure 3.7 illustrates a simple example of such node vibration. Node C here, acts as the sole bridge between A and B, and intersection of the coverage areas around B and C is very small. To keep either of these nodes from disconnecting, C can only move within the small shaded intersection area of the coverage circles.

Directly detecting the extent of the constraints on a node’s ability to move is difficult, so I instead indirectly detect that a node is vibrating by monitoring its past movements. By looking at the node’s most recent movement history, and finding out whether the node has made at least some large number of movements (e.g., 100 in our implementation) over a very small period of time (e.g., 1 ms in our implementation), I recognize the node is effectively not really going anywhere, despite all its simulated movement events. It is worth noting that such movement does not impact the mobility model behavior, nor the behavior of the radio propagation. For a movement to have
impact on the radio behavior, it has to be at least within a quarter to a half of a wavelength. In Wi-Fi frequency of 2.4 GHz which is used in radio signals, the wavelength is around 12.5 cm. Therefore, movement over very short distances of less than 3 cm does not make impact on the radio propagation. In this case, the node is then stopped from moving for a small time (e.g., the node remains stationary for 1 s in our implementation) before attempting to continue. Other nodes that have not been detected to be vibrating continue to move normally throughout. This pause time has the following two important properties:

- It’s small enough to possibly allow some but not skip over too many changes in the constraints that limits the node’s mobility.
- It’s large enough to allow significant forward progress in the simulation time.

This mechanism is both simple and effective, ensuring the reasonable forward progress of time within the generation and network simulation, without real impact on the model behavior or wireless protocol performance.

### 3.6.2 Encouraging Additional Topology Changes

In very sparse scenarios, the motion of nodes may become so constrained that change in network topology becomes very difficult. We introduce optional *leaf mode* behavior to encourage topology change even in such cases. Nodes with only a single neighbor have the highest degree of movement freedom since their only constraint is being within the radio range of their only neighbor. After a large cumulative time of being a leaf, denoted by *max-leaf-duration*, a node may enter leaf mode. The node then pairs up with a randomly chosen other node (leaf or non-leaf), and these two nodes then try to move towards each other. Paired nodes stay in this mode until either of the following conditions is met:
Figure 3.8  A spends a long cumulative time of being leaf as a result of network becoming static; it gets into leaf-mode by pairing up with a random node D. Two nodes then start moving towards each other

- the two paired nodes reach within the wireless radio range of each other (become neighbors); or
- after a maximum time of max-leaf-period in leaf mode, after which the two nodes exit leaf mode and resume normal motion.

The value of both max-leaf-duration and max-leaf-period is chosen to be 100 seconds in the implementation.

For example, in Figure 3.8, a situation is depicted where the scenario has been static for a long period of time, and the max-leaf-duration threshold is reached for leaf node A. A picks a randomly chosen node (B), and pairs up with this node. The two nodes then start traveling towards each other, until they successfully reach within the wireless coverage range of each other.
3.7 Chapter Summary

This chapter first discussed the motivation behind designing RCS, and then delved into the different design aspects of the model. Among the details of the design, it described the concept of partitioning the space into wedges, and different constraints the model manages between the nodes motions. Then, the probability distribution function of wedge selection was presented. I then elaborated the vibration phenomenon which can happen as a result of simulating the motion of bridge nodes, and how it is crucial to handle this properly. Finally, the additional feature of the model was presented that encourages dynamic topology change, when nodes become overly constrained and the topology becomes nearly rigid.
Chapter 4

RCS Implementation Challenges

This chapter discusses some of the challenges in implementing the Random Controlled Sparse mobility model (RCS) based on the RCS design presented in Chapter 3.

4.1 General RCS Implementation Structure

The RCS model uses a discrete-event simulation model, where the network graph changes at discrete events, when a new link is formed or breaks. The network state is examined at each individual event to determine the connectivity of the graph, along with the distance changes among nodes in the network (in the extended version, as explained later). Upon detection of disconnection, the network stops at its current connected state, and the accountable node for disconnection is found. This is one of the endpoint nodes of the latest broken link, in which the node with the highest speed is selected. This node would then go through the process of selecting a wedge according to the density control parameter, and picking a waypoint within the wedge, until it can either successfully make a trip without disconnecting the network, or pauses if such movement turns out to be impossible at the moment (vibration case).

The simulation of mobile nodes is implemented as a discrete-event simulation process. This model is particularly suitable for our purpose because of the following reasons:
• Discrete nature of the model. The network graph goes through changes as different links are added or break. The state of network graph remains the same in between these changes.

• Event-based trip rules. In RCS, nodes pick their next waypoint based on the current density state of the network graph; these states must be examined at each graph changing or node arrival events. Due to the flexibility nature of event-based paradigms, it’s easier to implement our intended motion behavior in this paradigm.

4.2 Scheduler and Description of Events

At each time instant in the simulation, a node’s state is expressed in the form

\[ \{\text{TIME}, \text{position}, \text{speed}, \text{arrival-time}\} \]

where \text{TIME} is the last update time of the current node’s position, \text{position} the latest coordinates of the node in the space, \text{speed} the movement speed of node towards its next waypoint, and \text{arrival-time} being the calculated time when the node reaches its next waypoint. An event is observed if either of the following conditions happens:

1. Node reaches its next waypoint which is equal to its \text{arrival-time}.

2. A link is formed between two nodes; this event happens when two mobile nodes come within radio transmission range of each other.

3. A link breaks between two nodes; this events occurs when two mobile nodes go outside each others’ transmission range.

The scheduler where events are stored is implemented as a \textit{hashed priority queue} data structure, where extraction of events occur according to increasing order of the
timestamp. This implementation is able to provide very efficient event operations; extraction, removal, and update all done in $O(\log n)$, as well as $O(1)$ event access.

The events $E$ in the scheduler are stored as a pair of node IDs, and two timestamps. The node IDs are equal for an **arrival-time** event type, and they represent the node IDs involved in a **link-creation** or **link-removal** event. The first timestamp $E.T_1$ represents the actual time where the event is to be executed. The second timestamp is only meaningful for a **link-creation** or **link-removal** event. This corresponds to the second time where the pair cross each others’ radio range, or a negative value if they cross the range only once. For an arrival event

Upon execution of each event from the head of the scheduler, the following two actions occur:

1. The state variables corresponding to the network graph is updated (graph partitioning check is performed here).

2. Execution of the event may lead to creation of new events or removal of some existing events from the scheduler.

The processing of an **arrival-time** event type, generates another **arrival-time** event. This is the arrival time of the node at its chosen waypoint (calculated according to its chosen uniform random speed in interval $[v_{min}, v_{max}]$), or the wakeup time of the node, if it ends up sleeping because of vibration.

Processing an **arrival-time** event type, also updates the **link-creation** and **link-removal** events involving this node. That is, all the events’ timestamps involving a pair of nodes consisting of this node and any other network is updated in the scheduler. The calculation of these time stamps is done by finding the time when the pair of nodes come within radio range of each other. This turns into finding the roots
of a second degree polynomial equation, namely \( t_1 \) and \( t_2 \). Depending on the sign of each root, the type of each event is determined, as described below (here \( \text{TIME} \) refers to the global simulation time):

- \( t_1 < 0 \) and \( t_2 < 0 \). There are no future time solutions to the polynomial equation.

- \( t_1 > 0 \) and \( t_2 > 0 \). The pair of nodes cross the radio range of each other twice. The minimum value \( \text{TIME} + \min(t_1, t_2) \) is considered as the next execution time of this event (\( E.T_1 \)). \( \text{TIME} + \max(t_1, t_2) \) is copied into the event’s second timestamp field (\( E.T_2 \)) to remember that this event is a link-creation event when processed.

- \( (t_1 > 0 \text{ and } t_2 < 0) \) or \( (t_1 < 0 \text{ and } t_2 > 0) \). The pair of nodes cross the transmission range of each other exactly once, which happens at current simulation time plus the positive root: \( \text{TIME} + \max(t_1, t_2) \). A negative timestamp is copied as the second timestamp field, to remember that this event is a link-removal event when processed.

- \( (t_1 = 0 \text{ and } t_2 > 0) \) or \( (t_1 > 0 \text{ and } t_2 = 0) \). This corresponds to the calculation of a pair that are currently within radio range of each other, hence one of the roots being zero. The positive sign of the other root indicates these nodes cross the radio range once more; therefore, the state of the link between the pair should be updated to reachable (link-creation event).

- \( (t_1 = 0 \text{ and } t_2 < 0) \) or \( (t_1 < 0 \text{ and } t_2 = 0) \). Similar to the above, this case corresponds to the calculation of a pair that are currently within radio range of each other. The negative sign of the other root indicates these nodes had
been within the transmission range in the past; therefore, the state of the link
between the pair should be updated to unreachable (link-removal event).

Execution of a link-removal may trigger no event or additional future events,
depending on the state of the network graph. In case of the network graph disconnec-
tion, new arrival-time is generated, which essentially replaces the previous motion
of the node. A graph disconnection is detected by examining the connectivity of the
network graph at the instant when the event is executed. In the standard version, this
check is done by using a DFS traversal of the network graph, and checking whether all
the nodes are visited (in which case the graph is connected) or not (network partition
is detected). Floyd-Warshall algorithm is used in the extended version to determine
the partitioning as well as gathering the shortest path distances among pairs of nodes.

To improve the running time of the partition detection in both scenario generator
versions, an LRU-based cache of the most recent network graph states is introduced.
This cache serves to accelerate the process of the partitioning check, and gathering
the distance information, for the already visited graph structures. This cache is
responsible for maintaining the most recently visited graph structures, and discarding
the old graph data structures. This enables a big advantage over performing the
partition-check algorithm from scratch at every single event, in case the graph can be
recovered from the existing entries in the cache.

In the event of a network disconnection, the network graph state is reverted back
to the instant right before the new event was executed. The current simulation
time (TIME), however, remains the same since the network graph was still connected
right before the new event takes effect. At this stage, the responsible node for the
disconnection is found. In the implementation, the responsible node is taken as the
one with the highest speed among the two endpoints of the broken link.
The responsible node must be in either of the following states:

1. It is connected to the entire graph by a single link, and that link is broken by further movement of the node in its current direction.

2. It acts as the sole bridge between clusters of nodes, and a link loss results in splitting the network graph into those clusters.

The fastest moving node at the end of the broken link is chosen, as such node would contribute faster to the graph disconnection. The disconnecting node then triggers an arrival-time event, and this process continues until either the simulation time moves forward, or a maximum number of events is generated crossing the vibration-threshold number of events in a short period of time (100 events in 1 millisecond in the implementation). In this case, the node pauses for a short time pause-time (1 second in the implementation), before resuming its normal operation.

Choosing waypoints by nodes is according to the density metric of the wedges around each node. This density metric requires calculation of the intersection between a wedge and the radio range circles around each node. In the following section, I discuss the design and implementation of a robust algorithm that computes this intersection efficiently.

### 4.3 Intersection Algorithm

The intersection between a wedge and the radio range circle around each node is used as part of finding the density metric corresponding to each wedge (Section 3.4). The intersection computation is done extensively through the running of the model, and thus a robust and efficient intersection algorithm is required. No general algorithm exists that solves the intersection among arbitrary shapes (such as a sector and circle
of arbitrary sizes). Current geometric algorithm generally require identifying all the possible input cases. Such enumeration of inputs can easily exceed 11 cases without even considering the degenerate configurations. The algorithm presented here, however, is able to handle all the possible inputs, including all the degenerate cases, efficiently and in a general manner.

