

**AN EFFICIENT APPROACH TO DISTRIBUTED
INFORMATION DISSEMINATION IN
MOBILE AD HOC NETWORKS**

by
Nicholas B. Bauer

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematical and Computer Sciences).

Golden, Colorado

Date _____

Signed: _____
Nicholas B. Bauer

Approved: _____
Dr. Tracy Camp
Associate Professor

Approved: _____
Dr. Michael Colagrosso
Assistant Professor

Golden, Colorado

Date _____

Dr. Graeme Fairweather
Professor and Head
Department of Mathematical and
Computer Sciences

ABSTRACT

In order to ease the challenging task of information dissemination in a MANET, we employ a legend: a data structure that is passed around a network to share information with all the nodes. Our motivating application of the legend is sharing location information. Previous research shows that a location service using a legend performs better than other location services in the literature. To improve the legend-based location service, we develop and evaluate three methods for the legend to traverse a network in this thesis and compare their performance in simulation. We also explore several improvements to the traversal methods, and describe our way of making the legend transmission reliable.

TABLE OF CONTENTS

ABSTRACT	iii
LIST OF FIGURES	vii
LIST OF TABLES	ix
ACKNOWLEDGEMENTS	x
Chapter 1 INTRODUCTION	1
1.1 All-to-All Broadcast	1
1.2 Related Work	2
1.3 Legend Traversal Overview	4
Chapter 2 TRAVERSAL METHODS	9
2.1 The LAR Method	9
2.2 The LRV Method	11
2.3 The TB Method	12
2.4 Traversal Comparison	16
Chapter 3 SIMULATION ENVIRONMENT	17
Chapter 4 TRAVERSAL IMPROVEMENTS	21
4.1 Improving the Traversal Methods	21
4.2 Simulation Results on the Traversal Improvements	22

Chapter 5	COMPARING THE TRAVERSAL METHODS	27
Chapter 6	LEGEND RELIABILITY	35
6.1	Providing Reliable Traversal	35
6.2	Preventing Multiple Simultaneous Legends	36
6.3	Minimizing the Duration of Multiple Simultaneous Legends	37
6.4	Simulation Results on Legend Reliability	41
Chapter 7	OPTIMIZING THE LRV METHOD	45
7.1	Legend Pause Interval	45
7.2	LRV Update vs. Visit	45
7.3	LRV Visit Approximation	48
7.4	Simulation Results on LRV Update vs. Visit	49
Chapter 8	CONCLUSIONS	53
	REFERENCES	55
Appendix A	TRAVERSAL METHOD PSEUDOCODE	59
A.1	Common Pseudocode	60
A.2	LAR Pseudocode	64
A.3	LRV Pseudocode	67
A.4	TB Pseudocode	70
Appendix B	CORRECTNESS PROOF FOR THE LRV TRAVERSAL METHOD	73

LIST OF FIGURES

1.1	An Example Legend Visit	5
1.2	A More Detailed Example Legend Visit	7
2.1	An Example LAR Traversal	10
2.2	An Example LRV Traversal	13
2.3	An Example TB Traversal	15
4.1	Performance and Overhead Plots for the Traversal Improvements	23
4.2	Performance and Overhead Plots for the LRV Improvements	25
5.1	Initial Plots	28
5.2	Performance vs. Overhead (2 m/s)	29
5.3	Performance vs. Overhead (5 m/s)	29
5.4	Performance vs. Overhead (10 m/s)	30
5.5	Performance vs. Overhead (15 m/s)	30
5.6	Performance vs. Overhead (20 m/s)	31
5.7	Summary Plots	33
6.1	Legend Reliability - Performance vs. Overhead (2 m/s)	38
6.2	Legend Reliability - Performance vs. Overhead (5 m/s)	39

6.3	Legend Reliability - Performance vs. Overhead (10 m/s)	39
6.4	Legend Reliability - Performance vs. Overhead (15 m/s)	40
6.5	Legend Reliability - Performance vs. Overhead (20 m/s)	40
6.6	Realistic LRV Method - Performance vs. Overhead (2 m/s)	42
6.7	Realistic LRV Method - Performance vs. Overhead (5 m/s)	42
6.8	Realistic LRV Method - Performance vs. Overhead (10 m/s)	43
6.9	Realistic LRV Method - Performance vs. Overhead (15 m/s)	43
6.10	Realistic LRV Method - Performance vs. Overhead (20 m/s)	44
7.1	Pause Interval - Performance vs. Overhead (2 m/s)	46
7.2	Pause Interval - Performance vs. Overhead (5 m/s)	46
7.3	Pause Interval - Performance vs. Overhead (10 m/s)	47
7.4	Pause Interval - Performance vs. Overhead (15 m/s)	47
7.5	Pause Interval - Performance vs. Overhead (20 m/s)	48
7.6	Update/Visit - Performance vs. Overhead (2 m/s)	50
7.7	Update/Visit - Performance vs. Overhead (5 m/s)	50
7.8	Update/Visit - Performance vs. Overhead (10 m/s)	51
7.9	Update/Visit - Performance vs. Overhead (15 m/s)	51
7.10	Update/Visit - Performance vs. Overhead (20 m/s)	52

LIST OF TABLES

3.1	Simulation Details	20
6.1	Reliability Statistics	38

ACKNOWLEDGEMENTS

Thanks to my thesis advisors, Dr. Tracy Camp and Dr. Michael Colagrosso, and to my thesis committee members, Dr. William Navidi and Dr. Dinesh Mehta, for their help and support. This work was partially supported by National Science Foundation grant ANI-0240588. Research group's URL is <http://toilers.mines.edu>. We thank Charlie Colburn for the inspiration to develop the TB method.

Chapter 1

INTRODUCTION

1.1 All-to-All Broadcast

A Mobile Ad hoc Network (MANET) is a network of mobile nodes communicating without relying on any static network infrastructure. One of the most expensive operations in a MANET is the all-to-all broadcast, in which every node has information to communicate to every other node in the network. In order to accomplish this operation efficiently, we create a data structure to be passed around the network, gathering and sharing information with each node it visits. This structure is called a legend.

There are many useful applications of an all-to-all broadcast. One example application is disseminating network topology information. Having up-to-date information about the network topology can ease the task of routing. Although a node can gather information about its local topology, sharing that information with the other nodes in the network can be a daunting task. With an efficient mechanism for performing all-to-all broadcast, this task becomes easier. The legend is not limited to all-to-all broadcasting. It is also useful for doing one-to-all broadcasting, also called cheap advertisement.

While there are several applications for a legend, in this thesis we focus on the sharing of location information. Choosing to share location information has a “rebound effect” because it makes the legend traversal easier, which in turn makes the sharing of location information easier. Previous research shows that the legend idea performs well compared to other location services [1]. For example, the Legend Exchange and Augmentation Protocol (LEAP) outperforms the Simple Location Service (SLS) [2], the Reactive Location Service

(RLS) [2], and the Grid Location Service (GLS) [3] at every speed studied in [1]. According to the simulation results presented in [1], LEAP consistently showed the smallest location error, delay, and overhead of all four location services studied. While the legend shows excellent performance in this comparison, its algorithm for traversing MANETs was not carefully studied. Indeed, the traversal method used in [1] is an unrefined, idealized version of the Location Aided Routing method discussed in Chapter 2.1. Our proposed algorithms outperform LEAP [1].

Because the legend showed such high performance in the location service application, and shows high potential for many other possible broadcast applications, in this thesis we explore ways for the legend to traverse a network. We develop two new methods for a legend to traverse a MANET, and then run simulations to compare their performance against the traversal method proposed in [1]. The rest of the thesis is organized as follows: Chapter 1.2 describes related work. Chapter 1.3 introduces the common features that are shared by all three traversal algorithms. Then, in Chapter 2, we describe the actual traversal methods. Our simulation environment is described in Chapter 3. In Chapter 4, we describe a few improvements we made on the traversal methods, and Chapter 5 gives the results of our simulations. Chapter 6 details how the legend traversal was made reliable, and Chapter 7 describes how the traversal method found to be most efficient was optimized. Chapter 8 provides our conclusions on the traversal method comparison, as well as our proposals for future research on legends in MANETs.

1.2 Related Work

The all-to-all broadcast operation is relatively unexplored, possibly because it is so expensive in MANETs. The effect of all-to-all broadcasting on energy efficiency in a MANET is studied in [4]. However, that study assumes that all nodes are directly connected. To our

knowledge, the more complicated problem of all-to-all broadcasting in MANETs larger than a single node's transmission range requires further study.

One application of all-to-all broadcasting is sharing location information. Many routing protocols proposed for MANETs benefit from having accurate, up-to-date information about the locations of other nodes in the network. The Location Aided Routing (LAR) protocol [5], for example, uses location information to reduce the overhead of finding routes. Other protocols, such as the Depth First Search (DFS) protocol [6], use location information to determine routes directly. [7] and [8] provide surveys of MANET routing protocols that use location information.

The legend *traversal* problem is similar to both the Traveling Salesman Problem (TSP) and the Minimum Spanning Tree (MST) problem [9]. While there are several known algorithms to solve these problems, they work only for static networks. Also, some rely on global knowledge to solve the problem; the legend is limited to its own local knowledge. Thus, new methods are needed to solve the legend traversal problem.

The algorithm in [10] takes a different approach to the TSP. Its ant colony system uses ant-like agents to find a distributed solution to the TSP. However, they also use global knowledge to solve the problem. The GPSAL algorithm proposed in [11] uses ant-like agents that resemble a legend in some ways. However, these agents are unicast from one node to another, instead of traversing the entire network like a legend.

To the best of our knowledge, the legend traversal problem for a mobile ad hoc network has been unexplored. In order to find an efficient way for the legend to migrate throughout a MANET, we design our own traversal methods. In order for the legend to be a useful tool for all-to-all broadcasting in MANETs, it has to move about the network in an efficient manner. Any improvement in efficiency due to our methods can be immediately realized by applications using all-to-all broadcast, e.g., routing.

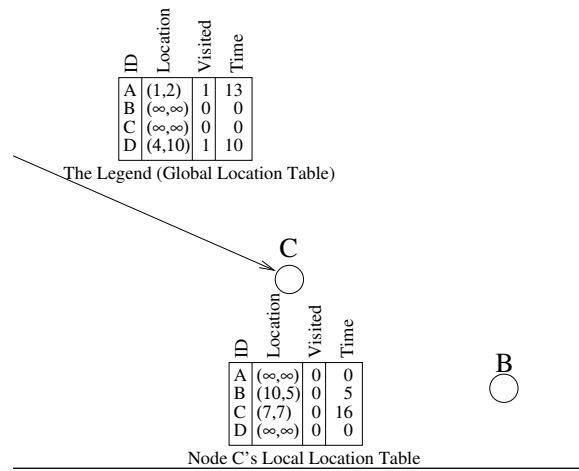
1.3 Legend Traversal Overview

In this chapter we begin to discuss the legend traversal methods. Before looking at the specific traversal methods, we describe the details that are common to all three. The parenthetical line numbers appearing in this and later chapters refer to lines of pseudocode that can be found in Appendix A.

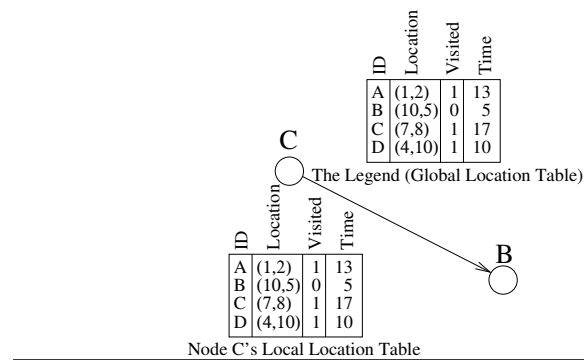
Following the work in [1], our application of the legend is sharing location information. Therefore, in all three traversal methods, the legend is a location table with an entry for each node. Each entry contains a location, a visited bit that stores whether the legend has visited the node, and a timestamp of when the node was last visited by the legend. In addition, nodes maintain a local location table with the same fields. When a node receives the legend, both the legend and the node's local location table are updated based on the most recent timestamp for each entry in the two tables (lines 41–50). Also, the entry for the node itself is updated with the node's current location information, the current time, and the visited bit is set (lines 36–40). After this update procedure finishes, the legend is sent to another node following its traversal method. Thus, at the time of transmission, the most recently visited node and the legend have identical location information for all visited nodes.