Here are some of the terminologies I will use throughout this section. Let $P_{ij} = P_j - P_i$, and a line segment $seg_i$ from $P_i$ to $P_j$ be $seg_i = \overline{P_iP_j}$. Furthermore, let $C_i = (c_i, r_i)$ be circle indexed $i$ with center $c_i$ and radius $r_i$. A sector or arc having a supporting circle $C_i$ which starts at angle $start$ and ends at $end$ is represented as $S_i = (C_i, start, end, dir)$, where $dir$ is the movement direction which is either clock (CW), or counter-clockwise (CCW). To find the intersection area between circle $C_i = (c_i, r_i)$ and sector $S_j = (C_j = (c_j, r_j), start, end, dir)$, it is first noted that the intersection is a convex shape. The algorithm finds the vertices of the intersection region, from which a convex polygon is constructed (unless there are less than three corner points). The shape of the intersection region is identified by this convex polygon, where some of the polygon sides are arcs belonging to either the circle or sector. Figure 4.1 shows a general form of this intersection region in the shaded area. The polygon $p_1 - p_2 - p_3 - p_4$ is the inscribed convex polygon in the intersection region that our algorithm tries to find.

To find the vertices of the intersection region, we first divide the sector into 3 components, two line segments and one circular arc:

(i) $arc = (c_j, start, end, dir)$.

(ii) $seg_1 = c_i, startPoint$.

(iii) $seg_2 = c_i, endPoint$. 
where \textit{startPoint} and \textit{endPoint} are the points corresponding to the beginning and end of the arc.

We first determine whether either of three corner points of the sector (center, startPoint, endPoint) belong to the intersection region. This check is made by testing whether the points lies within the circle $C_i$. The next step consists of finding the intersection points between the two shapes. The intersection includes finding the following set of intersections:

(i) intersection between $seg_1$ and $C_i$.

(ii) intersection between $seg_2$ and $C_i$.

(iii) intersection between $arc$ and $C_i$. 

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure4.1.png}
\caption{A general intersection region shape between an arbitrary circle and sector}
\end{figure}
4.3.1 Intersection of a Line Segment and Circle

To find the intersection points between segment $l = \overline{p_1p_2}$ and circle $C = (c_1, r)$, we first find the projection of the center of circle onto $l$:

$$\text{proj}_{c_1} = p_1 + \frac{(c_1 - p_1) \cdot (p_2 - p_1)}{(p_2 - p_1) \cdot (p_2 - p_1)} \times (p_2 - p_1)$$

Then we calculate the distance from the segment to $C$:

$$\text{dist}(\text{proj}_{c_1}, C) = \sqrt{(\text{proj}_{c_1} - c_1) \cdot (\text{proj}_{c_1} - c_1)}$$

If this distance is greater than $r$, there is no intersection; otherwise we compute the length of the chord intercepted by the segment cutting the circle:

$$\text{length( chord)} = \frac{h}{2} = \sqrt{r^2 - \text{dist}^2(\text{proj}_{c_1}, C)}$$

The two intersection points (or one in case the segment is tangent to the circle) lie equi-distance at length $h/2$ on either sides of the line segment:

$$\text{inter}_1 = \text{proj}_{c_1} - \frac{h}{\text{length}(p_2 - p_1)}(p_2 - p_1)$$

$$\text{inter}_2 = \text{proj}_{c_1} + \frac{h}{\text{length}(p_2 - p_1)}(p_2 - p_1)$$

Next, we check whether either of these two candidate points lie on the line segment. If they belong to the line segment, they are returned as the intersection points. The following procedure describes how we can determine whether a point belongs to a line segment.
4.3.2 Checking the Inclusion of a Point on a Segment

For a point $q$ and line segment $seg = \overline{p_1p_2}$, we should first determine whether $(p_2 - p_1)$ and $(q - p_1)$ are collinear. This is true if $(p_2 - p_1) \times (q - p_1) = 0$. If this test is passed, we then check if the point $q$ lies between the two segment end points:

$$0 \leq (p_2 - p_1) \cdot (q - p_1) \leq (p_2 - p_1) \cdot (p_2 - p_1)$$

If this test is passed too, the point belongs to the line segment.

Now we need to find the intersection between the sector’s arc and the circle.

4.3.3 Intersection of a Circular Arc and a Circle

To find the intersection of circular arc $S = (C_j = (c_j, r_j), start, end, dir)$ with circle $C_i = (c_i, r_i)$, we first treat the circular arc as a full circle, and find the intersection of these two circles; then we determine which one of the intersection points lie on the arc. Let $d$ be the distance between the two centers: $d = length(c_j - c_i)$. The following condition indicates there is at least one solution:

$$|r_j - r_i| \leq d \leq |r_i + r_j|$$

We also check whether $d = 0$ which indicates overlapping arcs, where we return $NULL$ as well (the two endpoints of the arc would serve as the vertices of the intersection region, which we considered earlier). Let the intersection chord be the line segment connecting the two intersection points (or zero length chord if there is only one intersection), and $h$ be half the length of this chord. We measure the
distance between the arc center and this chord as follows:

\[ r_j^2 = a^2 + h^2, \quad r_i^2 = (d - a)^2 + h^2 = d^2 + a^2 - 2da + h^2 \]

Subtracting the two equations gives:

\[ r_j^2 - r_i^2 = 2da - d^2 \Rightarrow a = \frac{r_j^2 - r_i^2 + d^2}{2d} \]

Let \( p \) be the center of the intersection chord:

\[ p = c_j + \frac{a}{d}(c_i - c_j) \]

We first note that if \( h = 0 \) the two circles are tangent to each other, intersecting at one point. Otherwise the two intersect at the following points:

\[ \text{inter}_1 = p + \frac{h}{d}(c_i - c_j) \]
\[ \text{inter}_2 = p - \frac{h}{d}(c_i - c_j) \]

The next phase is checking whether either of these points belong to the circular arc, described below.

### 4.3.4 Checking the Inclusion of a Point on a Circular Arc

To determine if a point \( q \) lies on arc \( S = (C = (c_i, r), \text{start}, \text{end}, \text{dir}) \), we should first check whether the point lies on circle \( C \). This check is done by testing the distance of point \( q \) from center \( c_i \) which should equal the radius \( r \). The second check is finding out whether the point lies on the arc. Let \( \text{startPoint} \) and \( \text{endPoint} \) be the start
and end points of $S$. If the direction of travel is counterclockwise, for $q$ to be part of $S$, it should be located to the right of line segment $\overline{seg = startPoint endPoint}$. If the direction is clockwise, however, $q$ needs to be to the left of $seg$ when traveling from $startPoint$ to $endPoint$. We can perform this check by using cross product as follows:

\[
\begin{align*}
\text{if } (S.\text{dir} == \text{CCW}) & \\
\text{return } \text{length}(q - c_i) = r \land \\
& (\text{endPoint} - \text{startPoint}) \times (q - \text{startPoint}) \leq 0 \\
\text{else} & \\
\text{return } \text{length}(q - c_i) = r \land \\
& (\text{endPoint} - \text{startPoint}) \times (q - \text{startPoint}) \geq 0
\end{align*}
\]

After finding the vertices of the intersection region, we look at the number of vertices obtained. If it’s either 0 or 1, the intersection either contains the full shape (which is either the circle or sector), or it’s empty. The determination is based upon whether the center of either shape lies within the interior of the other shape or not:

\[
\begin{align*}
\text{if } ( \text{is\_inside\_sector}(S, c_i) ) & \\
\text{return } \pi \times r_i^2 \\
\text{elseif } ( \text{is\_inside\_circle}(C, c_i) ) & \\
\text{return } 0.5 \times r_j^2 \times \theta \\
\text{else} & \\
\text{return } 0
\end{align*}
\]

where $\theta$ is the angle spanned by the sector. We make a general procedure where we can determine whether a point lies inside a sector. We note that the circle case is a special case of sector with start and end angles being 0 and $2\pi$, respectively.
4.3.5 Checking whether a Point Lies within a Sector

The first check that needs to be done is whether the distance of point $q$ from center $c_i$ is within radius $r$. Then we should verify that the angle of the point is in between start and end angles of the sector. However, the second test is not as simple since two angles could differ by multiples of $2\pi$ and still are equivalent. Since we are designing a general purpose function, the input’s sector angles could not be limited to a specific range. To address this, we convert all the angles to a value in $[0, 2\pi]$. Let $q\_angle$ be the angle of $q$ with respect to the positive $x$ axis. We calculate the following 3 quantities in which $diff\_angle$ is a function that returns the difference between the first and second angle arguments in the interval $[0, 2\pi]$, if we travel in the direction of $dir$.

$$
\begin{align*}
\text{diff1} &= \text{diff}\_\text{angle}(S\_.\text{start}, q\_angle, S\_.\text{dir}) \\
\text{diff2} &= \text{diff}\_\text{angle}(q\_angle, S\_.\text{end}, S\_.\text{dir}) \\
\text{total\_diff} &= \text{diff}\_\text{angle}(S\_.\text{start}, S\_.\text{end}, S\_.\text{dir})
\end{align*}
$$

For $q$ to be within the sector’s angular range, the sum of $\text{diff1}$ and $\text{diff2}$ should equal $\text{total\_diff}$. Thus we return the following

$$
\text{return } length(q - c_i) \leq r \ \&\& \ \text{diff1 + diff2} == \text{total\_diff}
$$

After taking care of the case with zero or 1 intersection vertices, we are left with two or more corner points. We sort these points counterclockwise to find the sides of the convex polygon. The sorting of points is done based on the angular position of points with respect to the center, in which center point for a set of points $p_1, \ldots, p_n$
is:

$$center = \frac{p_1 + \cdots + p_n}{n}$$

To construct the *less-than* operation for these points, we first divide the points into two half planes; the ones above and below (higher and lower $y$ values than) the center point. The higher half plane points precede the lower one in the ordering of the points:

```plaintext
if ( \text{sign}(p_i.y - center.y) \neq \text{sign}(p_j.y - center.y) )
return p_i.y > p_j.y
```

For points on the same half plane, we use the cross product of two vectors corresponding to the points. If the result is positive, the first point precedes the second one (it makes a smaller angle with the positive half of $x$-axis):

```plaintext
return (p_i - center) \times (p_j - center)
```

This will order the points counter-clockwise starting from the positive half of $x$-axis.

After finding out the polygon vertices, we calculate the area of this convex polygon. Area of a convex polygon consisting of vertices $p_1, \ldots, p_n$ ordered counterclockwise is given by:

$$\frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) = \frac{1}{2} \sum_{i=0}^{n-1} p_i \times p_{i+1}$$

where $p_0 = p_n$. After computing the polygon’s area, we loop over all the polygon sides and for each side, we find out which polygon side should be extended to a circular arc as part of the intersection region. There are two scenarios where the boundary of the intersection between two consecutive polygon vertices in an arc:

1. The polygon side does not sit on any of the line segments belonging to the sector.
Figure 4.2 A general intersection region shape between an arbitrary circle and sector

2. The polygon side belongs to a sector’s line segment; however, the intersection region lies to the right of the polygon side.

The boundary of the intersection region is a subset of the boundary of two shapes. Consequently, if such a boundary is not on either of the line segments (case 1), it should belong to a circular arc. The second case arises in the following situation: The intersection region of the sector and circle in Figure 4.2 is defined by the shaded area. When traversing the vertices counterclockwise, the region always lies to the left of the boundary curve \((\overline{p_1p_3}, \overline{p_3p_2p_1})\). In the \(\overline{p_1p_3}\) traversal, the region lies to the left, therefore the line segment \(\overline{p_1p_3}\) is part of the region boundary, whereas if we had considered \(\overline{p_3p_1}\) as the boundary segment, the region lies to its right. Therefore \(\overline{p_3p_1}\) cannot be part of the boundary, and the circular arc \(\overline{p_3p_2p_1}\) forms the boundary curve.
To find which circular arc is the right boundary curve, we turn our attention to the possible positioning of the center of the circle with respect to the polygon side. In Figure 4.3, we have shown 4 different circle sharing the same chord $AB$. The circular boundary arc is to the right of $p_1p_2$. As the center of the circle moves toward the left of the chord (from $c_4$ to $c_1$), the circular segment formed becomes smaller, each one lying completely within the previous circular segment. Since the intersection region contains the smallest circular segment resting on the same chord, we should find the circle with the largest center’s distance from the chord. The distance is defined to have a negative sign to the right of the chord and positive sign to the left. For instance between $c_1$ and $c_2$, $c_1$ has the largest distance; however, among $c_3$ and $c_4$, $c_3$ has the largest distance since both of their distances have a negative sign.