Figure 1.1 shows an example legend visit. Figure 1.1(a) shows the legend's global location table and node C 's local location table just *before* the legend arrives at node C . Figure 1.1(b) shows the same tables just *after* the legend leaves node C . Notice that both the legend and node C 's local location table are identical when the legend traverses to the next node. Note also that node C 's entry in both tables has been updated.

In each traversal method, nodes have knowledge of who their neighbors are. A node keeps track of its neighbors in a neighbor table. A node B is considered to be a neighbor of another node C if and only if B is within C 's transmission range. In order to fill in its



(a) The legend in flight to Node C at time $t = 16$. The legend has not yet visited C , so its visited and time entries are zero in the legend.



(b) The legend leaves Node C at time $t = 17$. The legend and node C have updated each other; both tables have identical, up-to-date information.

FIG. 1.1. An Example Legend Visit; C 's location table is updated with A and D 's information and the legend is updated with B and C 's information.

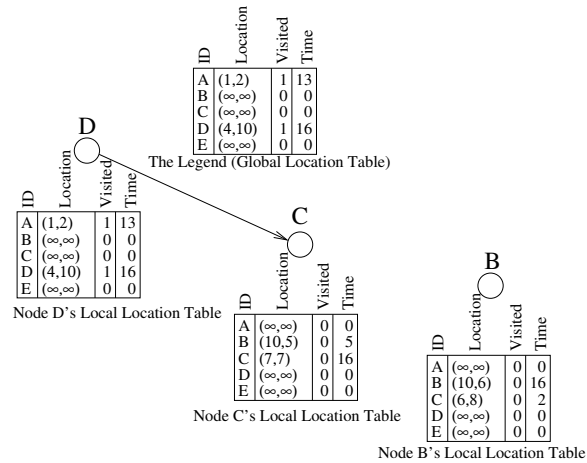
neighbor table, a node periodically shares its own location with its neighbors via “hello” packets. When a node receives a “hello” packet from another node, it adds that node’s ID to its neighbor table (lines 31–34). The location information from the “hello” packet is used to update the node’s local location table (lines 27–30).

Notice in Figure 1.1(a), node C received a “hello” packet from node B at time $t = 5$. Because this “hello” packet contained B ’s location, node C ’s local location table contains that location. In Figure 1.1(b), after visiting node C , the legend has been updated with B ’s location at time $t = 5$ also.

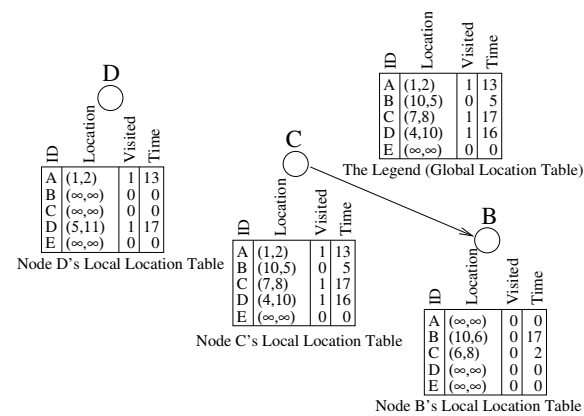
Figure 1.2 shows a more detailed example of a legend visit. Notice in Figure 1.2(b) that node D ’s local table changes as node D moves after sending the legend to node C , but the legend does not reflect those changes. Also notice that since the legend has not visited node E , or any nodes that have heard a “hello” packet from node E , node E ’s entry in the legend and all the nodes’ tables remain zero.

While maintaining neighborhood information adds overhead to the traversal methods, the “hello” packets are very small in size and are transmitted locally. Also, in a network with substantial data traffic, neighbor information could be obtained via promiscuous listening or could be available from the MAC layer. In our simulations, because our focus is on comparing the traversal methods themselves, we chose not to send any data traffic and not to gain neighbor knowledge through a lower layer protocol. Therefore, we send “hello” packets for nodes to find their neighbor information. All three traversal methods send the same number of “hello” packets. Potentially the overhead for obtaining neighbor information could be reduced.

For all the traversal methods, the legend must be created by a node. We choose the node with the lowest ID for this task. This node initializes the legend by setting all visited bits to false and the current location to undefined for every entry in the table (lines 1–6).



(a) The legend in flight from Node *D* to Node *C* at time $t = 16$.



(b) The legend leaves Node *C* at time $t = 17$. Node *D* has moved since time $t = 16$, so its entry in its own table has changed. The legend has not yet visited any node with information about Node *E*, so its visited and time entries are zero in the legend.

FIG. 1.2. A More Detailed Example Legend Visit; *C*'s location table and the legend are updated by joining their records.

It then sets its own visited bit and updates its own current location information and the locations of its known neighbors, and then sends the legend on to the next node following its traversal method.

Chapter 6 describes how the traversal methods ensure reliable legend traversal in an unreliable network. The next chapter describes the different traversal methods in detail.

Chapter 2

TRAVERSAL METHODS

The purpose of this thesis is to find the best way for a legend to traverse a MANET. We developed three different traversal algorithms for the legend, and then ran simulations to compare their performance. In the following chapter we describe the different algorithms in detail.

2.1 The LAR Method

We call the first traversal method the Location Aided Routing (LAR) method, because the method uses the LAR protocol [5]. The LAR method is based on the idea of always sending the legend to the closest unvisited node. This method is detailed in [1]; we summarize its functionality herein.

Using the LAR method, a node that has the legend sends it to its *closest neighbor* that has not been visited by the legend (i.e., whose visited bit is not set in the legend table) (line 113). If no such neighbor exists, then the legend is sent via LAR to the *closest node* that has not been visited by the legend and that has been tried fewer than three times (i.e., whose number of attempts is less than three in the table) (lines 114–115). Thus, in addition to storing location information, visited bits, and timestamps for every node in the network, the legend also stores a number of attempts for each node. Every time a node tries to send the legend to another node, either directly or via the LAR protocol, it increments the number of attempts in the destination node's entry in the legend's global location table (line 102).

If the destination node is a neighbor then the source node transmits the legend directly

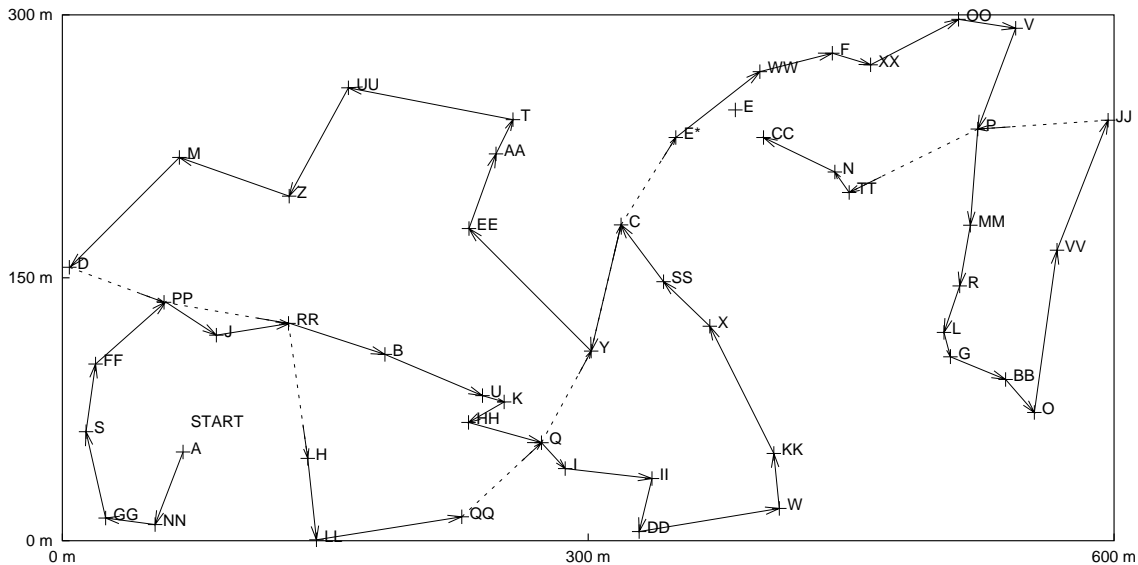


FIG. 2.1. An Example LAR Traversal. Solid lines represent direct transmission, and dashed lines show the legend being sent via LAR. When two arrows lead from a node, the legend always migrates along the solid arrow first. The legend moves from *A* to *D* on solid lines, then from *D* to *H* via LAR. It then moves from *H* to *QQ* on solid lines, and then from *QQ* to *E* via LAR. The legend then moves from *E* to *JJ* on solid lines, and from *JJ* to *TT* via LAR. Finally the legend moves from *TT* to *CC* on solid lines.

to the destination node (lines 103-106). Otherwise, the source node sends the legend to the destination node using the LAR protocol [5] (lines 107-112). Once all the nodes in the network have been visited, or tried at least three times, then the node with the legend clears all the visited bits and number of tries in the legend table, pauses the legend for a short period, and then begins the traversal again.

In the case of a partitioned network, it is possible that the legend does not have knowledge of some of the nodes' locations. In this case, the legend is sent to the node with the lowest node ID that has not been tried at least three times.

Figure 2.1 shows an example of how the legend traverses a network using the LAR method. This figure is not a snapshot of the network at a specific time, but rather a record of

the position of each node when it is visited by the legend. The dashed lines show how the legend is sent using the LAR routing protocol. The solid lines show direct transmissions of the legend. The legend begins at node A and traverses until it visits every node at least once. Note that the LAR protocol is used to forward the legend three times in this example. The case where node QQ uses the LAR protocol to send the legend to node E is different than the other two cases. This difference is due to the fact that the network is partitioned the first time node QQ tries to send the legend to node E . Thus the first LAR attempt fails and the legend is forced to wait for a timeout before being resent (see Chapter 6 for a discussion of timeouts). During this timeout, node E moves closer to node C and reconnects the network. After the timeout, node E is picked as the next destination again, and this time the LAR attempt succeeds and node E is able to receive the legend at the E^* position in Figure 2.1.

2.2 The LRV Method

The Least Recently Visited (LRV) method is a greedy algorithm in which the legend's next destination is the neighbor with the oldest timestamp in the legend. Because the goal of the legend is to keep all the node's information up-to-date, the node that most needs to be visited is the node that hasn't been visited in the longest time.

Using this method, a node that has the legend sends it to its closest neighbor that has *not been visited* by the legend (i.e., whose visited bit is not set in the legend table) (lines 145–146). If all of its neighbors have been visited, the source node sends the legend to the neighbor which has been visited by the legend less recently than any other neighbor (i.e., the neighbor with the lowest timestamp in the legend table) (lines 147–154). Because the destination node is always a neighbor, the source node transmits the legend directly to the destination node. After traveling a certain number of hops (we use 25 hops in a network

of 50 nodes) since the last pause, the node with the legend pauses the legend for a short period and then begins the traversal again (lines 132–134, 156–160). Note that using this method, visited bits are never reset. (See Section 7.1 for our exploration of how often the legend should pause.)

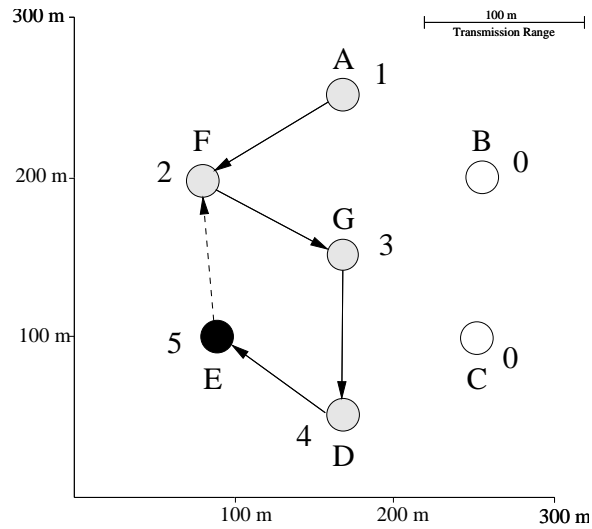
Figure 2.2 shows an example of how the LRV traversal method works. The letters in the figures represent node IDs, while the numbers represent the time that each node was last visited. Grey nodes are those that have been visited, and white nodes are those that have not been visited. The black node currently has the legend. The solid lines show where the legend has been, and the dashed line shows the legend’s next destination. In Figure 2.2(a), the legend began at node *A* and moved to *A*’s closest unvisited neighbor, node *F*. It then repeated this procedure to node *G* and then to node *D* and finally to node *E*. Notice that the current node (node *E*) has no unvisited neighbors (*D*, *F*, and *G*), and so chooses to send the legend to its *least recently visited neighbor*, node *F*.