The case of $c_1$ and $c_2$ where the center point lies to the left of the polygon side indicates a minor arc (less than 180 degrees), whereas the case of $c_3$ and $c_4$ where the
center sits to the right of the chord represents a *major* arc (an arc having a measure greater than or equal to 180 degrees) as the boundary curve.

The next task is to add the area of the circular segment to the total area if the arc is part of the boundary curve. Let $\theta$ be the central angle in radians of a sector which is the angle between the two segment legs of the circular segment. This angle can be computed by $\theta = 2 \arcsin \left( \frac{\text{length}(p_{ij})}{2 \text{radius}} \right)$. For a minor arc, we define $\theta' = \theta$, and for a major one, $\theta'$ is defined as $\theta' = 2\pi - \theta$. Then the area of the circular segment is given by:

$$\text{circular segment’s area: } \frac{1}{2}r^2(\theta' - \sin \theta')$$

The polygon’s sides are visited and processed one by one and the area of each circular segment is added to the total area when an arc is part of the boundary curve of the intersection region. The final area that we are left with would be the exact area of the intersection region between the circle and the circular sector.

The above computational geometry algorithm requires many floating-point arithmetics on floating-point numbers. Floating-point computations are inherently inexact, and can introduce significant errors. These errors can lead to erroneous decisions in the algorithm, and thus run-time failures. Small errors can make many geometric algorithm fails and their implementations to crash, due to floating-point inaccuracies such as rounding errors. In the next section, I discuss how different arithmetic operations and algorithms in the implementation are designed to produce very little errors, leading to a robust intersection algorithm.
4.4 Floating Point Considerations

Floating point computations are not precise in computers due to the limited number of bits available bits for representing them. Relative and absolute errors can occur because of rounding errors in the calculations. Since we are dealing with the floating point numbers heavily, we need to take these factors into account. In particular, we are faced with following two challenges: sophistication in comparing two float numbers, and loss of significance in computations. They are described in the following sections.

4.4.1 Comparison Between Two Floating point Numbers

It’s very unlikely that comparing two different float numbers results in exact match. The error would depend on the order of the calculations and the double precision on the machine. Machine Epsilon is defined to be the upper bound on the relative error in a CPU architecture, which we will represent it by $\text{ROUND\_ERROR}$. Instead of the regular exact comparison operations, we design custom built comparison functions in order to take the floating errors into account. The following is how we define the equality, less than or equal, and greater than or equal operations.

```plaintext
almost_eq(a, b)
    if ( |a - b| < ROUND_ERROR )
        return true;
    relative_error = max(|a|, |b|) × ROUND_ERROR;
    return |a - b| ≤ relative_error;

almost_leq(a, b)
    if ( |a - b| < ROUND_ERROR )
```
\begin{verbatim}
return a ≤ b + ROUND_ERROR;
relative_error = max(|a|, |b|) × ROUND_ERROR;
return a ≤ b + relative_error;
\end{verbatim}

\begin{verbatim}
almost_geq(a, b)
if ( |a − b| < ROUND_ERROR )
    return a ≥ b − ROUND_ERROR;
relative_error = max(|a|, |b|) × ROUND_ERROR;
return a ≥ b − relative_error;
\end{verbatim}

The first test in the above functions takes care of numbers that are both very close to zero. In these cases, the relative_error measure alone can produce a large error, whereas the actual difference between the numbers could be within the acceptable range of ROUND_ERROR; hence, the numbers should be considered equal.

4.4.2 Loss of Significance

This condition is related to the arithmetics themselves, and how some operations can increase the relative error significantly compared to the absolute error. One major case of this problem occurs when two large nearly equal numbers are subtracted from each other. Below we describe two instances of this phenomenon in our simulation, and the solution we chose accordingly.

Calculating the roots of a second degree polynomial

Section 4.2 discussed that the problem of computing the event’s time for link-creation or link-removal events turns into solving the roots of a quadratic equation. It is shown that the typical root calculation technique for equation \( f(x) = ax^2 + bx + c \) using
the quadratic formula:

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \]

(4.1)

can produce inaccurate results when \( a \) and \( c \) are very small compared to \( b \). One of the roots would be either zero or a value very close to 0 with very few meaningful significant digits left. This unstable root calculation translates to inaccurate results when one of the roots is very close to zero (case 4 and 5 of link change event’s timestamp calculation in Section 4.2). Since we are dealing with a lot of boundary cases in root calculations when nodes lie on radio boundary range of each other, we need to take extra care in the precision of the algorithms. The more stable algorithm that avoids the cancellation problem in the numerator for one of the roots is the following formula:

\[
x_1 = \frac{-b - \text{sgn}(b) \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{c}{ax_1}
\]

(4.2)

where \( \text{sgn} \) denotes the sign function. Effectively, this method would choose adding two positive or negative numbers together for the potentially problematic root, and the subtraction is removed.

**Comparing a number with zero that comes from internal subtraction**

Another instance of this problem happens when checking for segment inclusion (Section 4.3.2). We described how we should first verify \((p_2 - p_1) \times (q - p_1) = 0\). This test requires performing the following operation:

\[(p_2 - p_1)_x \times (q - p_1)_y - (p_2 - p_1)_y \times (q - p_1)_x = 0\]
If the two operands subtracted above are large, the result suffers loss of significant bits. Instead of performing the subtraction operation and testing the result against zero, we compute each operand separately; then compare the two operands for equality:

\[
\text{almost\_equals}((p_2 - p_1)_x \times (q - p_1)_y, (p_2 - p_1)_y \times (q - p_1)_x)
\]

This avoids the cancellation problem in the original calculation.

### 4.5 Chapter Summary

This chapter described the implementation details of the scenario generator. Events were defined, and their processing was discussed by the scheduler which is implemented as a hashed priority queue. The triggering of other events by currently processed events was explained, and how the state of the network graph changes, as a result of executing certain events. Vibration detection mechanism was described, and the solution was presented that ensures that simulation does not stall, and progresses forward in a timely manner. The underlying intersection algorithm used in the wedge selection was detailed. The impact of hardware floating-point arithmetic errors on the failure of geometric algorithms was discussed. I then discussed how the effect of these errors were minimized in the implementation by carefully designing arithmetics and algorithms that introduce errors limited by the machine epsilon.
Chapter 5

Evaluation of the Model and Implementation

In this chapter, the Random Controlled Sparse mobility model (RCS) is first evaluated in terms of different mobility metrics such as the average number of neighbors and average physical path length among different pairs of nodes. I then describe the effectiveness of the model in presenting the unexpected behavior of routing protocols by showing the performance result for DSDV, a well-studied traditional unicast routing protocol, across a wide range of sparse and dense network scenarios.

As discussed in Section 4.1, the model is implemented as a discrete-event simulator using highly efficient computational geometry and graph algorithms. The asymptotic complexity of the scenario generator’s running time is presented. Then, a parallel version of the model is discussed where certain expensive parts of the scenario generator is parallelized in order to improve the cost of the scenario generation. The running time graphs of the parallel version is presented for a wide range of number of nodes and density levels, and compared to that of the sequential version.

5.1 Evaluation of the Model in terms of Mobility Behavior and Revealing Protocol Behavior

RCS, as discussed earlier, can be used to produce scenarios of arbitrary density levels, according to the input density control parameter. In the following section, I will first visit the behavior of the model; this enables us to understand the characteristics of
topologies generated by the model, which later are used to evaluate the performance of routing protocols. I compared these characteristics with the behavior of Random Waypoint model to highlight the deficiencies in current models, and how limited they are in the scope of the resulting generated scenarios. Subsequently, I show the effectiveness of the model in revealing the behavior of routing protocols, where I show the performance of DSDV, a common unicast multihop wireless ad hoc routing protocol.

All simulations were done for 50 nodes for a duration of 1000 seconds. Using RCS, there is no system area that needs to be defined since there is no boundaries in the model. To have a fair comparison in terms of the behavior of the two models, two topology-related metrics are defined that are used for a model’s usefulness. I use these metrics to find the practical simulation area sizes for Random Waypoint model that can be used for protocol evaluation.

The first metric used is total partitioning time. This metrics computes the total time the network has been partitioned. The second metric is average pair partitioning time. This metric captures the amount of time, on average, any pair of nodes are disconnected from each other over the simulation time. This metric is measured by summing up all the times when any pair of nodes belong to different partitions. This sum is then divided by the number of pairs \((n^2 - n)\) to obtain the average time over a single pair. This metric essentially indicates how long a source and destination node are outside the communication range of each other (there is no multi-hop path between the two nodes). While the first metric looks at the network as a whole, the second one focuses on the partitioning from the standpoint of individual nodes.

Figure 5.1 shows these two metrics for Random Waypoint model. Since there’s no partitioning in RCS, it’s not plotted as the partitioning time would be equal to zero.
The $x$-axis indicates the simulation area as a square of size $x \times x \text{ m}^2$. Each point in the graphs represent the average of 10 independent runs of that scenario size, with no pause in between trips, and speed range of $[5, 20]$. The partitioning grows as the area size increases since nodes are spread across a larger area. For the two metrics, I find the first area size that the value of the partitioning time becomes noticeably above zero. For the total partitioning time metric, this occurs at size $800 \times 800 \text{ m}^2$ where it gets 11.85 s. This size of simulation is enough to make some significant partitioning occur. Nonetheless, the average pair partitioning time still remains relatively close to zero at this size. At this size, the majority of nodes remain within the same partition, with a few nodes intermittently disconnecting from the rest of the network every now and then. The largest we can go in simulation size, however, according to the average pair partitioning time metric, is $1100 \times 1100 \text{ m}^2$. Beyond this size, partitioning rapidly becomes a more serious problem, since there is a significant time duration when no connected path exists between a source and destination node.

For evaluation of the characteristics of the models, I use 4 different density control parameter values of 0.1, 0.5, 0.9, and 1, for RCS, and area sizes of $670 \text{ m} \times 670 \text{ m}$, $1100 \text{ m} \times 1100 \text{ m}$, $1500 \text{ m} \times 1500 \text{ m}$, and $2500 \text{ m} \times 2500 \text{ m}$, for the Random Waypoint model. These parameters are meant to generate the largest possible range of different density levels in each mobility model. The higher parameter for RCS leads to denser scenarios, whereas larger area size in Random Waypoint model, should result in sparser networks. I have not included the result for 0 density control parameter, however, since at this density level, nodes stretch out the farthest, and the topology becomes essentially rigid, with nodes exhibiting no interesting dynamic behavior.

In the evaluation of the routing protocol performance, however, I use two different parameter values for each model. These parameters represent the two ends of spec-
trum corresponding to very sparse and dense networks for each model to cover all ranges of behavior; in Random Waypoint model, the sparsest usable scenario size is $1100 \times 1100 \text{m}^2$, as discussed. For the densest choice, $670 \times 670 \text{m}^2$ area size is used, which is the size most typically utilized in the simulation of ad hoc networks; this size has a very low probability of disconnection and results in very dense networks.

5.1.1 Network Topology Results

I evaluated the behavior of the mobility models by measuring two metrics throughout the simulation runs, *average node degree* and *average path length*. These metrics are calculated by keeping track of all the *link-creation*, and *link removal* events in the scenarios, and updating the metrics as the network connectivity changes.
Figure 5.2 Average node degree vs. simulation time

Figure 5.2 illustrates the average number of nodes as a function of the simulation time. Along with this time-varying behavior of the models, each curve is also labeled with its overall average value. The average number of neighbors periodically increases and decreases over the simulation time, and both models can generate topologies over
a range of different average number of neighbors. However, reaching sparser topologies with Random Waypoint model comes at the expense of a substantial number of partitions. This situation is better observed in Figure 5.3 where the average path length is shown over time. RCS can generate scenarios over a range of average number
of 2 hops for a density control parameter of 1, up to an average of 10.67 for the sparse case (density of 0.1). It’s also important to note that while the average path length for the sparse scenario remains relatively high, such scenario maintains its dynamic structural change as witnessed by the fluctuations in the value of this metric over time. Random Waypoint model, however, can only generate scenarios with short paths (less than 4 hops). It’s also worth noting that as the simulation area increases, average path length increases and then decreases in Random Waypoint model, due to the larger number of partitions.

5.1.2 Example Routing Protocol Performance Evaluation

In this section, I discuss the range of behavior that an example routing protocol shows in different types of scenarios generated by RCS, and compare it with Random Waypoint model. Such behavior is shown to be drastically different in certain scenarios compared to what is generally observed in Random Waypoint model simulations. I demonstrate the effectiveness of the model by studying the behavior of an example routing protocol, namely DSDV, a traditional unicast multihop wireless routing protocol. Significant drop is observed in the performance of the protocol, in particular in sparse networks. For instance, in Figure 5.4a, the packet delivery ratio metric (PDR) is studied, which is the fraction of the packets successfully delivered to destination nodes. It’s observed that PDR drops severely as network becomes sparser. This is due to the longer routes and less redundant paths in sparse networks. Longer routes have a higher chance of breaking as a result of node mobility. Consequently, there’s a higher probability of packet drops, and the routing protocol is unable to maintain up-to-date routes in such networks. The performance drop is significant and it hits close to 10% PDR for a highly mobile network. Additionally, this figure shows the
limited capability of Random Waypoint model in showing such performance loss. The situation for packet delivery latency is depicted in Figure 5.4b. Longer path lengths in sparse networks translates to more hops to get to the destination, thus incurring larger delays. Total number of routing packets, as depicted in Figure 5.4c, stays at a lower level in sparse networks than dense ones. This is mainly due to the unreliability of the broadcast routing packets utilized in these protocols, which has a high chance of being dropped. This, in turn, translated to low PDR as was shown in Figure 5.4a, because the protocol is not able to find and maintain up-to-date routes to destination nodes in sparse networks.

As was discussed in the extended version of RCS, the scenario generator is able to maintain the shortest hop count paths among all pairs of nodes along the simulation time. These statistics are used to generate the graphs in Figure 5.4d. This $x$-axis shows the path length difference between the optimal route over which the packet was forwarded and received by the destination, and the optimal physical path length existed at the time the packet was generated by the sender. $y$-axis depicts the fraction of data packets successfully received over each one of these path length differences in DSDV. This figure shows the majority of packets generated by Random Waypoint model, and in dense networks generated by RCS, remain mostly within less than 7 hop count difference. However, the situation is quite different for sparse networks, and the path length difference greatly varies on both positive and negative end of the spectrum (from -18 all the way to 14 hop count difference). The reason for this large variation in delivered path length vs. shortest paths in sparse networks is that the packets are in flight for much longer time (longer hop count). Also, this figure illustrates that the path length difference used for route efficiency of routing protocol
cannot be a reliable metric definition in sparse networks, since such definition can easily generate negative path length differences in such networks,
Figure 5.4  Second set of performance graphs for DSDV; generated by RCS for different density levels, and RWP for different area sizes

5.2  Asymptotic Complexity of the Sequential Version

As was discussed in Chapter 4, two versions of the scenario generator are implemented. The standard version that implements the full functional simulator for generating
scenarios of different density levels, and the extended version which gathers more
statistics such as hop count distances among all network node pairs, to be used
in generating performance graphs for routing protocols. The running time of the
standard version is
\[ \mathcal{O}(|E| (n^2 + n \log n + n)) \]
where \(|E|\) and \(n\), are the total number mobility events and number of nodes, respectively. The mobility events as discussed in Chapter 4 consist of either the arrival of
a node at its chosen waypoint, or the creation or removal of a link between a pair
of nodes. \(n^2\) is the combination of the running time of the partition detection algo-
rithm, performed by a DFS-traversal of the network graph, along with the network
state updates at discrete event points (or rolling back to a previous state when parti-
tion is detected). The \(n \log n\) component relates to the implementation of the event
scheduler. A majority of earlier model’s implementation of scheduler use a linked list
representation that requires \(n^2\) time to update and extract the next event’s time. I
substantially reduced this overhead by using a combination of heap and a hash table
that indexes the events in the heap (Section 4.2). This improves the rescheduling of
events while avoiding the unnecessary computation of the unchanged events’ time.

The most recently accessed graph structures are stored in an LRU cache to more
efficiently compute the status of the network graph at different events. The caching
employs a hash table to determine whether a graph structure can be found in the
cache. If it is, the partition check algorithm is eliminated, and the partition status
can be retrieved in constant \(\mathcal{O}(1)\) time for the cache lookup.

\(\mathcal{O}(n)\) is the factor associated to the intersection algorithm which runs in \(\mathcal{O}(n)\) for
computing the wedges’ density around each node (Section 4.3).
The extended version works very similar to the standard version with the exception that Floyd-Warshall all pairs shortest path algorithm is run to compute the minimum hop count distances between distances, as the network graph state changes. The running time of Floyd-Warshall algorithm is $n^3$ which results in the following asymptotic complexity for the extended scenario generator:

$$O\left(|E'| n^3 + |E| (n^2 + n \log n + n)\right),$$

from state to another state Floyd-Warshall all pairs shortest path algorithm. As was described in Section 4.2, the most recently accessed graph structures are stored in a LRU cache to more efficiently compute the status of the network graph at different events. The caching employs a hash table to determine whether a graph structure can be found in the cache, in which case the partition status of the graph as well as the shortest paths (for the extended version) are retrieved in constant $O(1)$ time. This caching has a much more prominent advantage in case of using the extended version, since the dominant cost of the model is the Floyd-Warshall algorithm; by reducing the effect of the $n^3$ complexity, there will be a proportional reduction in the execution time of the scenario generator. The term $|E'|$ refers to the number of cache misses in retrieving the graph structure which is smaller than total number of the events in the simulator ($|E|$).

5.3 Experimental Evaluation of the Sequential Version

To examine the running time efficiency, we ran experiments for a wide range of density control parameter values and number of nodes (Figure 5.5). The experiments were conducted on a machine that has Intel Xeon E5-2640 running @ 2.50 GHz, with 64
60

GByte main memory. Figure 5.5a shows the running time for the standard sequential version. The graphs are shown for different density control parameter values, and for different number of network nodes (from 25 to 200 nodes), and the simulation is performed for constantly moving nodes over 1000 seconds of simulation time. The generation of scenarios takes less than 100 seconds to finish for number of nodes up to 100. For larger network, the generation takes at most 800 seconds (13.3 minutes) to finish. In the standard version, the cost of generating scenarios generally decreases as the density level goes higher (network scenarios become denser). This is because in the sparse networks, vibration happens much more frequently, causing the generation of a large (but limited) number of events in a short span of time, until the vibration is detected. Whereas, in denser networks, such partitioning occur much less often, and node motion is generally successful towards its chosen waypoint. Figure 5.5b illustrates the running time for the case of the extended version. The generation of scenarios take a lot longer because of the much more expensive Floyd-Warshall algorithm that is run for gathering shortest distance information. This graph also shows the opposite behavior of running time as the density increases. This is due to the fact that the cost of the scenario generation is dominated by the Floyd-Warshall algorithm, which is $O(n^3)$. Sparse networks benefit could benefit greatly from caching of the graph structure because many of the disconnected graphs are recent and already existing in the LRU cache. Denser scenarios, on the other hand, cannot benefit as much from caching because of the many more topology changes that happen more frequently.

In the next section, I describe a parallel version of the model implemented that can greatly reduce the generation cost in both versions.
Figure 5.4

5.4 Parallel RCS Implementation

In order to improve the cost of the scenario generation, two major parts of the model are parallelized, which induce the largest of the overhead. Before discussing the de-
tails of the parallelization, designing a parallel version of a discrete event simulator such as RCS is very difficult. This is specially the case for RCS, because the model dictates that the computation of the events performed in a sequential order. There is cause/effect relationship across both the time and spatial domain of the scenarios, and the simulation must adhere to the generation of events according to their timestamps. Consequently, any parallelization should be done very carefully in order to still maintain the same type of behavior while not breaking the original cause/effect relationships among events.

Here, I focus on two major computationally expensive components of the scenario generation, which take up a large portion of the running time. The first one is the generation of the vibration events for a vibrating node, and the second is the Floyd-Warshall algorithm that takes a significant portion of time in dense networks.

OpenMP was used in order to split the workload among different threads. OpenMP is a powerful tool that supports multi-platform shared memory parallel programming in C, C++, and Fortran. It provides a set of portable compiler directives and library function, in which different sections of the code can be specified for parallel execution. There are also a number of constructs used for data race protection, defining local state variables for each thread, as well as splitting up tasks among different threads.

In the following section, the parallelization technique used for RCS is detailed.

5.4.1 Parallel Computation of the Vibration Checks

In the discussion of the sequential standard version in Section 5.3, it was noted that in sparse networks, a lot of vibration events are generated per bridge node, causing the generator to run slower as the number of nodes grows. These node sleeps for a 1 second before waking back up to re-check for vibration. Since the rate of these
periodic checks is high compared to the topology changes, it can induce runtime overhead. This part of the scenario generator is a good candidate for parallelism, and the work of checking for vibration can be divided among different threads because of the following: all the nodes that are sleeping because of vibration will wake up in a 1-second interval. This is the case since all the vibrating nodes follow the same rule of sleeping for 1 second when the vibration is detected (Section 3.6.1). The argument for choosing 1 second sleep time was that it is small enough so that the network does not skip over too many changes that limits a node’s mobility while allowing a forward simulation progress. The same argument applies in case of processing sleeping nodes concurrently; the network graph does not go through many changes in 1 second therefore, the state of the network graph at the beginning of a 1-second interval can approximate the network graph during that 1 second. The parallel version uses this fact to run individual independent copies of the scheduler for each sleeping node. While each one of these independent copies are run by separate threads, the original global sequential scheduler remains intact. At the end of the processing of the events corresponding to the sleeping node (which goes on for at most 1 millisecond), it is either determined that the node needs to go to sleep again, or the node is able to start moving again. In the case of the former, the wakeup time of the processed node gets updated in the main sequential scheduler with its original wakeup time in the main scheduler plus 1 second. This is the only phase that the shared data structure (the main scheduler) is being touched, and thus needs to be protected for data race.

In the implementation, each thread handles the processing of 3 sleeping nodes. This is so that each thread has enough work to do and the overhead of parallelism does not take over its performance gains. Figure 5.6 shows the execution path of the scenario generator before and after the concurrent execution of the sleeping threads.
The main global scheduler is the main scheduler that runs the simulator. Upon processing a wakeup event for a sleeping node, the set of all the current sleeping nodes is gathered. If the number of sleeping nodes exceeds 3, it provides the simulator the opportunity to run the wakeup events for the sleeping nodes in parallel. Here the main thread spawns \( \min(\lceil \frac{n}{3} \rceil, P) \), where \( P \) is the number of processors (or cores) in the hardware. Each thread then handles executing the events in a 1 millisecond interval for 3 sleeping nodes sequentially. At the end of the parallel execution, each thread then decides to update the global sequential scheduler based on whether or node the node vibrates. If vibration is detected, the node wakeup timestamp is updated in the global scheduler, otherwise nothing is changed.

5.4.2 Parallel Computation of Floyd-Warshall Algorithm

The second expensive part of the simulator is the execution of Floyd-Warshall algorithm in the extended version of the scenario generator. To more efficiently execute the algorithm, I parallelized it using a row-wise decomposition of the intermediate distance matrix. The outer loop in Floyd-Warshall iterates over the intermediate vertices from whom the shortest path among pairs have so far been computed. This outer loop must run sequentially since the result of parallel execution of multiple sets of intermediate nodes cannot easily be combined. However, the first inner loop has the potential of being parallelized. In particular, the second loop which iterates over all the rows can run concurrently across multiple threads since the results of these runs are independent of each other.