Figure 2.2(b) shows the same network as Figure 2.2(a), four hops later. From node *E*, the legend moved back to node *F*, and then followed the same logic back to node *A*. Node *A* then sent the legend to its closest unvisited neighbor, node *B*, and from there the legend was sent to node *C*. Notice that, once the legend reaches node *C*, all of the nodes have been visited. Thus, from this point forward, the legend will always move to the current node’s *least recently visited neighbor*.

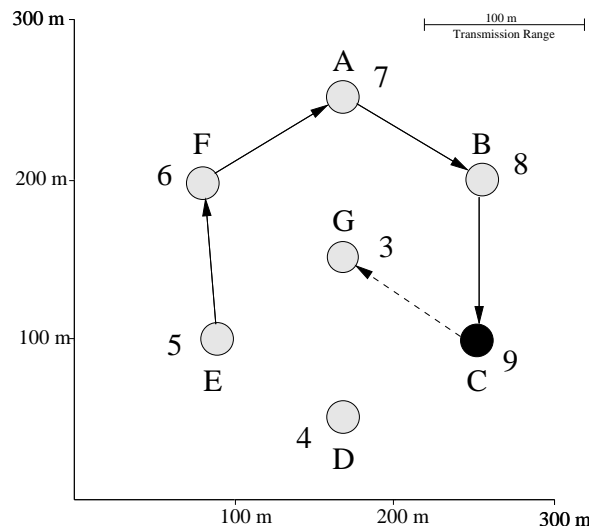
2.3 The TB Method

The Trace Back (TB) method is based on doing a Depth First Search (DFS) traversal. The idea is that the legend visits every unvisited neighbor of a node before returning back to the previous node.

Using this method, a node keeps track of the ID of the node that sent it the legend



(a) Step 1: The legend traverses from $A \rightarrow F \rightarrow G \rightarrow D \rightarrow E$ (solid lines). E will send the legend to its Least Recently Visited neighbor, F (dashed line).



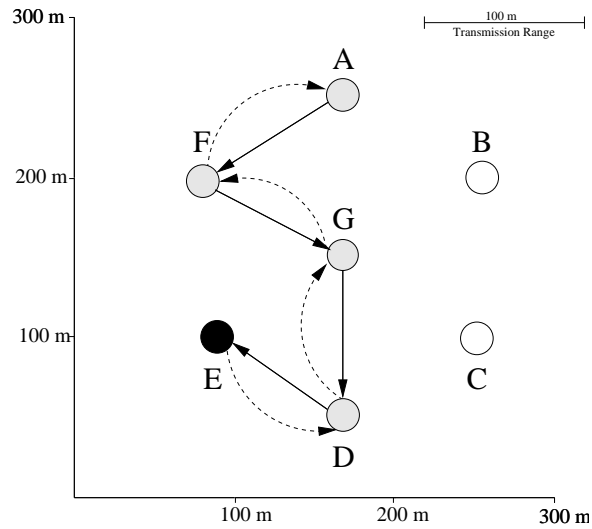
(b) Step 2: The legend moves from $E \rightarrow F \rightarrow A \rightarrow B \rightarrow C$ (solid lines). C will send the legend to its Least Recently Visited neighbor, G (dashed line).

FIG. 2.2. An Example LRV Traversal; LRV sends the legend to the Least Recently Visited neighbor

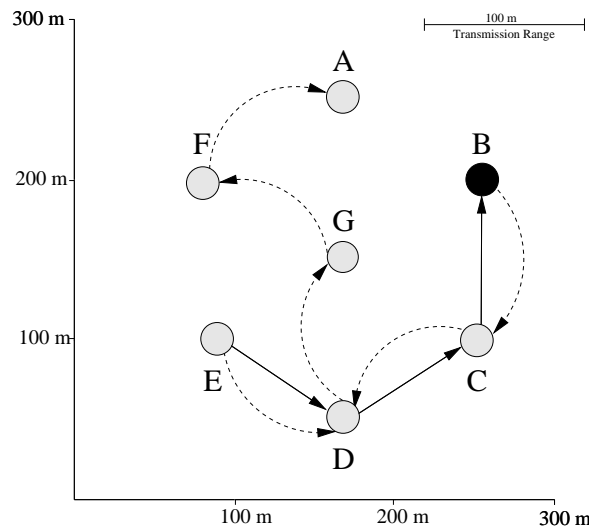
for the first time in a given loop (i.e., when it sets its visited bit in the legend table) as its traceback node (lines 168–169). A node with the legend sends it to the *closest unvisited neighbor* (lines 188–189). If no such neighbor exists, then the node sends the legend to its traceback node (lines 190–191). If its traceback node is no longer a neighbor, the node invalidates its own traceback node. If its traceback node is invalid, or if all of the nodes in the network have been visited, then the node clears all the visited bits in the legend table, invalidates its own traceback node (if necessary), pauses the legend for a short period, and then begins the traversal again (lines 173–177, 192–197, 199–201). Because the destination node is always a neighbor, the source node always transmits the legend directly to the destination node.

Figure 2.3 shows an example of how the TB traversal method works. In the figures, a dashed arced arrow leading from a node shows that node’s traceback link. As in Figure 2.2, a solid arrow shows where the legend has been. The letters indicate node IDs. The grey nodes have been visited by the legend, while the white nodes are unvisited. The node currently in possession of the legend is in black. In Figure 2.3(a), the legend began at node *A* and then moved to *A*’s closest unvisited neighbor, node *F*. Notice that node *F*’s traceback link leads to node *A* because *A* was the first node to send it the legend in the current loop. The legend then traversed to node *G*, and from there to node *D* and finally to node *E*. Notice that node *E* has no unvisited neighbors, so it chooses to send the legend back to its *traceback node*, node *D*.

Figure 2.3(b) is a follow-up to Figure 2.3(a), three hops later. Continuing from the previous figure, node *E* sent the legend back to its traceback neighbor, node *D*. Notice that even though node *D* received the legend from node *E*, its traceback link still points to node *G*. It doesn’t change that link because node *G* is still the first node to send node *D* the legend in the current loop. Because node *D*’s visited bit was set before it received the



(a) Step 1: The legend travels from $A \rightarrow F \rightarrow G \rightarrow D \rightarrow E$ (solid lines). The arced dashed lines are traceback links.



(b) Step 2: E sends the legend to D , following its traceback link. D sends the legend to $C \rightarrow B$.

FIG. 2.3. An Example TB Traversal; TB sends the legend along its backlinks when a node doesn't have an unvisited neighbor

legend from node E , it knows not to change its traceback link. From node D , the legend then moves to the closest unvisited neighbor, node C , and then again to node B . Once the legend reaches node B , all of the nodes in the network have been visited. At this point, the visited bits (and thus the traceback links) are all reset, and the traversal begins again.

Suppose that, in Figure 2.3(a), node D moves out of node E 's range after sending the legend to node E . Node E then tries to send the legend back along its traceback link to node D , as before; however, this time node D does not receive the legend. Once node E infers that its traceback node, node D , is no longer its neighbor, node E will invalidate its own traceback link, reset all the visited bits in the legend's global location table (thus resetting all the nodes' traceback links), and begin the traversal again.

2.4 Traversal Comparison

The three traversal methods we developed have some similarities. Using the LRV or the TB traversal methods, the current node will always send the legend to one of its neighbors. Only the LAR traversal method can pick a non-neighbor for the legend's next destination; the LAR protocol [5] is used to route legend packets to non-neighbors. Both the LAR and the TB methods keep track of a current loop. These two inherently require external knowledge of how many nodes exist in the network. The LRV method, on the other hand, can traverse a network repeatedly without resetting its algorithm. Because of this, it does not require knowledge of the number of nodes in the network, so long as it can add entries to the legend's global location table and the nodes' local location tables dynamically. In addition, all three traversal methods occasionally pause the legend. The LAR and TB methods pause the legend when it is logical to do so, i.e., when they are resetting their algorithms. The LRV method, on the other hand, has no logical reset point; therefore we chose to pause it every x (25) hops.

Chapter 3

SIMULATION ENVIRONMENT

We performed extensive simulations to compare the three traversal methods. All three of our traversal methods are implemented in the Network Simulator, NS-2 (version 2.1b7a) [12]. The simulator uses the IEEE 802.11 MAC sublayer. The performance of each method is tested in a network of 50 mobile nodes with a transmission range of 100 m in a 300 m x 600 m area. The nodes move according to the steady-state distribution for the Random Waypoint Mobility Model (see [13] for details) with the speed set to 2, 5, 10, 15, 20 m/s $\pm 10\%$ and pause time set to 10 s $\pm 10\%$. This mobility model initializes node positions and movements in such a way as to avoid the problems caused by the distribution of node movements changing over the simulation duration (see [14, 15] for details).

The nodes transmit “hello” packets once per “hello” packet interval. This interval is set to $10\text{m}/\bar{V}$, where \bar{V} is the average node speed (2, 5, 10, 15, 20 m/s). Thus the nodes transmit “hello” packets more frequently when the average node speed is higher, allowing them to have decent neighbor knowledge when the node mobility is high. The simulations begin with 100 s for nodes to share “hello” packets and for any initialization to stabilize. The legend begins traversing the network after 100 s and traverses for 400 s.

These simulation details are summarized in Table 3.1. Derived parameters are calculated from the input parameters [16]. Node density is simply the number of nodes divided by the simulation area. Coverage area is the area of the circle with radius equal to one node’s transmission range. A node’s transmission footprint is the percentage of the simulation area covered by a node’s coverage area. The maximum path length is the length of a diagonal of the simulation area. The network diameter is the maximum path length divided

by a node's transmission range. The network connectivity is based on the average number of neighbors. The value labeled "no edge effect" is calculated by dividing the coverage area by the node density. Accounting for the fact that nodes near the edges do not have neighbors on all sides of the node yields the value labeled "edge effect".

In these simulations, we measure performance in terms of average location error. We define location error as the absolute distance between where a node is predicted to be in another node's local location table and the node's actual position. If a node doesn't have any information about another node's location when the location error is measured, then the error used is half the diagonal of the simulation area, which is the maximum possible error if the node had guessed that the other node was located at the center of the simulation area. This case only occurs at the beginning of the simulations, before the legend visits all the nodes in the network. In the simulations, the legend is given one second to start traversing before the first error measurements are taken. From that point on, the location error is measured for each entry of every node's local location table, every five seconds.

We measure overhead in terms of legend packets transmitted. Legend packets include all packets transmitted that contain the legend's global location table, as well as all ACK packets and any LAR routing packets used to find routes to send the legend or ACKs in the LAR method. As we mentioned in Chapter 1.3, because we didn't have any data traffic in our simulations, nodes had to rely on "hello" packets to learn who their neighbors were. However, in a more realistic network with data traffic, it is possible for nodes to obtain neighbor knowledge by listening promiscuously to their neighbors' communication, or via a lower layer protocol. In this way, it is possible to reduce the overhead of the "hello" packets. Because all three traversal methods used exactly the same number of "hello" packets in all the simulations, and because we assume that these packets may not be necessary in a more realistic network (i.e., one carrying data), we chose to not include the "hello" packets

in our overhead measurements. We focus on packets rather than bytes transmitted because in a wireless environment, there is a great amount of overhead associated with transmitting a packet, regardless of its size. Therefore the number of packets transmitted gives a better representation of the amount of overhead than the number of bytes transmitted.