By decomposing the distance matrix into contiguous rows, each thread is responsible for computing and updating the distance information of its assigned block of rows. Figure 5.7 shows such row-wise decomposition of the distance matrix. In addi-
Figure 5.6 The execution path of the scenario generator before, while, and after performing vibration checks in parallel.

To put this all together, the code fragment in Figure 5.8 describes the parallel implementation of the Floyd-Warshall algorithm.

In the code, \( \mathbf{1} \) is the distance matrix of a graph of size \( n \) stored as a one-dimensional array since it offers better memory locality, resulting in faster memory accesses. At
the end of the algorithm, $l$ holds the minimum shortest paths among all the network pairs.

### 5.5 Experimental Evaluation of the Parallel Version

In this section, I present the running time evaluation of the parallel versions of the scenario generator, as was detailed in the previous section. Similar to the sequential runs, the scenarios were generated across a wide range of node numbers and density control parameters. The graphs of the running time are shown in Figure 5.9. To put these comparisons in the context of the sequential version, two major improvements are observed in the running time of the scenarios.

In case of the standard version, the performance gain is more noticeable towards the sparse networks, where a parallel speedup of 2.2 is observed (Figure 5.9a). For example, scenarios of density control parameter of 0.1 take 350 (less than 6 minutes)
for(int k = 0; k < n; k++)
{
    # pragma omp parallel
    {
        int *kth_row = (int *) malloc(n * sizeof(int));
        memcpy(kth_row, &l[k * n], n * sizeof(int));

        # pragma omp for
        for(int i = 0; i < n; i++)
        {
            int lik = l[i * n + k];
            for(int j = 0; j < n; j++)
            {
                int lij = l[i * n + j];
                if ( lik + kth_row[j] < lij )
                    l[i * n + j] = lik + kth_row[j];
            }
        }
        free(kth_row);
    }
}

Figure 5.8  OpenMP-based implementation of parallel Floyd-Warshall algorithm

seconds to generate for a 1000 second simulation time, as opposed to 13 minutes for
the sequential version. This gain comes from the concurrent processing of the sleeping
nodes as was discusses.

For the extended version, such performance gains are observed for both sparse and
dense networks, where a speedup of close to 2 is achieved (Figure 5.9b). This gain
is the result of parallelizing the Floyd-Warshall algorithm, which is the dominating
cost of the scenario generation in the extended version, as well as the use of multiple
threads for processing sleeping nodes concurrently.
Figure 5.9 Running time of the parallel scenario generator for a simulation time of 1000 seconds, across different range of density levels and number of nodes

5.6 Chapter Summary

In this chapter, I first evaluated RCS in terms of its mobility characteristics, and to show the application of the model in evaluating a real protocol, I studied the
performance of DSDV across different degrees of density levels. The results showed that the performance of DSDV is significantly worse in sparse networks than the typical scenarios generated by Random Waypoint model, which are dense. Also, metrics such as path optimality was highly variable in such networks, where the current definition of this metric turns out to be insufficient for protocol evaluation.

This chapter also presented the performance study of the model in terms of its asymptotic complexity analysis of the model, as well its the experimental evaluation of its running time across a wide range of density values and number of nodes. Additionally, a parallel version of the model was introduced that improves the running time of the expensive components of the model in both standard and extended versions. The generation cost of the parallel version was presented which achieved a speedup of close to 2 in both versions.
Chapter 6

Protocol Design and Performance Study using RCS

To more fully show the use of the Random Controlled Sparse mobility model (RCS) in real protocol design and evaluation, I present the design and evaluation of the Local-Approximation Multicast Protocol (LAMP), a new on-demand multicast routing protocol for mobile wireless ad hoc networks.

Prior work in multihop multicast routing has generally attempted to provide good packet delivery performance with low overhead. These protocols, however, have often been designed so that they perform well under generally dense types of networks; the optimizations made (such as the use of link-layer multicast or broadcast transmission for data packets) often turn into a liability in networks with sparser average node densities, as link-layer multicast or broadcast transmission is inherently less reliable than unicast transmission. In contrast, LAMP is a unicast-based multicast protocol that delivers high performance in both sparse and dense scenarios. Routing in LAMP is based on a new algorithm in which each intermediate node utilizes a local approximation of the globally optimal multicast forwarding tree. To improve routes in cases in which this local approximation deviates from the global optimal multicast tree, LAMP also introduces a new distributed optimization known as anticipatory forwarding, improving both the overhead and the delivery latency of the protocol.

I have evaluated LAMP through detailed ns-2 simulations using scenarios from both RCS and Random Waypoint mobility models. LAMP’s performance is com-
pared with that of ODMRP [23] and ADMR [17], two prior on-demand multicasting protocols that both rely on link-layer multicast or broadcast transmission for data packets; both protocols have also been previously well studied and shown to perform well in the typical relatively dense scenarios generated by the Random Waypoint mobility model. LAMP generally equaled or outperformed both protocols for dense as well as sparse scenarios on all metrics.

The rest of this chapter is organized as follows. Section 6.1 gives an overview of the current multicast protocols. Section 6.2 provides a general overview of LAMP and how the intermediate nodes delegate receivers to each other. In Section 6.3, I detail the local approximation algorithm utilized by each intermediate node in order to find the most efficient set of next neighbors to forward the multicast packet to. Section 6.5 discusses the distributed optimization known as *anticipatory forwarding* that improves the route efficiency and latency in cases where the local approximation deviates from the globally optimal multicast forwarding tree. Section 6.6 presents the simulation setup for evaluation of LAMP vs. the other multicast protocols, in which scenarios of different density levels are generated using RCS as well as Random Waypoint mobility models. Finally, Section 6.7 presents the performance results of LAMP in comparison with other protocols. In particular, I highlight how the other multicast protocols’ performance degrade severely in sparse networks, where such behavior could not be observed using Random Waypoint mobility model, and how LAMP overcomes such performance drop by the mechanisms described in the prior sections.
6.1 Overview of Ad Hoc Network Multicast Protocols

Multicasting, the dissemination of a data packet to a group of receivers in a network, has been well studied in the context of wired networks, and many wired multicasting protocols have been proposed in the literature. However, due to the substantial differences between wired and wireless ad hoc networks, wired multicasting protocols cannot easily be adapted for use in wireless ad hoc networks. The low transmission quality, dynamic behavior of the network nodes, and low bandwidth in such networks require different and more efficient handling of multicast routing. A number of multicast routing protocols have been proposed for ad hoc networks (e.g., [31, 18, 40, 41, 9, 33, 17, 23, 36]), most of which try to form a mesh or a tree containing the set of senders and the receivers of the group. Most of these protocols are proactive protocols, but some have on-demand (reactive) nature [23, 19, 18, 33, 17]. Pacifier was introduced in [21] for reliable multicasting in mesh networks and the author in [26] presented a simulation-based performance comparison of shortest path trees (SPTs) and minimum cost tree (MCTs) for wireless mesh networks.

Unlike mobile ad hoc networks, nodes in wireless mesh networks have no mobility and consequently the protocols tailored for these types of networks are not generally suited for mobile networks of varied densities.

For example, ODMRP [23] is a widely-studied on-demand multicast routing protocol for ad hoc networks. Active multicast sender nodes in ODMRP periodically flood a JOIN QUERY packet throughout the network, and each each receiving member of the multicast group returns a JOIN REPLY packet along the reverse path toward the sender. Each node that forwards the REPLY becomes part of the forwarding group mesh of nodes for that multicast group and remains part of the forwarding group as long as it has forwarded a REPLY within the latest few (e.g., 3) periodic JOIN QUERY
flood intervals. Multicast data packets sent from this sender node to the group are then flooded by the members of the forwarding group mesh to reach all group receivers. ADMR [17], another on-demand ad hoc network multicast routing protocol, uses a similar forwarding group mesh flooding mechanism for multicast data packet transmission but uses of a more complex, adaptive set of mechanisms for building the forwarding group mesh and maintaining it as the network topology or multicast group receiver membership changes; these mechanisms, such as local routing state repair and adaptation to the sender’s traffic pattern, attempt to reduce overhead and improve performance. The performance of ADMR and ODMRP have previously been compared, and both perform well [17].

6.2 LAMP Protocol Design

Achieving globally optimal multicast routing, resulting in the fewest total number of forwarding transmissions of a multicast data packet to allow it to be received by all members of the multicast group, is a difficult problem. For example, Figure 6.1 illustrates a simple ad hoc network in which source node $S$ wants to send a multicast packet to a group with two receivers, $R_1$ and $R_2$. The optimal multicast tree for delivering the packet to these two receivers consists of nodes $S$, $Q$, $T$, $R_1$, and $R_2$, resulting in a total of 4 transmissions of the data packet. If instead, the alternative shortest paths on either side of $S$ to each individual receiver node (through $M$ and through $O$) is used for forwarding, a total of 6 transmissions would be required to reach both receivers.

Finding the globally optimal set of forwarding nodes is a form of the minimum Steiner tree problem and is NP-complete [20]. In addition to its computational complexity, solving this problem also requires global knowledge of the network topology,
or to be implemented as a distributed protocol, would require this knowledge to be known consistently at all nodes in the network. Such globally consistent knowledge is in many cases infeasible in a wireless mobile ad hoc network, due to the potentially frequent (or even continuous) network topology changes, the generally low bandwidth of the wireless links, and the resource limitations (e.g., energy, CPU, and memory) of the nodes in the network.

LAMP is a receiver-initiated multicast protocol that works entirely on demand. Each receiver node begins a group join by flooding through the network a one-time Rcvr-Join packet; likewise, a node leaves a multicast group by flooding a one-time Rcvr-Leave packet. Rcvr-Join and Rcvr-Leave packets have a sequence number field generated by the originating receiver node. This sequence number is stored at recipients of these packet types, as the pair (receiver id, sequence number) is used as a packet identifier in order to forward each of these packets only once. The sequence number is also used to maintain up-to-date group-membership information about the receiver members, by having packets with higher sequence numbers supersede those with lower ones. For instance, if a node has received a Rcvr-Join packet originated by some receiver node and later receives a Rcvr-Leave packet for the same node...
with a lower sequence number, the node discards the second packet and still considers the receiver as a member of the group.

The RCVR-JOIN packets establish the initial routes between the nodes in the network (some of which might become senders for this group) and the receiver members of the group. The routes are further maintained between future group senders and the receivers by an on-demand multi-route unicast protocol. A multicast packet begins at some sender and branches out at different intermediate forwarders based on a local approximation of the optimal multicast tree at the intermediate nodes, with each forwarder being responsible for a different set of receiver members, until the packet arrives at the intended group members. Distributed optimization of the routes is performed throughout LAMP’s operation, improving the routing decisions made at the intermediate nodes.
An example of multicast data packet forwarding in LAMP is depicted in Figure 6.2. Along each link between the multicast forwarders in this figure is shown the set of receivers that are delegated to the node at the endpoint of that link. For example, node $S$ delegates receivers $R_1$, $R_2$, and $R_3$ to node $M$, and delegates $R_4$ and $R_5$ to node $T$; each node in turn chooses a set of its own neighbors to handle the receivers for which it is responsible, until all group receiver members receive the packet. Choosing the set of neighbors to deliver the packet to their associated receivers is based on a local approximation (of the globally optimal multicast tree) made at each intermediate node, which is described in the following section.

### 6.3 A Local Approximation of the Optimal Multicast Tree

In the local approximation problem, a source node $S$ wants to send a packet to a multicast group consisting of receivers $R_1, \ldots, R_n$. Each one of the node’s neighbors has a cost in reaching each receiver member defined as the cost of the shortest path known by this node between the neighbor and the receiver. In case of no routes to a receiver, this cost is defined as infinity. Based on this local knowledge, a subset of neighbors is found through which we can hit all the receivers that minimizes the overall cost of packet delivery. A metric is defined capturing the essence of the optimal multicast tree as seen by each node locally.

Referring again to Figure 6.1, for source node $S$, neighbor $Q$ has the same cost of reaching each of the two receivers in the multicast group as does neighbor $M$ for receiver $R_1$ or neighbor $O$ for receiver $R_2$. However, since both receivers are reachable through the same neighbor $Q$, the distance to both of them is reduced simultaneously by one transmission. On the other hand, by using neighbor $M$ for transmission to $R_1$
and neighbor $O$ for transmission to $R_2$, every reduction in hop counts along the path to each receiver requires a separate transmission.

Intuitively, to find an efficient route based on local knowledge about the receivers, we should reduce both the cost to the receivers and the number of neighbors involved, collectively. Thus, the first local component of the metric is defined as the number of hops between a node's immediate neighbors and the receivers. The second component is the number of total neighbors a node selects for forwarding the packet or, more generally, the sum of the link costs associated with the selected neighbors. Providing this local information is the job of the underlying multi-route unicast routing protocol, detailed in Section 6.4. The local metric is denoted by $\zeta$ and is equal to the sum of the two components above, which corresponds to an assignment $A$ that assigns each receiver to a neighbor: $\zeta_A = \sum_{r \in R} c'_r + \alpha \times |\eta_A|$, where $R$ is the set of receivers, $c'_r$ the cost between a receiver to its associated neighbor, $\alpha$ the cost of using each neighbor, and $|\eta_A|$ the number of total neighbors selected in assignment $A$; for simplicity the same cost $\alpha$ is taken for all the neighbors.

The local approximation thus chooses an assignment $A$ of receivers to neighbors that minimizes the $\zeta$ cost. In the next section, the upper bound on the total number of transmissions achieved by this local approximation is established.

### 6.3.1 Upper Bound Analysis

I prove the following upper bound for the local approximation of the optimal multicast tree in terms of the cost of the multicast tree (i.e., the total number of transmissions required to disseminate the data packet to all the receiver members, represented by $|M|$):

$$\sum_{r \in R} d_r - \alpha \left( |R| - \min \{ |R|, |N| \} \right)$$
where $\sum_{r \in R} d_r$ is the sum of the minimum connection costs from the node to the receivers, $\alpha$ is the neighbor cost, and $N$ and $R$ are the set of neighbors and receivers, respectively.

**Proof of Upper Bound**

Let $N$, $R$, and $\alpha$ be the set of node’s neighbors, receiver members, and the neighbor cost, respectively. An assignment $A : R \mapsto N$ is a function from the set of receivers to neighbors. The image of $R$ under $A$ is denoted by $\eta_A$, which is equivalently the set of neighbors selected in assignment $A$.

An assignment $A$ is made up of $(n, D_A(n))$ pairs, where $n$ is a neighbor and $D_A(n)$ is the set of receivers assigned to this neighbor in assignment $A$. The set of receivers constituting the first element of these pairs (which are also selected neighbors in this assignment) is represented by $\eta_A$ (the assignment index $A$ is usually dropped whenever the context is clear).

$$F_A(r) = n, \text{ for } r \in D_A(n)$$

The following symbols are also used throughout the proof:

- $c_{nr} = \text{minimum connection cost between } n \text{ and } r$
- $c_{r \in R} = \min_{n \in N} c_{nr}$
- $c'_{A, r \in R} = c_{nr} \text{ such that } A(r) = n$
- $d_{r \in R} = \alpha + c_r \text{ is the min distance of node from } r$
The cost of an assignment $\mathcal{A}$ is the sum of the connection costs $c_{ij}$ and the neighbor costs $\alpha$ (the notation $|\ldots|$ indicates the cardinality of a set): $\zeta_\mathcal{A} = \sum_{r \in R} c'_{A,r} + \alpha \times |\eta_\mathcal{A}| = \sum_{n = A(r)} c_{nr} + \alpha \times |\eta_\mathcal{A}|$.

**Theorem 6.1**

Let $|M|$ be the cost of the multicast tree obtained by the local approximation, and $\eta_\mathcal{A}$ be the set of selected neighbors at the source node. Then, the upper bound limit for $|M|$ is

$$\sum_{r \in R} d_r - \alpha (|R| - \min \{|R|, |N|\})$$

where $\sum_{r \in R} d_r$ is the sum of the minimum connection costs from the node to the receivers.

**Proof 6.1** Construct the assignment $\mathcal{A}'$ consisting of the neighbors with minimum connection costs to receivers:

$$\mathcal{A}' : R \mapsto N \text{ where } \forall r \in R \ (c'_{A',r} = c_r)$$

The number of assigned neighbors in any assignment cannot be larger than the minimum number of entries in $R$ and $N$. In other words, $|\eta| \leq \min \{|N|, |R|\}$. Since $\mathcal{A}$ is optimal:

$$\zeta_\mathcal{A} \leq \zeta_\mathcal{A}' \Rightarrow \sum_{r \in R} c'_{A,r} + \alpha |\eta_\mathcal{A}| \leq \sum_{r \in R} c'_{A',r} + \alpha |\eta_\mathcal{A}'|$$

$$= \sum_{r \in R} c_r + \alpha |\eta_\mathcal{A}| = \sum_{r \in R} c_r + \alpha |R| - \alpha |R| + \alpha |\eta_\mathcal{A}'|$$

$$= \sum_{r \in R} d_r - \alpha |R| + \alpha |\eta_\mathcal{A}'|$$

$$\leq \sum_{r \in R} d_r - \alpha |R| + \alpha \min \{|N|, |R|\}$$
But $\zeta_A$ is the cost of the multicast tree formed by the local approximation algorithm. Therefore, the bound is reduced to:

$$|M| \leq \sum_{r \in R} d_r + \alpha (\min \{|N|, |R|\} - |R|)$$

which completes the proof.

### 6.3.2 Solving the Local Approximation Problem

The local approximation problem is NP-complete. I present two different approaches to find the minimal solution.

The first approach, which is often feasible, is simply a brute-force method. In this method, the algorithm tries all the possible assignments of neighbors to receivers and chooses the one assignment minimizing the $\zeta$ cost. The feasibility is due to the bounded number of states the brute-force approach should visit in most cases. The number of states equals $|R|^N$.

Due to the locality of the neighbors in the local approximation, the number of states remains small enough for the brute-force approach to be solvable in polynomial time for the networks of the same density, but of different sizes. However, the number of states in the minimum Steiner tree problem is proportional to the total number of nodes and hence cannot scale well with the number of network nodes. Thus, the brute-force approach for the real global multicast tree optimization remains impractical.

In case of having large number of neighbors or receivers, I present a second approach for the linear-time approximation of the optimal solution to local approximation problem. This is done by mapping the local approximation problem to a facility location problem as discussed below.
In the facility location problem, there are a set $F$ of $n_f$ facilities and a set $C$ of $n_c$ cities. The cost of opening each facility $i$ is $f_i$, and the service cost (connection cost) of facility $i$ to city $j$ is $c_{ij}$. The goal is to find a set of facilities to service all cities such that the total cost of opening facilities and their connection costs to the cities serviced is minimized.

The local approximation employs the following mapping. The set of neighbors represents the set of facilities, and using each neighbor incurs a cost as does opening a new facility. The set of cities corresponds to the set of receiver members belonging to the multicast group, and the service cost of a facility for a city is the number of network hops between the neighbor and the receiver. Finally, our goal in the original problem was finding the most cost-efficient set of neighbors through which all the receivers can be reached in the same way that we want to provide services to all the cities through opening facilities in the facility location problem. Therefore, to find the best set of neighbors, we can solve its counterpart problem, the facility location problem, giving us the set of neighbors and the receivers for which they are responsible. Several good approximation algorithms exist that solve the facility location problem in linear time, giving approximations very close to the optimal solution. The greedy approach mentioned by Jain et al. [14] is one example of such linear approximations, which I used in the implementation for cases where the number of neighbors is large. This greedy approach solves the facility location problem in time $O(n^3)$, where $n = \max(n_f, n_c)$, and achieves an approximation guarantee of 1.61 times the cost of the optimal solution.

In the local approximation, as discussed, a source node typically knows multiple paths along with their costs to receiver members through different neighbors. In Section 6.4, I introduce a new loop-free routing protocol for this purpose.
6.4 Maintaining Unicast Routing Information

This section presents a multi-route unicast routing protocol used to maintain routes to individual receiver members in LAMP. Since the local approximation made at nodes is based on local knowledge, the routes provided by the underlying multi-route unicast protocol must be loop-free, creating a directed acyclic graph (DAG) in the network. Nodes can be mobile, and hence the links can become broken; therefore the connectivity of this DAG must be maintained as network condition changes. Moreover, I make no assumption on the reliability of transmissions; packets can be dropped due to reasons such as collisions, link failures, signal interference. The protocol thus must be resilient to such conditions. Designing loop-free multi-route protocols in mobile ad hoc networks is challenging. For example, although TORA [28] was “proven” loop-free, it was later found not to be. Routing loops in TORA can happen in two circumstances: first, when UPD (update) packets in the link-reversal process are lost and hence nodes no longer have a consistent view of the direction of the link between them; and second (which is more common), in the meantime of UPD transmissions, since the network is not yet stable during those intervals. The fixes that have been proposed for TORA were able to solve loops that occur due to the first reason above, but they were unable to solve this second cause of loops. The underlying unicast protocol in LAMP guarantees loop-free routes, despite packet drops and the latency of control packet transmissions.

Two control routing packets are used: ROUTE-BROKEN and ROUTE-REPLY. Both packet types are small and efficient in terms of link bandwidth, while achieving the routing goal of maintaining multi-route loop-free routing tables for the nodes. Nodes initially obtain routes through the propagation of the Rcvr-Join packets, as described in Section 6.2 (the conditions for adding a route are given later). Ev-
Every node maintains a unique identifier (UID) for each receiver for which it keeps
routes. Link breaks are detected on-demand, either by link-layer acknowledgments or
through passive acknowledgments. Upon detecting a broken link, a node removes the
link from its routing table, and, upon losing all the routes for some receiver, the node
broadcasts a Route-Broken packet to its direct neighbors. In sending this packet,
the node also generates a new UID for that receiver if it has sent a Route-Reply
packet previously that relied on its old routes. A Route-Broken packet includes
only the IP address of the receiver or receivers for which all routes have been lost.

Nodes receiving the Route-Broken packet delete the routes to the receiver that
pass through the sender of the Route-Broken. If such a node still has at least one
route for the receiver, it generates a Route-Reply packet in response. Otherwise,
the node also generates a Route-Broken and a new UID, as described above. In
addition to generating a route-broken, the node remembers the neighbor from
which it received the Route-Broken. When, later, this node obtains a route to
the receiver and has not received signs of having routes from this neighbor (e.g., a
Route-Reply from the neighbor indicating that the neighbor has obtained a route),
it generates a Route-Reply packet.

Each Route-Reply packet includes the minDist and maxDist and the UID
corresponding to the receiver. The minDist and maxDist metrics are the minimum
and maximum number of hops, respectively, between the node and the receiver in the
DAG. The minDist is used as the connection cost between a neighbor and receiver
in the local approximation, and the maxDist enables neighboring nodes to add more
loop-free routes to their routing table, as explained later.

A node receiving a Route-Reply from its neighbors that decides to add the route
to its routing table (Section 6.4.2) create a new entry for this receiver and neighbor
(if it does not already exist). The minDist and maxDist metrics are incremented and stored in this entry in addition to the UID received in the ROUTE-REPLY. Therefore, associated with a route to a receiver through a neighbor is a UID that is generated by that neighbor and stored at the node. The UID is used to avoid loops due to inconsistencies in the knowledge of the nodes about the network graph, as described in the next section.