Input Parameters	
Number of Nodes	50
Simulation Area Size	300 m x 600 m
Transmission Range	100 m
Simulation Duration	500 s, legend traverses from 100-500 s
“Hello” Packet Interval	10m/Average Node Speed
Derived Parameters	
Node Density	1 node per 3,600 m ²
Coverage Area	31,416 m ²
Transmission Footprint	17.45%
Maximum Path Length	671 m
Network Diameter (max. hops)	6.71 hops
Network Connectivity (node degree)	8.73 (no edge effect)
Network Connectivity (node degree)	7.76 (edge effect)
Mobility Model	
Mobility Model	Random Waypoint [13]
Mobility Speed	2, 5, 10, 15, 20 m/s \pm 10%
Pause Time	10 s \pm 10%
Simulator	
Simulator Used	NS-2 (version 2.1b7a)
Medium Access Protocol	IEEE 802.11
Link Bandwidth	2 Mbps
Number of Trials	10
Confidence Interval	95%

Table 3.1. Simulation Details

Chapter 4

TRAVERSAL IMPROVEMENTS

4.1 Improving the Traversal Methods

In this section, we discuss several ideas to improve the basic legend traversal algorithms. These include:

promiscuous legend mode Using *promiscuous legend mode*, the nodes listen to overhear the legend being transmitted to another node and update their local location tables (lines 53-59).

overwrite legend mode Conventionally, a node updates the legend's global location table when it first receives the legend. Using *overwrite legend mode*, a node overwrites the legend's global location table with its own location table just before transmitting the legend. In this way the legend receives the most up-to-date information in case the node has received new "hello" packets or has moved significantly since receiving the legend, which may occur when the legend has been paused or when a previous legend transmission failed.

promiscuous neighbor table mode Traditionally nodes add entries to their neighbor table only when they receive "hello" packets. Using *promiscuous neighbor table mode*, if a node overhears any packet being transmitted, it adds the sender's ID of the transmitted packet to its neighbor table.

check neighbor mode Originally a node would remove an entry from its neighbor table only after a transmission to that neighbor failed. Using *check neighbor mode*, a node

removes entries in its neighbor table older than two “hello” packet intervals before choosing the next destination for the legend (lines 60–64).

We also improved upon the LAR traversal method presented in [1]. These two proposed improvements interact with the LAR protocol:

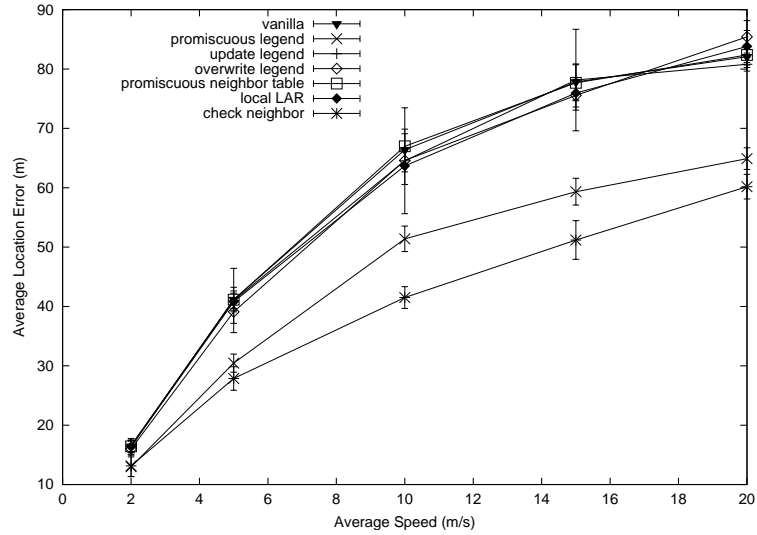
update legend mode Using *update legend mode*, intermediate nodes sending the legend along a LAR route update their own local location tables and the legend’s global location table as they send the legend.

local LAR mode The LAR protocol traditionally shares location information by piggy-backing it into every LAR packet. Nodes use this location information to reduce the overhead of flooding route requests by doing a directed flood. Using *local LAR mode*, nodes use the location information gathered from the legend to improve the LAR protocol.

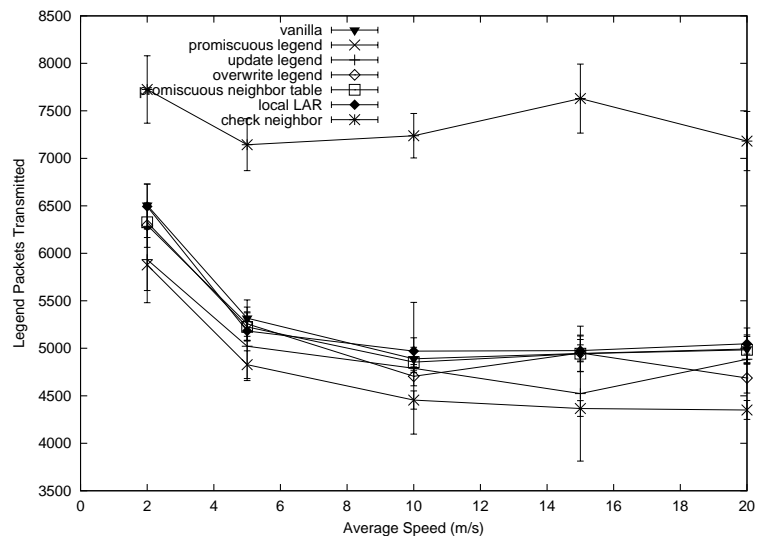
4.2 Simulation Results on the Traversal Improvements

To determine the effects of the proposed traversal method improvements, we simulated each of them and compared their performance. Figure 4.1 shows the results of our comparison for the LAR method. We show the LAR method because all of the proposed improvements can be applied to the LAR protocol. In Figure 4.1, the results labeled vanilla are the LAR protocol without any of the proposed improvements added.

During our performance investigations, we found the largest performance improvement occurred with the check neighbor mode, especially for the LRV and TB traversal methods. The LAR traversal method’s performance was not improved as much by the check neighbor mode because it does not always send the legend to a neighbor. Another major improvement was the promiscuous legend mode. The overwrite legend mode had



(a) Average Location Error vs. Average Node Speed



(b) Legend Packets Transmitted vs. Average Node Speed

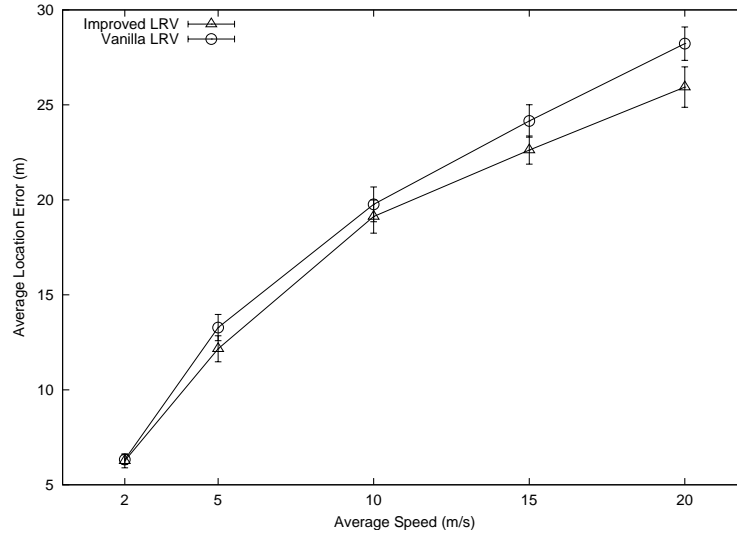
FIG. 4.1. Performance and Overhead Plots for the Traversal Improvements. Check neighbor mode and promiscuous legend mode help the most. The other proposed improvements do not help significantly.

only a slight improvement in performance. The local LAR mode and the update legend mode both showed minimal performance improvement to the LAR traversal method.

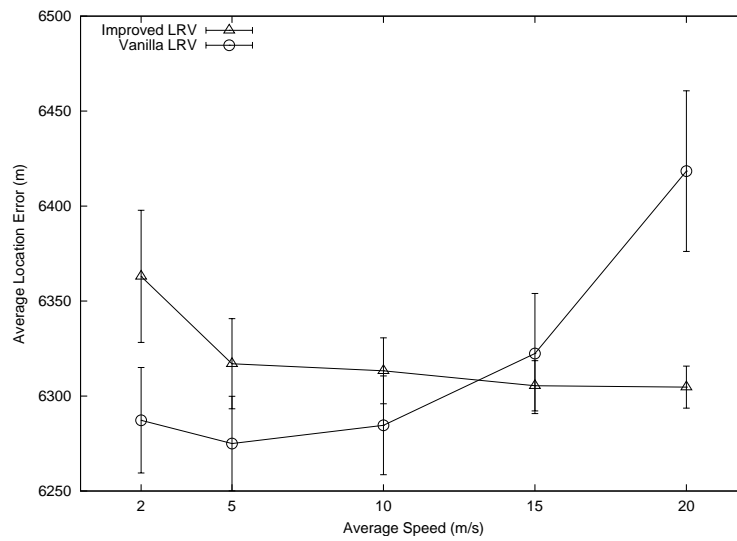
Promiscuous neighbor table mode actually decreased the performance of the traversals. We suspect that it hindered the legend's performance because nodes have entries in their neighbor table without an accurate location, i.e., a location gained via "hello" packets or the legend. This violates the assumption of the check neighbor improvement that neighbor tables are updated only via "hello" packets. Because of the unintended consequences, we chose not to include promiscuous neighbor table mode as part of the traversal methods.

We chose to apply all the traversal improvements that decreased the location error. This combination of traversal improvements was used for all three traversal methods in all of the simulation results shown in this thesis. This combination includes promiscuous legend mode, overwrite legend mode, and check neighbor mode for all three traversal methods, with the addition of update legend mode and local LAR mode for the LAR traversal method. Although operating in promiscuous mode generally yields high energy consumption, nodes already need to listen promiscuously to ensure the legend's reliability (see Chapter 6).

To verify that the proposed improvements help with the other traversal methods, we apply our combination of improvements to the LRV traversal method. We then compare this "improved" LRV with an unimproved LRV method in simulation. Figure 4.2 shows the results of this comparison. The results labeled "Improved LRV" include all the proposed improvements that decreased the location error of the LAR method, excluding those that only apply to the LAR protocol. The results for the LRV method without any of the proposed improvements is labeled "Vanilla LRV." We note that we only consider improvements that reduce the location error, regardless of the effect on packets transmitted.



(a) Average Location Error vs. Average Node Speed



(b) Legend Packets Transmitted vs. Average Node Speed

FIG. 4.2. Performance and Overhead Plots for the LRV Improvements. The proposed improvements decrease the location error.

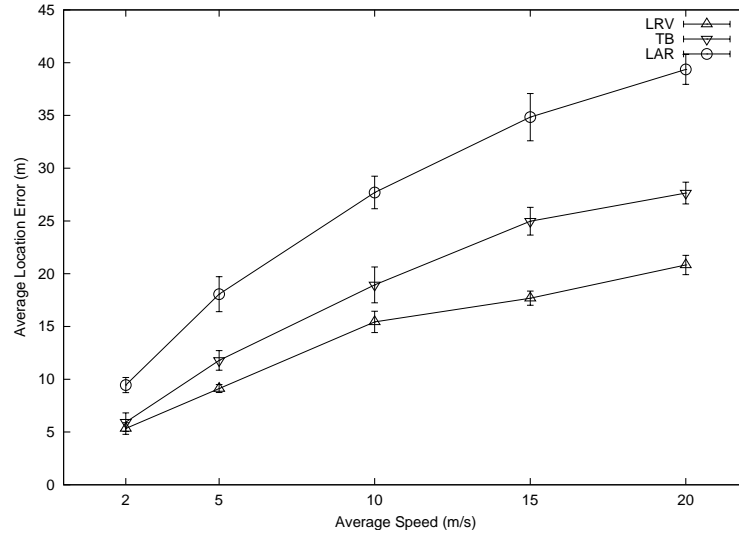
Chapter 5

COMPARING THE TRAVERSAL METHODS

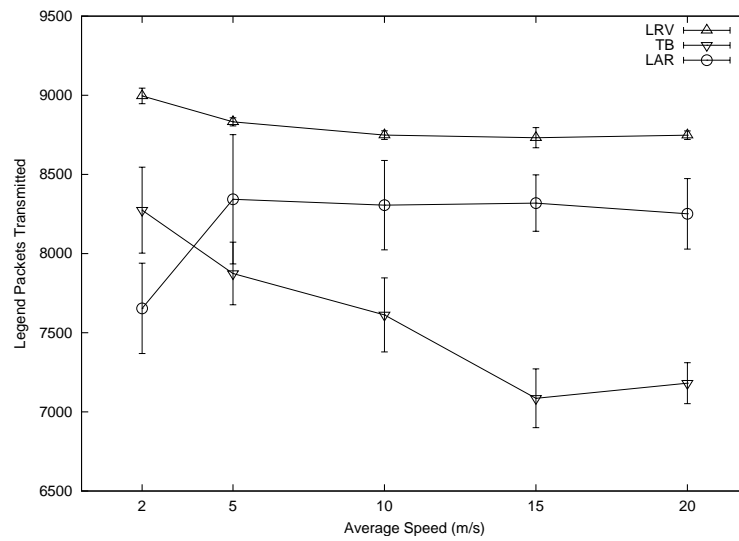
Using the simulation environment described in Chapter 3, we simulated all three of our traversal methods. The performance points in Figure 5.1(a) correspond to the overhead points in Figure 5.1(b) (our definitions of performance and overhead can be found in Chapter 3). Notice that while the LRV traversal method shows the best performance of the three methods, shown in Figure 5.1(a), it also has the most amount of overhead, shown in Figure 5.1(b). The shortcoming of these two figures is that they don't show the full picture: both the performance and the overhead of the traversal methods depend on the legend pause time used, but Figure 5.1 shows points for only one legend pause time for each traversal and for each speed. Also, all three of the traversals occasionally pause the legend, but the frequency of a pause is different for all three.

To get a clearer understanding of how the traversals perform, we created plots of performance vs. overhead for all three traversals. By adjusting the legend pause time of each traversal, we were able to plot a spread of points to more easily compare the trade-off between performance and overhead for each of the traversal methods. We made one such plot for each of our average node speeds, where each point on the plots is the average of ten simulation trials (see Figures 5.2–5.6). Note that the legend pause times used to generate the plots are shown on Figure 5.4. Similar legend pause times were used in the other figures. The confidence intervals are omitted in this figure for clarity.

Note that the results for the different traversal methods do not always cover the same range of overhead. In general the LAR traversal method has higher amounts of overhead, because the LAR traversal creates a lot of routing packets when it uses the LAR proto-



(a) Average Location Error vs. Average Node Speed



(b) Legend Packets Transmitted vs. Average Node Speed

FIG. 5.1. Initial Plots. An unsatisfactory comparison of three legend traversal algorithms. LRV has the smallest error (a) but the highest overhead (b).

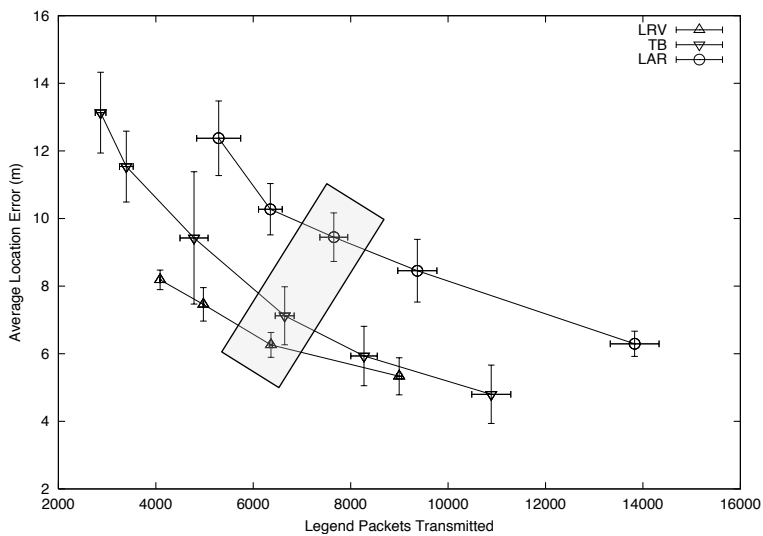


FIG. 5.2. Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 2 m/s)

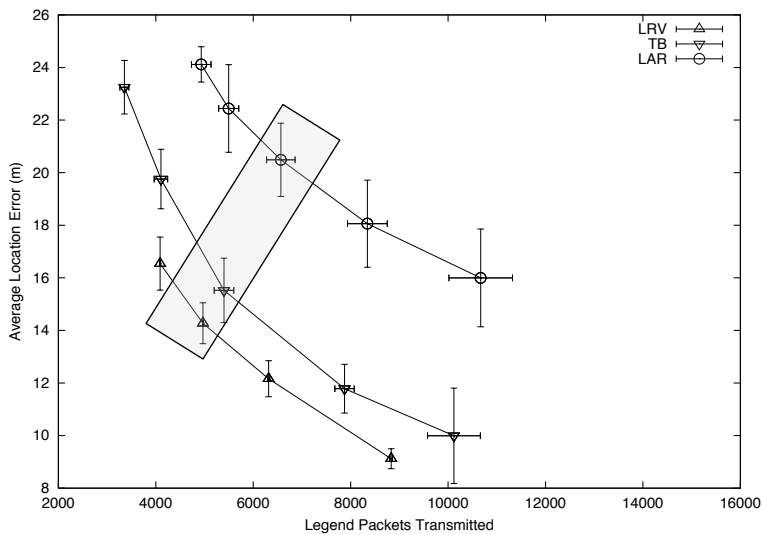


FIG. 5.3. Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 5 m/s)

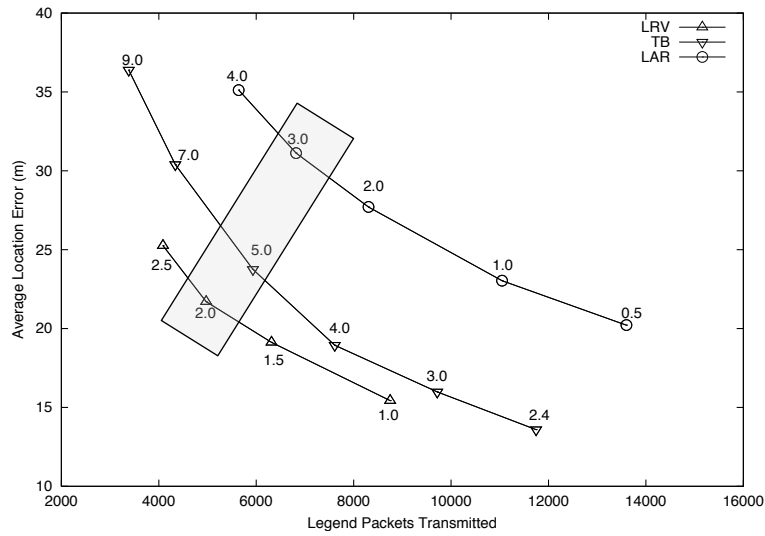


FIG. 5.4. Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 10 m/s). The numbers indicate the legend pause times (in seconds) used to generate the points.

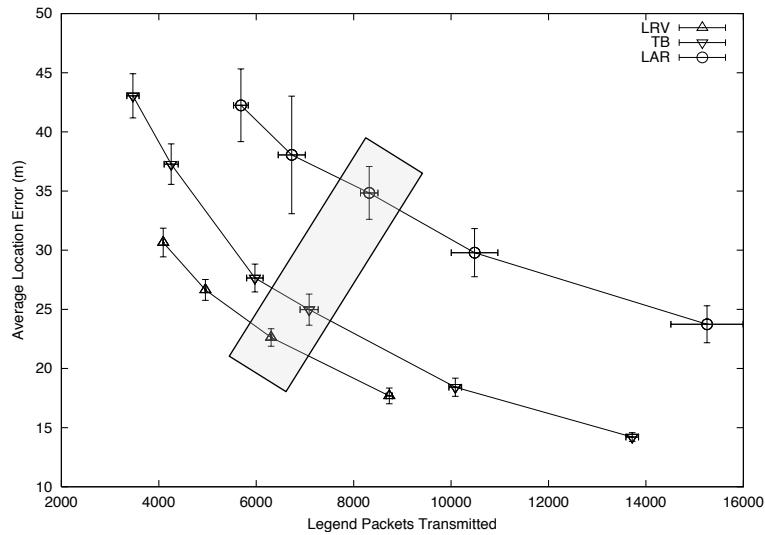


FIG. 5.5. Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 15 m/s)

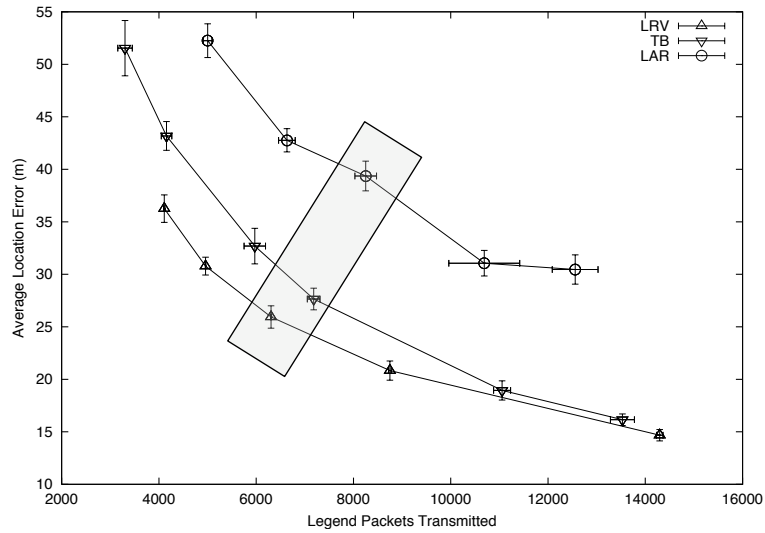


FIG. 5.6. Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 20 m/s)

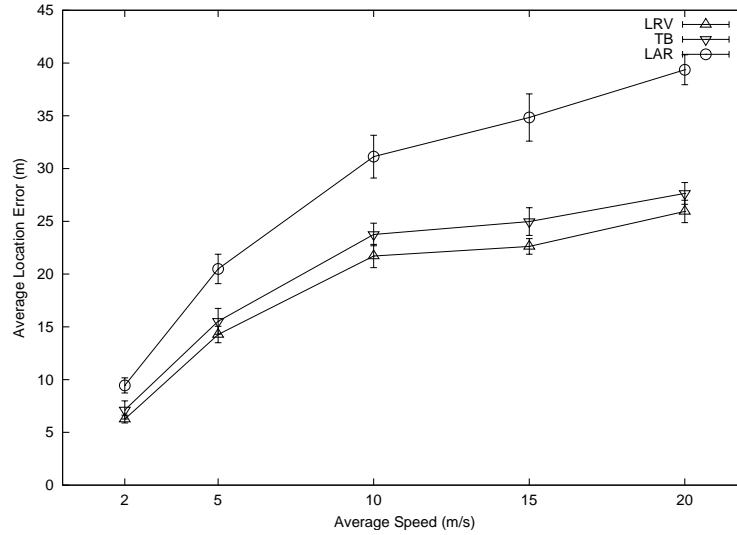
col. Pausing the legend more to reduce the overhead is counter-productive because the performance of the LAR method already suffers in comparison.

As one would expect, Figures 5.2–5.6 all show a decrease in location error as the number of legend packets transmitted increases, giving a trade-off between performance and overhead. By decreasing the time that the legend pauses, the legend is able to traverse the network more often, yielding an increase in both performance and overhead. This trade-off holds for all three of the traversal methods studied and for all of the different average node speeds simulated.

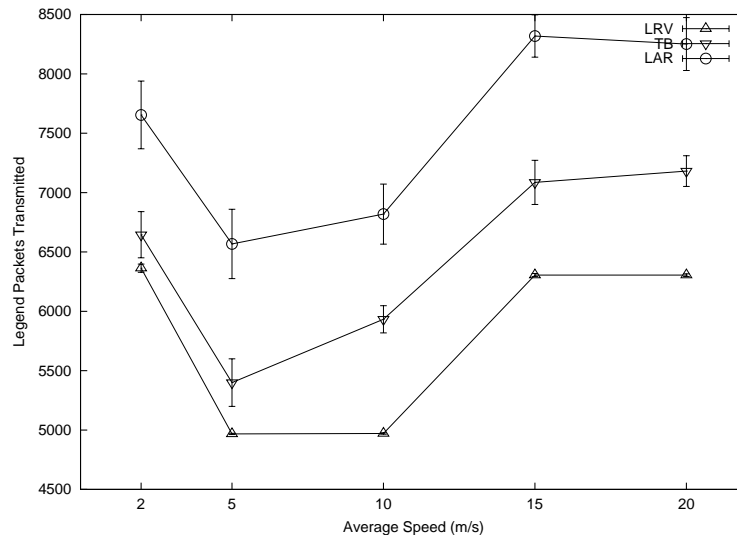
In general, as the average node speed increases, the average location error also increases. This trend is also expected. Suppose the average time between legend visits remains approximately the same regardless of average node speed. Then the average distance that each node travels in the time since the legend’s last visit would increase as the nodes’ speeds increase. This trend holds for all three of the traversal methods studied and for all of the different amounts of overhead simulated.