6.4.1 UID Mechanism

A node wishing to send a packet to a receiver through a next neighbor tags the data packet with the UID corresponding to the receiver received from that neighbor. The recipient node then compares its current UID for that receiver with the received UID and verifies that they match. In case of a mismatch, the node will not forward the packet further, as such inconsistencies can possibly be an indication of a loop in the network routes. Instead, the node backtracks the packet to the previous node that had an out-of-date knowledge about this node’s route. The backtracked packet includes the current UID of the node corresponding to that receiver.

On receipt of this backtracked packet, this previous node first deletes its old route to the receiver through that node and then inspects whether it has any remaining alternate routes for that receiver. If it has other routes, it forwards the packet along one of those paths. Otherwise, when it has no other routes to that receiver, the node would normally generate a ROUTE-BROKEN packet; however, instead of actually generating a separate ROUTE-BROKEN packet, the node instead backtracks the original packet to the previous hop from which it received that data packet. Since all backtracked packets include the UID corresponding to a receiver, a node overhearing the backtracked packet can determine whether or not it has an up-to-date route to
the receiver through the node, by comparing its stored UID for that route with the UID included in the packet. If there is a mismatch between the two, it deletes its stored routes through that neighbor. This backtracked packet will thus serve as both a ROUTE-BROKEN packet and a real data packet that needs to be forwarded to the receiver eventually. The backtracking of the packet continues either until one of the intermediate nodes on the backtracked path is left with a route to the receiver or, in the worst case, until the packet arrives back to the original sender of the packet and restarts its forwarding from that point on.

6.4.2 Adding Loop-free Routes

On receiving a ROUTE-REPLY from a neighbor, the route is added to the routing table if either of the following two conditions are satisfied: (1) the node has not sent any ROUTE-REPLIES for the receiver since it last lost its routes to the receiver; or (2) the node has sent a ROUTE-REPLY for the receiver after it lost its routes, but the maxDist of the neighbor to the receiver is less than or equal to its minimaxDist. Before introducing the maxDist and minimaxDist metrics, it is worth mentioning that the simple minimum hop count metric traditionally used in ad hoc networks for adding loop-free routes does not work in the DAG paradigm, as it can produce both false-positive and false negative results.

The false positive case occurs when a node does not add a loop-free route that it erroneously thinks that leads to a loop. For instance in Figure 6.3, node E would not add the link to C, since it thinks that it can reach the receiver in fewer hops (1 hop) than C (3 hops). However, such a link does not create a loop and is thus safe to be added. In the false negative case, a route gets added by the node that creates a loop although the node believes that it is a safe route. In the same example, C
would add the link to $E$, since it observes that $C$ is closer to the receiver than itself in the minimum hop-count sense. However, by adding this link, a loop would be created through the path $E-D-C-E$. Having such loops in the network can create serious troubles, since later, if links $E-R$, and $C-B$ break, nodes $E$ and $C$ will still think that they can reach the receiver through $D$ and $E$, respectively, leading to the $E-D-C-E$ loop.

The reason for the $E-D-C-E$ loop in the example above is that node $E$ uses the longer route $E-D-C-B-A-R$ passing through $C$ in addition to its shorter route $E-R$. However, $C$ cannot be aware of that loop by relying on the minimum distance of $E$ from $R$.

I define two metrics, $\text{maxDist}$ and $\text{minimaxDist}$, together used to decide on which routes are safe to be added. The first one, $\text{maxDist}$, represents the maximum distance of each node from the receiver. This metric is computed based on the $\text{maxDist}$ of a node’s downstream nodes to the receiver:

$$\text{maxDist}(A) = \max_{(A,B) \in E} \{\text{maxDist}(B) + 1\}$$
minimaxDist is defined as

\[
\text{minimaxDist}(A) = \min_{t \in T} \{\text{maxDist}(A)\}
\]

The minimum is taken over all the times since this node has sent a ROUTE-REPLY to its neighbors since it last lost all its routes to the given receiver (\(T\) is the interval between now and the time when the node last lost all the routes to receiver \(r\)). A node \(A\) adds a link \(A - B\), if the advertised \(\text{maxDist}\) of its neighbor, \(B\), to the given receiver is less than or equal to that of its minimaxDist:

\[
\text{maxDist}(B) \leq \text{minimaxDist}(A)
\]

Figure 6.3 is now revisited in which the minimum hop-count metric failed. For simplicity of description, minimaxDist of all nodes is assumed to equal their current maxDist. Here, node \(C\) will not add the link \(C - E\) since the maxDist of \(E\) is 5 (corresponding to the longest route \(E - D - C - B - A - R\)), which is longer than its own minimaxDist of 3. On the other hand, \(E\) can add the link \(E - C\) to its routing table since \(C\) has a maxDist of 3 that is less than its own minimaxDist of 5, whereas in the minimum hop-count scheme above, such a route is not added.

The addition of links according to the two rules described in this section, along with the UID operation in Section 6.4.1, guarantee that no loops are created throughout the protocol’s operation, as established in the following proof.

**Proof of Loop-Freedom**

Let \(\text{maxDist}(A)\) represent the maximum distance between node \(A\) and the receiver in the network graph (Section 6.4.2), and \(\text{minimaxDist}(A)\) be the minimum of the
$maxDist(A)$ of the node over all the times when a route-reply was generated since a node last lost all its routes.

I first establish the following two lemmas:

**Lemma 6.1**

A routing loop exists if starting at one node in the loop and traversing through the link directions, we get back to the starting node and all the link UIDs match with the nodes’ UID at the endpoints of the links.

**Proof 6.2** The proof follows from the protocol operation. Whenever a uid mismatch occurs while the packet is being forwarded, the packet is backtracked to the previous node so that the previous node fixes its current routes or use alternate routes. Therefore, the packet never gets stuck in a routing loop if a uid mismatch exists over the links contained in the loop.

**Lemma 6.2**

For any link $A-B$ in the network graph at each time instant, $\text{minimaxDist}(A) \geq \text{minimaxDist}(B)$ and $\text{maxDist}(A) \geq 1 + \text{minimaxDist}(B)$.

**Proof 6.3** According to the definition of $t_{xy}$, node $x$ has the information about the $x-y$ link in its routing table in the interval $(t_{xy}, t_n)$, thus all its route-reply packets sent in this interval should have a maximum distance larger than $\text{maxDist}_{xy}$. Route-reply packets sent in the interval $(t_x, t_{xy})$ can however have a lower maximum distance, but they cannot be more than one-hop better than $\text{maxDist}_{xy}$; otherwise, according to the protocol operation, the link $x-y$ will not be added at time $t_{xy}$.

Therefore, 2 cases should be considered: The first case happens when a route-reply packet with maximum distance of $\text{minmaxDist}_x$ is sent in the interval $(t_x, t_{xy})$; but no route-reply packets with such distance is sent after this point on (it’s either
because node $x$ does now send a route-reply at all after $t_{xy}$, or its maximum distance metric grows in this interval, which as just mentioned before, can be at most 1 hop greater than $\text{minmaxDist}_x$). The second case corresponds to when a route-reply packet with the maximum distance of $\text{minmaxDist}_x$ is sent in $(t_{xy}, t_n)$. Since such a reply is sent when node $x$ possesses the $x-y$ link, $\text{minmaxDist}_x$ should be at least as worse as $\text{maxDist}_{xy}$. Thus, corresponding to each one of these 2 cases, we have the following set of inequalities:

\begin{align*}
\text{case 1: } \quad \begin{cases} 
\text{minmaxDist}_x &\geq \text{maxDist}_{xy} - 1 \\
\text{maxDist}_x &\geq \text{maxDist}_{xy} \geq 1 + \text{minmaxDist}_y
\end{cases} \\
\text{case 2: } \quad \begin{cases} 
\text{minmaxDist}_x &\geq \text{maxDist}_{xy} \geq 1 + \text{minmaxDist}_y \\
\text{maxDist}_x &\geq \text{maxDist}_{xy} \geq 1 + \text{minmaxDist}_y
\end{cases}
\end{align*}

Therefore, for any $x-y$ link in the graph, the following relation exists between the $\text{minmaxDist}$ and $\text{maxDist}$ of node $x$ and $\text{minmaxDist}$ of node $y$:

\begin{align*}
\text{minmaxDist}_x &\geq \text{minmaxDist}_y \\
\text{maxDist}_x &\geq 1 + \text{minmaxDist}_y
\end{align*}

which completes the proof. □

Theorem 6.2

The routing is loop-free.

Proof 6.4 I show that in the following cases where a link is added in response to a ROUTE-REPLY, no loop occurs: 1) if the node has not sent a ROUTE-REPLY since it
last lost all its routes; and 2) if the $maxDist$ of the neighbor to the receiver is less than or equal to the $minimaxDist$ of this node at the time that the Route-Reply is received from the neighbor.

In the first case, a node changes its UID after losing routes. So, every neighboring node with a link to this node possesses a different UID for that link regardless of whether the original Route-Broken packet is dropped or not. So, according to lemma 6.1, loop is not created. For the second case, if adding the link creates a loop, then all the link UIDs along this loop should be valid (lemma 6.1). Let’s assume that adding a link $A \rightarrow B$ in the second case leads to a loop. Because of the loop, a path should also exist from $B$ back to $A$ which can be written in the form: \( \{X_1, X_2, \ldots, X_n\} \) where $X_1 = B$ and $X_n = A$. According to lemma 6.2, for any link $X_i \rightarrow X_{i+1}$ in the network graph, we must have:

$$minimaxDist(X_i) \geq minimaxDist(X_{i+1})$$

According to lemma 6.2, the $maxDist$ of $X_1$ can be related to $minimaxDist$ of $X_n$ in the following way:

$$maxDist(B) = maxDist(X_1) \geq 1 + minimaxDist(X_2)$$
$$\geq \ldots \geq 1 + minimaxDist(X_n) = 1 + minimaxDist(A)$$

$$\Rightarrow maxDistB \geq 1 + minimaxDistA$$

Thus, $maxDist(A) \geq 1 + maxDist(B) \geq 2 + minimaxDist(A)$, which is a contradiction since the $maxDist$ metric cannot grow to two hops larger than the $minimaxDist$. Thus, the theorem holds.
6.5 Distributed Optimization

Packet forwarding at each node in LAMP is based only on local information, using a local approximation of the globally optimal multicast tree. Due to this approximation, the routes selected may not always result in the globally optimal paths, when the routes through different selected neighbors meet at later intermediate nodes. To make the resulting paths closer to optimal, LAMP applies a distributed optimization algorithm, which we call *anticipatory forwarding*, at each node, also relying only on local information. With anticipatory forwarding, an intermediate node may forward a particular multicast data packet toward receivers for which it anticipates later receiving a request to do so.

A simple example is shown in Figure 6.4. Here, source node $S$ sends a multicast data packet to a group consisting of four receiver nodes, $R_1$ through $R_4$. Node $S$ delegates receivers $R_1$ and $R_2$ to neighbor $A$ in one transmission of the packet and $R_3$ and $R_4$ to neighbor $D$ in another transmission. In particular, since the distances to each of $R_2$ and $R_3$ through $A$ and through $D$ are equal, $S$ in general divides such receivers between the neighbors with equal distances in order to avoid concentrating the forwarding load on only one part of the network, as $S$ cannot know based on only local information that the two paths here subsequently merge (at node $M$), rather than the more likely case of the paths remaining separate.

Suppose $M$ first receives the packet from $B$, delegating $R_2$ to $M$, and thus forwards the packet to $N$. When $M$ then receives the packet from $E$, delegating $R_3$ to $M$, $M$ will forward the packet again, along the same link, also to $N$, resulting in a loss of total forwarding efficiency. *Anticipatory forwarding* helps reduce this inefficiency. Nodes remember which receivers they were delegated, at the last forwarding (here, $M$ remembers both $R_2$ and $R_3$). On the next packet from $S$ for the same group,
Figure 6.4  Anticipatory forwarding of overheard packets

\( M \) forwards the packet to \( N \) only \textit{once}, immediately upon receiving the packet the first time (regardless of whether it received from \( B \) or \( E \)). \( M \) delegates the full list of receivers remembered from the previous forwarding to \( N(\{R_2, R_3\}) \). Since the intermediate nodes used in forwarding of multicast packets to the same group remain the same until some relevant link cost changes such as due to the node motion, intermediate nodes are generally responsible for the same set of receivers from one packet to the next. As this optimization is applied at each forwarding node, the savings in total forwarding transmissions can be significant.