The real question is, *which of the traversal methods gives the best performance for a given amount of overhead?* Figures 5.2–5.6 indicate that the LRV traversal method gives the greatest performance per overhead (in general) because for any amount of overhead, LRV has the lowest error. Though the TB traversal method seems to give similar performance for very large amounts of overhead, with more reasonable overhead the LRV method performs statistically better for all but the slowest (2 m/s) average node speeds. Even for the slowest speeds, the LRV method’s performance is at least as good as the performance of the TB method. The LAR method performs relatively worse than both the LRV and the TB methods for every speed. Because the LRV traversal gives the best performance overall, we conclude that it is the most efficient traversal method of those studied. A proof that the LRV method will traverse a static, connected network is given in Appendix B.

Figure 5.7 summarizes our results. Similar to Figure 5.1, the performance points in Figure 5.7(a) correspond to the overhead points in Figure 5.7(b). The legend pause times that yield these points were chosen in such a way that the LRV traversal always exhibits the highest performance with the smallest overhead. While this trend holds for all the legend pause times studied (as seen in Figures 5.2–5.6), the chosen points show it clearly in the conventional performance vs. speed and overhead vs. speed plots. While we could have picked any three points in each plot, the actual points chosen are shaded in Figures 5.2–5.6. As seen in Figure 5.7(a), the LRV traversal always shows the best performance, while the LAR traversal always shows the worst performance. While these figures do not show the full picture of the pause spread as shown in Figures 5.2–5.6, they do clearly present our conclusions: The LRV traversal method is the most efficient (i.e., highest performance with the smallest overhead), followed by the TB method. The LAR traversal method [1] is clearly the worst of the methods studied.



(a) Average Location Error vs. Average Node Speed



(b) Legend Packets Transmitted vs. Average Node Speed

FIG. 5.7. Summary Plots. The points in these figures summarize Figures 5.2–5.6. The points chosen for these summary plots are shaded in those figures. LRV has the smallest error (a) and the smallest overhead (b).

Chapter 6

LEGEND RELIABILITY

6.1 Providing Reliable Traversal

In all networks, especially MANETs, reliable transmissions cannot be assumed. Because the entire network depends on the legend for the all-to-all broadcast, it's important for the legend to traverse the network reliably. In this chapter we verify that the results presented so far still hold when the legend traversal methods are made reliable.

To ensure that the legend has not been lost in transmission, when a source node sends the legend, it sets a timer and waits to learn that the destination node received the legend (106, 112, 144, 187). If the source and destination are neighbors, the source node can infer that the destination node received the legend correctly when the destination transmits the legend further. Nodes that receive the legend via the LAR protocol or that decide to pause the legend send an acknowledgment packet (ACK) back to the source node, to inform the source node that the legend transmission succeeded (lines 85–87, 73–75). This ACK is sent via the LAR protocol [5] in the former case (when the nodes are not neighbors) or directly in the latter case (when they are neighbors). If the source node determines that the legend arrived successfully, it cancels its timer (lines 52, 65–66, 117–118). If the source node does not hear the legend forwarded nor receives an ACK, after a short timeout (0.1 s for direct neighbor transmissions and 1 s for LAR protocol transmissions), it assumes the legend transmission failed. In this case the source node removes the destination node from its neighbor table, if needed, picks a new destination (according to its traversal algorithm), and resends the legend (lines 67–71).

6.2 Preventing Multiple Simultaneous Legends

The reliability design described in Section 6.1 prevents the legend from being lost, but it does not solve the whole problem: it is possible that an ACK packet is lost, or that a node does not overhear the legend forwarded. Because of the Two Armies [17] problem (i.e., the node that sends the last packet cannot know if it arrived successfully), there is no way to guarantee that both the destination node received the packet and that the source node realizes it. In this case, the source node may infer that the legend was lost when in fact it was not. Thus when the node resends the legend, it may cause multiple legends to exist simultaneously in the network.

Initially, we used global knowledge to check if the legend transmission had been successful, and prevented nodes from resending the legend when the timers expired if necessary. All of the results presented so far included this global check. These results will be referred to as “idealized” from this point forward.

To remove the global check and avoid the continued existence of multiple legends, we augmented the legend to track how many nodes it has visited in total, including repeated visits. Nodes also maintain the highest number of nodes that any legend it has heard from has visited. If a node receives a legend whose number of visited nodes is less than the node’s stored highest number, then the node determines multiple legends exist. Thus, the node does not retransmit the received legend (i.e., the legend with the smaller number of nodes visited), removing it from the network. This method ensures that at least one legend is allowed to continue propagating throughout the network. We refer to simulation results that do not include the global check as “realistic” from this point forward.

6.3 Minimizing the Duration of Multiple Simultaneous Legends

We explored three different policies to reduce the duration that multiple legends existed in the network simultaneously. All of them are policies on the actions for a node to take when its last transmission of the legend failed. The idea behind the policies is that nodes around the destination node that the source node believes did not receive the legend are likely to know if the legend was actually received by the destination node.

Resend to Failed Node Using this policy, the source node sends the legend to the destination node a second time, waiting only a very short time before assuming that the retransmission was not successful. If the destination receives two legends, it can ignore the second one, thereby minimizing the duration of the extra legend.

Send to Neighbor of Failed Node Using this policy, the source node sends the legend to its neighbor that is closest to the failed destination node. If this neighbor overheard the transmission of the legend by the destination node, then this neighbor can return an acknowledgement to the source node and delete the extra legend.

Adjust Visit Time of Failed Node Using this policy, the source node adjusts the failed destination node's entry in the legend table so that its timestamp is older, thus increasing that node's priority to be visited by the legend. When the destination receives this legend, it can ignore it if the previous legend was received correctly.

To evaluate the policies, we simulated them and compared their performance. Figures 6.1–6.5 show the results of our comparison. As shown in the figures, none of the policies showed any significant improvement on the performance of the legend. We suspect this is because all three methods change the LRV traversal method. The event where the legend needs to be resent happens far more often than the event where an extra legend

Legend Resends	
Resends (average)	400
Extra Legends Created (average)	25
Extra Legends Exist For	
Hops (average)	5
Median Duration	.1 s

Table 6.1. Reliability Statistics

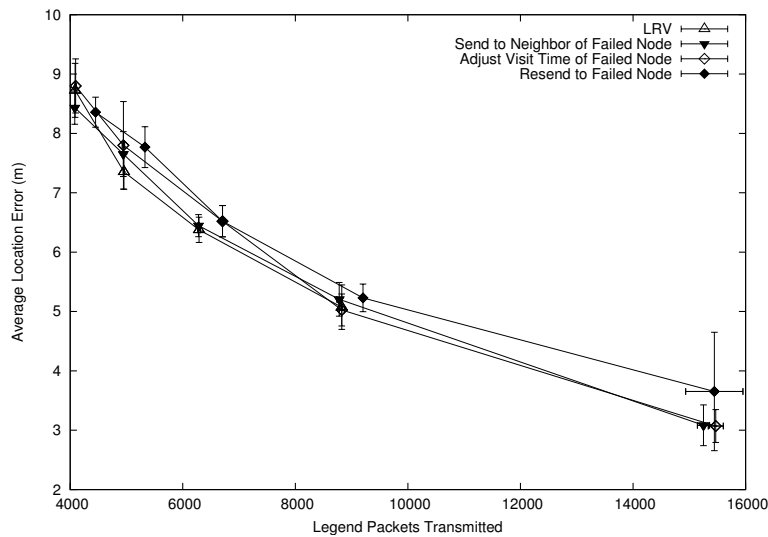


FIG. 6.1. Legend Reliability Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 2 m/s)

is spawned, so acting to reduce the amount of time that multiple legends exist can actually decrease the overall performance. Therefore, we decided it is best to not use any of the methods to reduce the time that multiple legends exist, but to simply let the extra legends be discarded at their natural time. For example, in the LRV traversal method, our simulation had a node resend the legend 400 times and create an extra legend only 25 times (see Table 6.1).

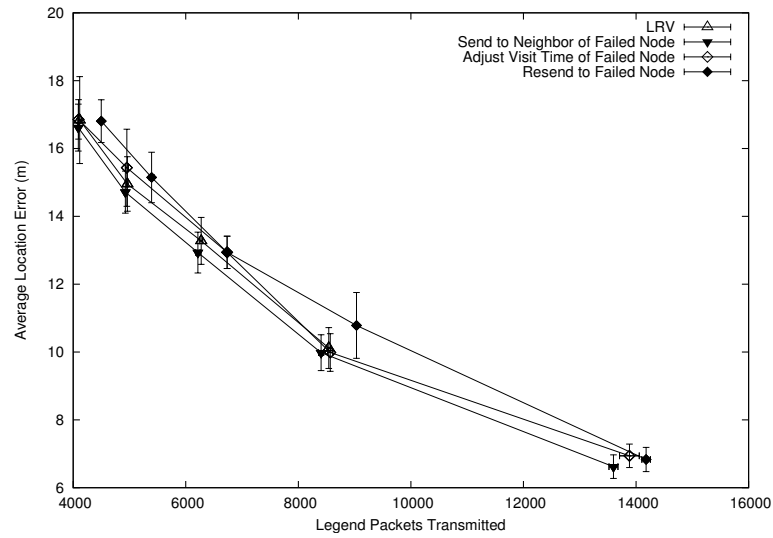


FIG. 6.2. Legend Reliability Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 5 m/s)

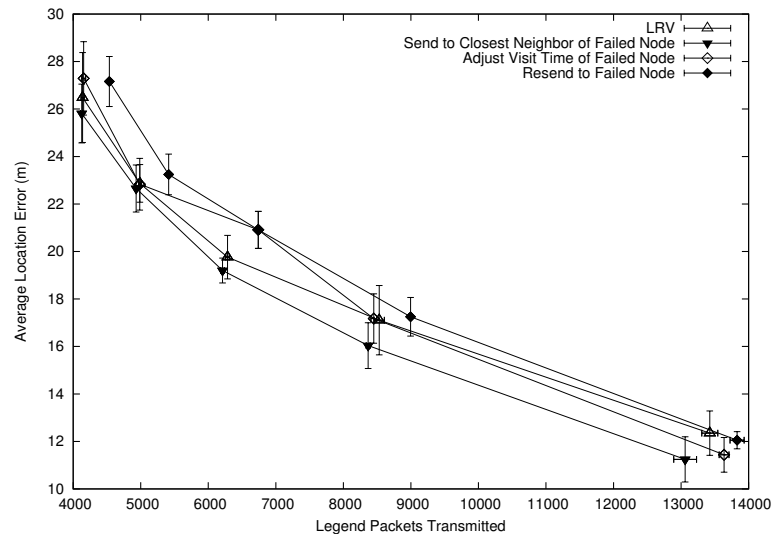


FIG. 6.3. Legend Reliability Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 10 m/s)

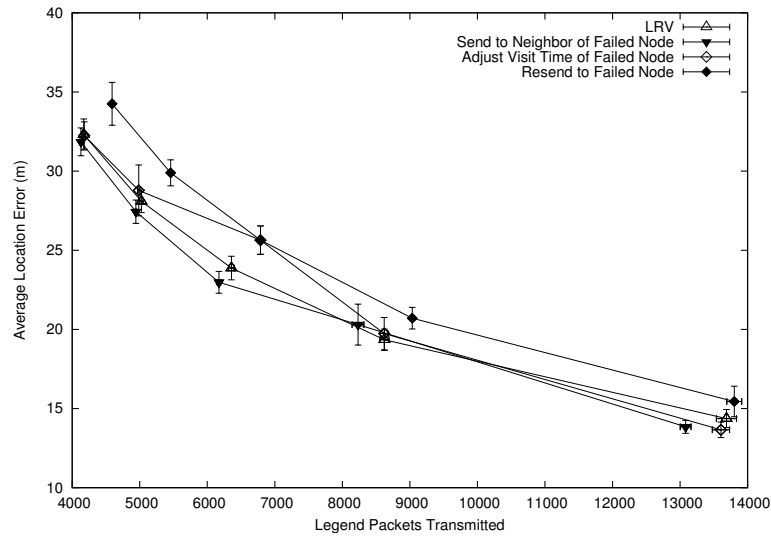


FIG. 6.4. Legend Reliability Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 15 m/s)

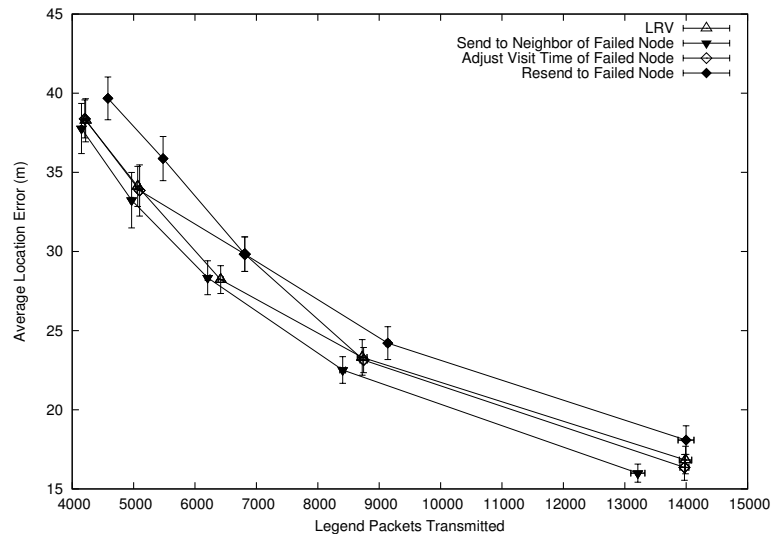


FIG. 6.5. Legend Reliability Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 20 m/s)

6.4 Simulation Results on Legend Reliability

After determining the best traversal method, we modified the simulator to handle multiple legends and removed our global check (see Section 6.2). That is, we extended the idealistic LRV implementation to a realistic LRV implementation. In doing so, we found that the performance of the realistic LRV decreased slightly, because extra overhead and extra traffic in the network occur during the existence of multiple legends. We compare the results of the realistic LRV simulations with the idealized TB simulations (see Figures 6.6–6.10) in the rest of this chapter.

Figures 6.6–6.10 compare the idealized LRV and TB traversal methods (which use global knowledge to prevent multiple legends) with the realistic LRV method (which does not use global knowledge). Similar to Figures 5.2–5.6, these figures show the trade-off of performance and overhead over a spread of legend pause times, for each of our average node speeds. As shown, even without global knowledge, the LRV traversal offers the best performance per amount of overhead (in general). There are a few cases, when the overhead is very high, that the realistic LRV and the idealized TB have statistically similar performance; however, the overall trend clearly shows the realistic LRV traversal method to be more efficient than the idealized TB traversal method.

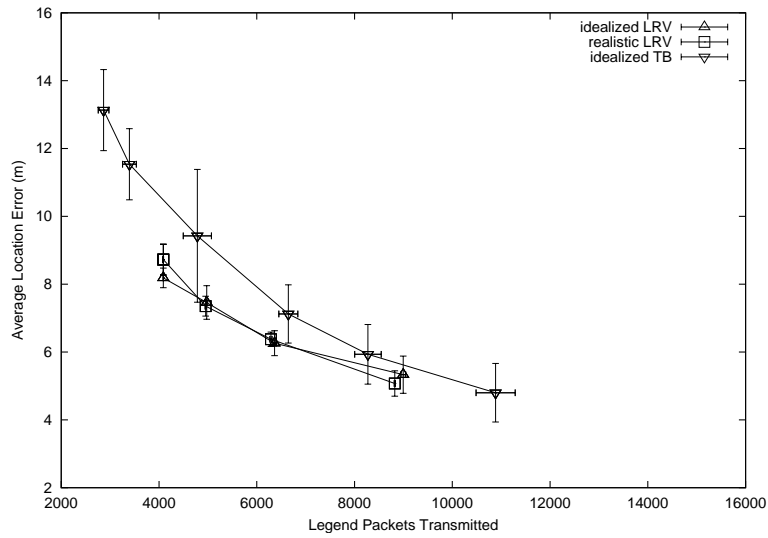


FIG. 6.6. Realistic LRV Method Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 2 m/s)

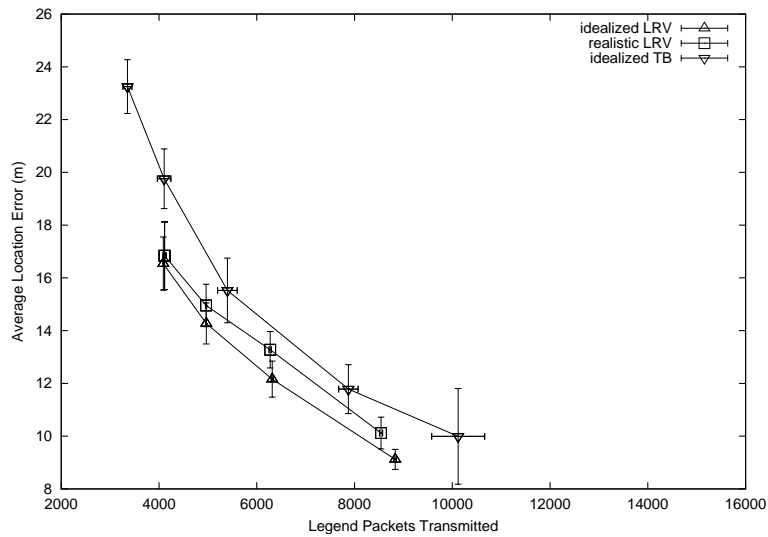


FIG. 6.7. Realistic LRV Method Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 5 m/s)

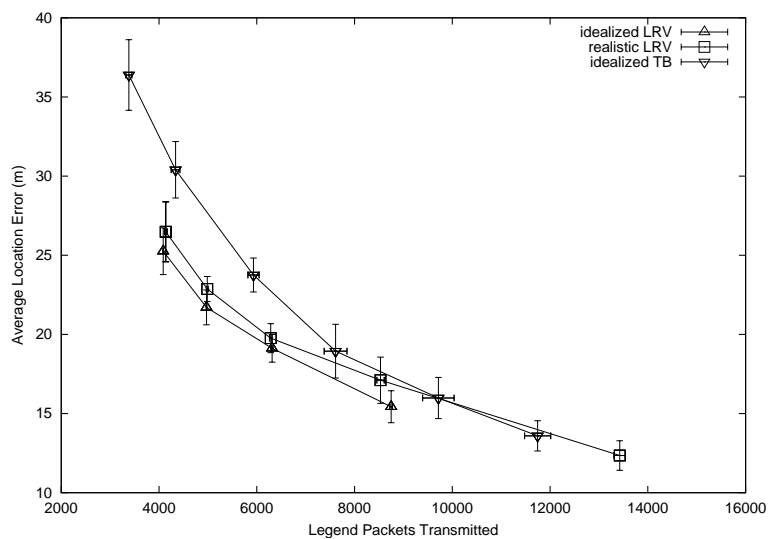


FIG. 6.8. Realistic LRV Method Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 10 m/s)

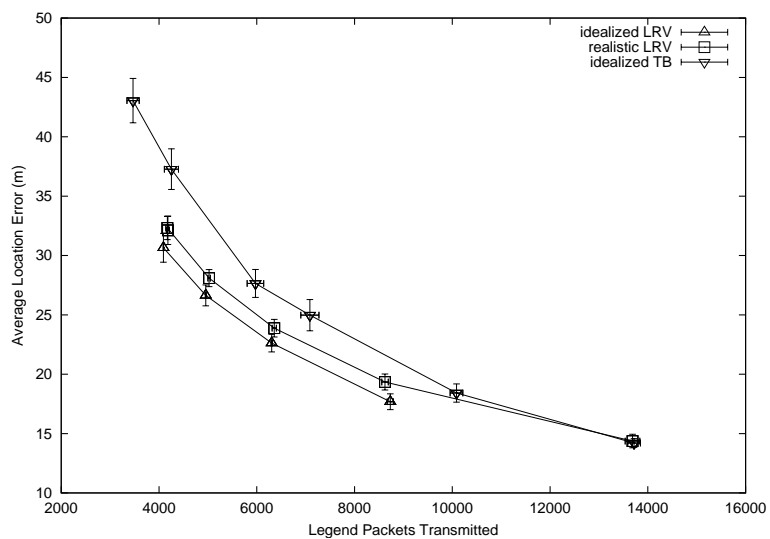


FIG. 6.9. Realistic LRV Method Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 15 m/s)

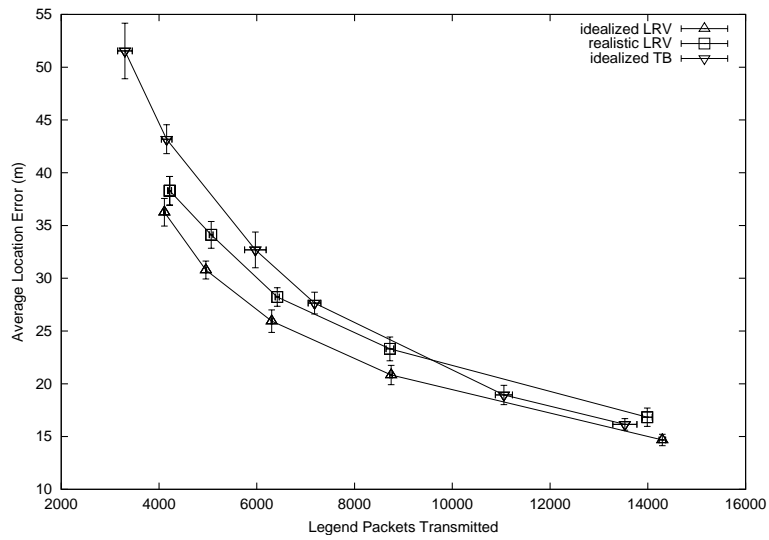


FIG. 6.10. Realistic LRV Method Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 20 m/s)

Chapter 7

OPTIMIZING THE LRV METHOD

7.1 Legend Pause Interval

The LRV traversal method has no natural point to pause the legend. Instead, the legend pauses after visiting i nodes, where i is the legend pause interval. The value used for i can affect the performance of the legend. A smaller value of i results in the time the legend spends paused to be more evenly distributed, but also results in more overhead spent on sending ACKs (see Chapter 6.1). Determining the optimal value for i requires investigation.

We executed several simulations to evaluate how different values of i affect the legend's performance. Figures 7.1–7.5 show the results of these simulations. As shown in the figures, a value of $i = 25$ hops yields the best overall performance with the least overhead of the legend pause intervals simulated.