Anticipatory forwarding not only improves the protocol’s the data packet overhead but also the latency of packet delivery. It is applied upon \textit{receiving} a multicast data packet (i.e., addressed to this node for forwarding) and upon \textit{overhearing} a multicast packet (e.g., using promiscuous receive mode). Each node remembers the last multicast transmission state to different multicast groups. This state includes the set of receivers that the node was responsible for last time, referred to by \textit{old set}. 
When later this node receives or overhears a new packet for the same multicast group, it aggregates the set of receivers in the old set with the possible new set of receivers delegated to it on this packet, referred to by new set (for an overheard packet, the new set is empty); for any subsequent receipt of the same data packet by this node, it forwards the packet again only if it includes additional delegated receivers for which it has not already forwarded. The complete set of receivers for this packet becomes the old set for the next new multicast data packet.

6.6 Evaluation Methodology

To evaluate the performance of LAMP, I conducted detailed ns-2 simulations of the protocol across a range of ad hoc network scenarios, generated by RCS and RWP, and compared its performance to that of the existing ODMRP [23] and ADMR [17] multicast routing protocols. ODMRP is a reasonably simple protocol that periodically rebuilds multicast routing state from each active sender to deal with effects such as node mobility, whereas ADMR is a more complex, adaptive protocol that attempts to reduce overhead and improve performance through techniques such as local routing state repair and adaptation to a sender’s traffic pattern. The nominal wireless transmission range of nodes (depending on possible packet collisions and the capture effect) in our simulations was 250 m, with a physical data rate of 2 Mb/s; all nodes used the IEEE 802.11 DCF MAC protocol.

I used two different types of ad hoc network scenarios from each one of RCS and RWP, in the evaluation: scenarios of 50 mobile nodes with density control parameters of 0.1 (sparse) and 1 (dense) from RCS, as well as scenarios generated by RWP, with mobile nodes moving about within areas of $1100 \times 1100$ m$^2$ and $670 \times 670$ m$^2$. The
latter two are the sparse and dense type of usable scenarios that can be generated by RWP, and are the same sizes used for evaluating DSDV performance in Section 5.1.2.

For each, the node movement speeds was chosen ranging from 0 (stationary networks) to 20 m/s (highly mobile network). In all scenarios the following traffic pattern was simulated: 1 multicast sender node generates multicast packets of 64-bytes at rate of 4 packets/s, sending the packets to a single group consisting of 15 receiver group member nodes.

I consider the following three metrics in the evaluation: Packet delivery ratio (PDR) is the total number of data packets received divided by the number expected, counting each group member at the time each data packet is originated. Delivery latency is the average time between a packet being transmitted and when that packet was received by a multicast group member, with each member counted individually. Total routing packets is the number of non-data packets transmitted during a simulation, reflecting how efficient the underlying routing protocol performs to maintain routes between senders and receivers of a group.

6.7 Simulation Results

The simulation results are shown in Figures 6.5 to 6.7. Each data point in the graphs represents the average of 10 randomly generated scenarios, with the same scenarios used for all three of the protocols. Each graph is labeled with the type of scenario (sparse and dense scenarios generated by RCS as well as usable sparse and dense scenarios generated by RWP) For each metric, the y-axis is plotted in the same scale across all four types of scenarios for easier comparison across mobility models and protocol performance.
Figure 6.5 shows the packet delivery ratio (PDR) results for the three protocols. LAMP is able to maintain better PDR across all types of scenarios, except the very highly mobile dense network generated by RCS at 20 m/s. In sparse scenarios, this performance improvement of LAMP becomes much more significant. For dense scenarios, LAMP somewhat outperformed ADMR and essentially equaled ODMRP;

In the sparse scenarios generated by RCS, shown in Figure 6.5b, ODMRP and ADMR both suffer a significant drop in PDR. At speed of 20 m/s, ADMR delivers fewer than 40% of the data packets in these scenarios, while ODMRP delivers only about 65%. This unique sparse behavior of these multicast protocols are not very noticeable in sparse scenarios generated by RWP (Figure 6.5d); these results also demonstrate the limited use of RWP in generating diverse type of scenarios.

These significant PDR losses for these two protocols highlight how broadcast-based multicast protocols, such as ODMRP and ADMR, can degrade dramatically in sparse scenarios. This is mainly caused by the unreliability of broadcast transmissions and the higher rate of topology changes in sparse networks. Nodes in sparse scenarios have only a small number of neighbors. Thus, transmission failures can result in more significant losses in the PDR since there is much less opportunity for redundant forwarding as opposed to in dense scenarios. However, the use of unicast data packet transmissions in LAMP makes the protocol reliable and resilient to such packet losses.

Another reason for the low PDR in these protocols is the lack of a recovery mechanism in case of packet losses. ODMRP, for instance, depends on the periodic refresh interval for JOIN QUERY floods to maintain an up-to-date forwarding mesh. Any packet losses due to the out-of-date multicast state cannot be detected nor recovered by ODMRP. In case of ADMR, due to the on-demand nature of the protocol, nodes wait for some multiple of the inter-packet time for a multicast packet to be received
Figure 6.5 Packet Delivery Ratio for multicast protocols for sparse and dense scenarios generated by RCS
Figure 6.5 Packet Delivery Ratio for multicast protocols for sparse and dense scenarios generated by RWP (different area sizes)

(c) RWP: Dense (670 m × 670 m area)

(d) RWP: Sparse (1100 m × 1100 m area)
before they trigger an attempt to repair the disconnection. All packets lost until next repair cannot be recovered and delivered to the multicast group later, since the nodes do not know whether the tree is still connected during those failures. However, when route failures happen in LAMP, packets are buffered until either a route is obtained or the timeout value is reached; buffered packets are recovered as soon as a route is obtained. Thus, LAMP is able to maintain a very high PDR even at higher loads and highly mobile networks, both in dense as well as in sparse scenarios.

Figure 6.6 depicts the delivery latency results for the three protocols. LAMP maintains the lowest delivery latency among all protocols in all scenarios except for cases that are highly mobile (speed of 20 m/s) in dense scenarios, where it is bested by 10 milliseconds (Figure 6.6a). In general, both LAMP and ADMR have increasing latencies as the speed increases (as the network becomes more mobile), since the rate of topology change increases, and the routing protocol generates more routing packets; thus, there is more competing load on the network. ODMRP, on the other hand, does not generally exhibit such a decreasing trend in latency, as its routing in maintaining the forwarding mesh is strictly periodic in nature; ODMRP is only on-demand in the sense of changing its action as new senders become active or are no longer active.

Despite using a separate unicast transmission for each next neighbor, LAMP is able to maintain better latency than the other two protocols in most cases. One reason for this low latency in LAMP is that, because of the unicast transmissions, LAMP does not need to use jitter before transmitting each data packet; however, broadcast jitter delay is required in protocols relying on flooding mechanism in order to help prevent collisions between the transmitted broadcast packets [8]. A second reason is that LAMP utilizes more efficient routes for forwarding multicast packets
Figure 6.6  Packet Delivery Latency for multicast protocols for sparse and dense scenarios generated by RCS
Figure 6.6  Packet Delivery Latency for multicast protocols for sparse and dense scenarios generated by RWP
by the local approximation algorithm to globally optimal forwarding (Section 6.3), which in turn reduces the average delivery latency for each data packet.

Figure 6.7 shows the performance for total routing packets, the number of non-data packets transmitted during each simulation. In all scenarios, LAMP maintains substantially lower overhead than ADMR and ODMRP except for highly mobile sparse scenarios at 20 m/s (Figure 6.7b). The higher routing packets at these speeds is due to the higher packet delivery ratio of LAMP compared to the other protocols. ADMR and ODMRP essentially fail to recover packets, nor to maintain valid routes at these speeds in sparse networks, lowering their overall routing overhead at the expense of lower packet delivery ratio (as shown in the discussion of Figure 6.5b). Both LAMP and ADMR have generally increasing routing overhead as the speed increases (node motion on average increases), while ODMRP exhibits a nearly constant overhead.

These results also indicate the route-efficiency of LAMP in terms of its underlying multi-route unicast routing protocol (Section 6.5b), which maintains a lower overhead of keeping up-to-date routes for individual receivers, than do ADMR and ODMRP for a group of receivers, all together.

6.8 Chapter Summary

In this chapter, the power of RCS was demonstrated in the design and evaluation of a new multicast routing protocol, the Local-Approximation Multicast Protocol (LAMP). It was shown that the current multicast routing protocol, such as ADMR and ODMRP fail in sparse networks due to their reliance on unreliable broadcast and multicast transmissions. LAMP instead uses reliable unicast transmission for higher reliability, as well as utilizing local information to approximate the globally optimal multicast tree.
Figure 6.7  Total routing packets for multicast protocols for sparse and dense scenarios generated by RCS
Figure 6.7 Total routing packets for multicast protocols for sparse and dense scenarios generated by RWP.
LAMP was extensively evaluated using scenarios from the RCS model, and its performance results were compared to ADMR and ODMRP, two existing multicast protocols which previously were shown to perform well under the limited range of scenarios generated by random waypoint model. The results indicated that the density has a significant impact on the performance of these routing protocols, in which the performance significantly drops in sparse networks. LAMP, however, was able to maintain high packet delivery ratio and low latency in both sparse and dense networks. The understanding and evaluation of of LAMP and other protocols across such a range of types of wireless multihop network topologies, including dense as well as sparse topologies, has not previously been possible before the development of the Random Controlled Sparse (RCS) mobility model.
Chapter 7

Conclusion

In this thesis, I have presented the design and evaluation of the Random Controlled Sparse (RCS) mobility model that can generate connected scenarios with tunable node densities. Such tool has not previously existed, preventing researchers from evaluating mobile wireless network protocols under a wide range of scenarios. In particular, using this model, I have shown that sparse networks can show a very different behavior, that has not otherwise been demonstrated by existing mobility models.

The RCS model is implemented internally as a discrete event simulator, utilizing highly efficient graph and computational geometry algorithms to generate connected scenarios of desired density levels. Expensive components of the scenario generator has also been parallelized in order to speed up the generation of scenarios of different node sparsity and density; the resulting efficiency of the model’s execution makes it practical to use in even large simulation studies.

I have shown the effectiveness of the model by studying the performance of an example unicast routing protocol (DSDV). The protocol suffered significantly worse performance in sparse networks. This performance loss suggests new opportunities for improved protocol designs and the need for further study in the wide range of different node densities enabled by the new RCS mobility model.

To further show the application of the model in protocol design and evaluation, I then presented the design and evaluation of the Local-Approximation Multicast Proto-
col (LAMP), a new on-demand multicast routing protocol for mobile wireless ad hoc networks, the evaluation of which was made possible through the use of RCS for generating scenarios of varied degrees of density levels.

In particular, I showed that many of the prior multicasting work suffer significantly in terms of their performance, particularly in scenarios with sparse node density due to their reliance on unreliable link-layer broadcast or multicast transmissions. LAMP, in contrast, is a unicast-based multicast protocol, which works by approximating the globally optimal multicast tree at each intermediate forwarding node using only local information. To improve routes in cases in which this approximation deviates from the globally optimal forwarding, LAMP also introduces a new distributed optimization known as anticipatory forwarding, improving both the overhead and the delivery latency of the protocol. I have evaluated LAMP through detailed ns-2 simulations using scenarios from the RCS model as well as the random waypoint model, and compared it with ODMRP and ADMR, two existing protocols shown to perform well in the previous studies. LAMP generally equaled or outperformed both protocols for dense as well as sparse scenarios on the performance metrics.
Bibliography