7.2 LRV Update vs. Visit

In order to implement the LRV method, two timestamps are necessary for each entry in the legend: one timestamp to maintain when each node's location was last updated, and another timestamp to store when each node was actually visited by the legend. The two are not always the same, because nodes share their locations in each "hello" packet they transmit and this information is in turn shared with the legend. If the legend is to be sent to the least recently *visited* neighbor, the time that each node was last visited must be available. Without a second timestamp, the legend can only be sent to the least recently

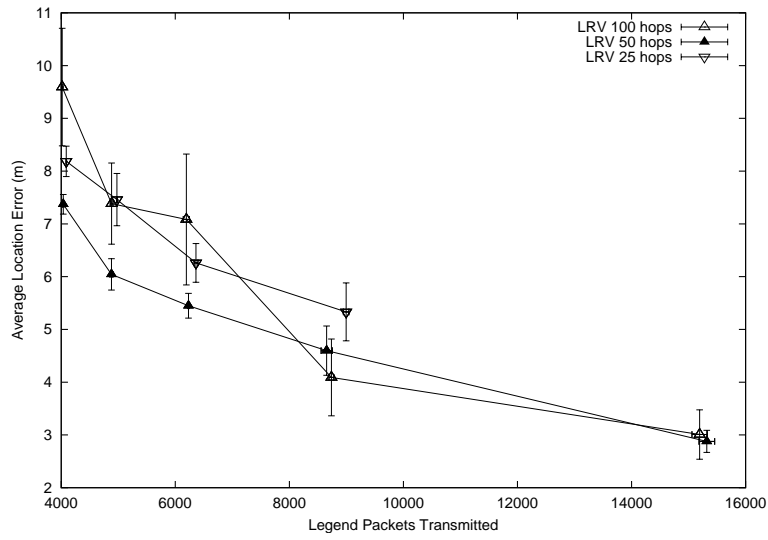


FIG. 7.1. Pause Interval Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 2 m/s)

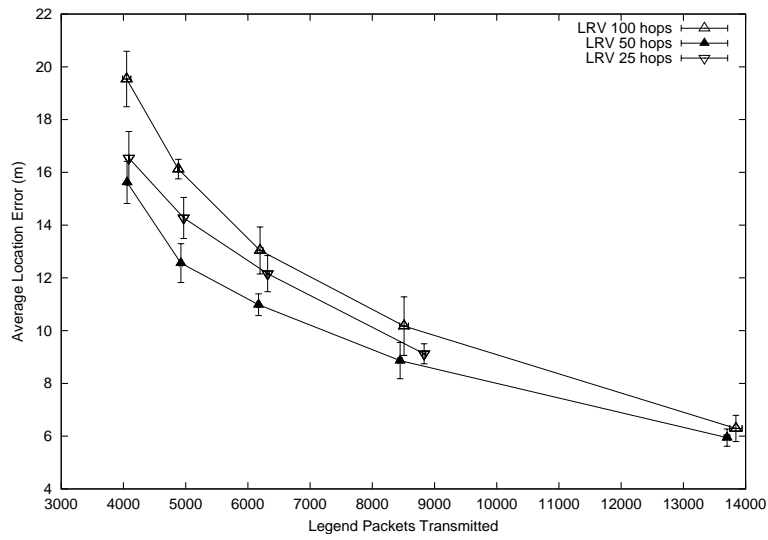


FIG. 7.2. Pause Interval Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 5 m/s)

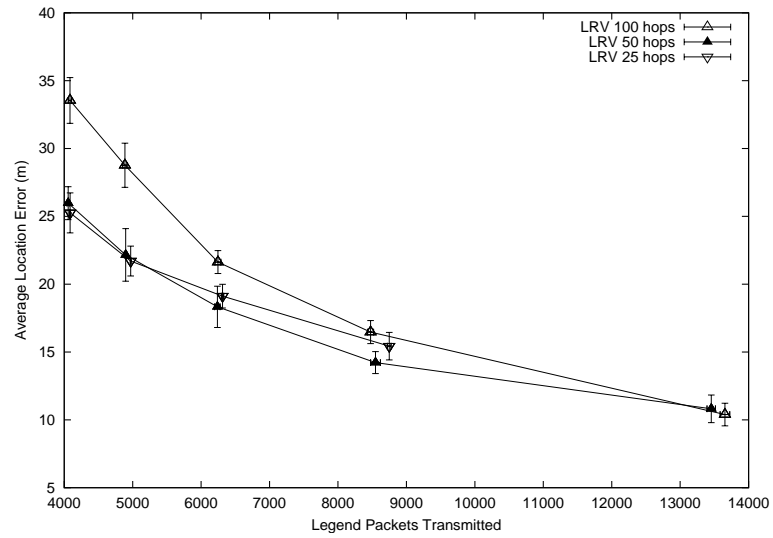


FIG. 7.3. Pause Interval Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 10 m/s)

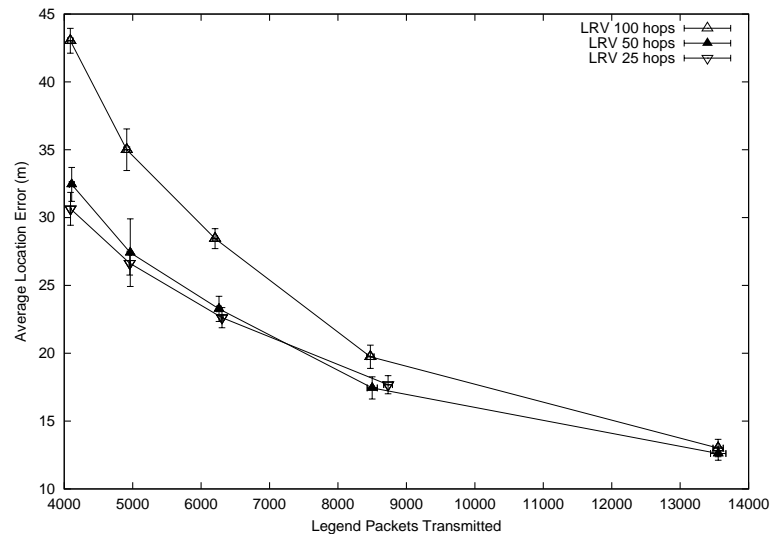


FIG. 7.4. Pause Interval Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 15 m/s)

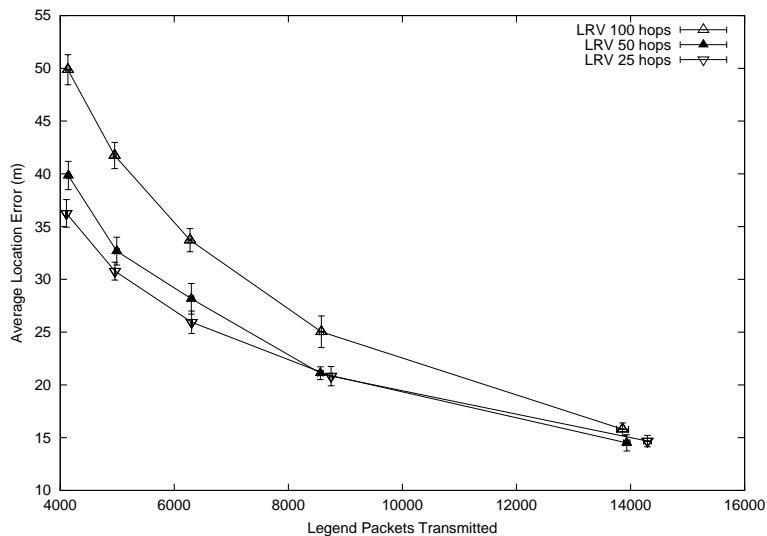


FIG. 7.5. Pause Interval Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 20 m/s)

updated (LRU) neighbor, i.e., the neighbor whose location information in the legend has been updated less recently than any other neighbor.

Storing an extra “visited” timestamp for each legend entry adds overhead to the legend; however, the legend’s performance increases by following the LRV method, as opposed to the LRU method. This chapter explores the trade-off of performance and overhead concerned with the updated/visited issue.

7.3 LRV Visit Approximation

The LRV method can be approximated without the additional overhead of extra timestamps in the legend. If the legend itself does not store the last visited times of all the nodes, it can still be sent to the LRV neighbor if the nodes themselves store this information. Using the LRV approximation, nodes maintain the time that each of their neighbors was last visited by the legend in their neighbor tables. That is, nodes include their last visited times

in their “hello” packets.

When determining the least recently visited neighbor, a node has two timestamps to use: the last update time from the legend, and the last legend visit time from its neighbor table. If the node has received a “hello” packet from the neighbor at least as recently as the legend’s “update” timestamp, it uses the neighbor’s last legend visit time in its calculations. Otherwise, it uses the neighbor’s “update” timestamp in the legend.

Using this method, the legend can approximate the LRV method without the overhead of transmitting extra “visit” timestamps in the legend. Although “hello” packets are slightly larger with this method, the legend produces a lower error by approximating the LRV visit method without increasing the size of the legend.

7.4 Simulation Results on LRV Update vs. Visit

In this section, we explore the performance/overhead trade-off of the update vs. visit controversy in simulation. Figures 7.6–7.10 show the results of these simulations. As shown, the LRV visit policy achieves the best performance per overhead. The overhead in these figures is measured by legend packets transmitted. The additional overhead used by the LRV visit method is not shown because no additional packets are transmitted; rather, each legend packet is slightly larger than in the LRV update and LRV visit approximation policies. Because the LRV visit policy yields the best performance, we determine that it is the best policy. Unless otherwise noted, the LRV visit policy is used in all of our LRV traversal method simulations.

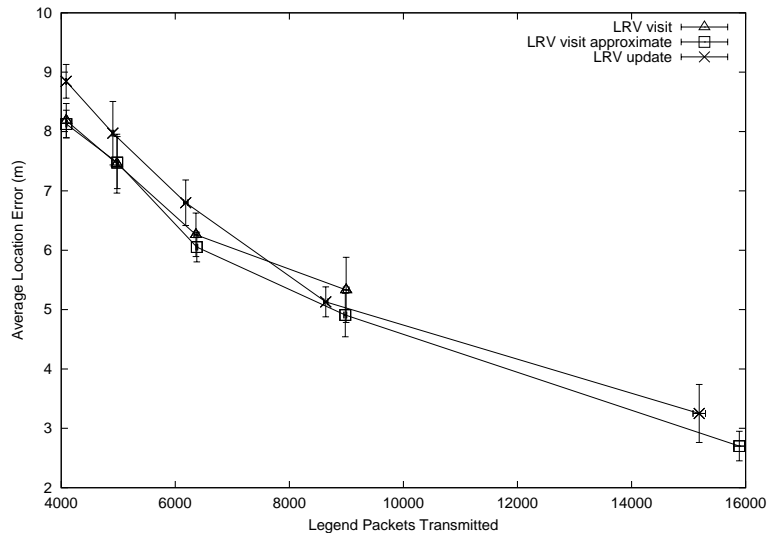


FIG. 7.6. Update/Visit Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 2 m/s)

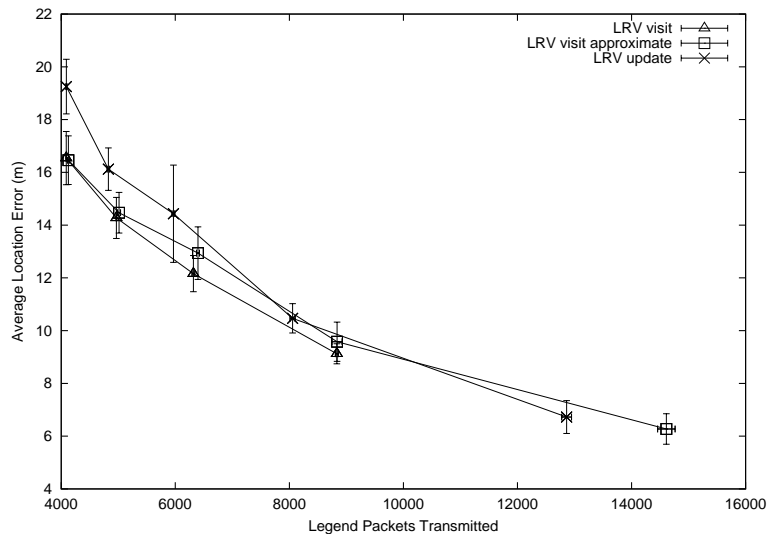


FIG. 7.7. Update/Visit Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 5 m/s)

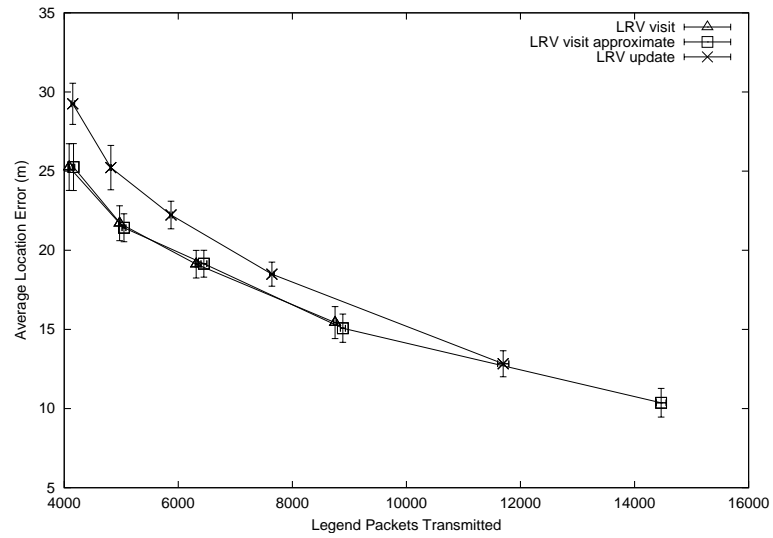


FIG. 7.8. Update/Visit Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 10 m/s)

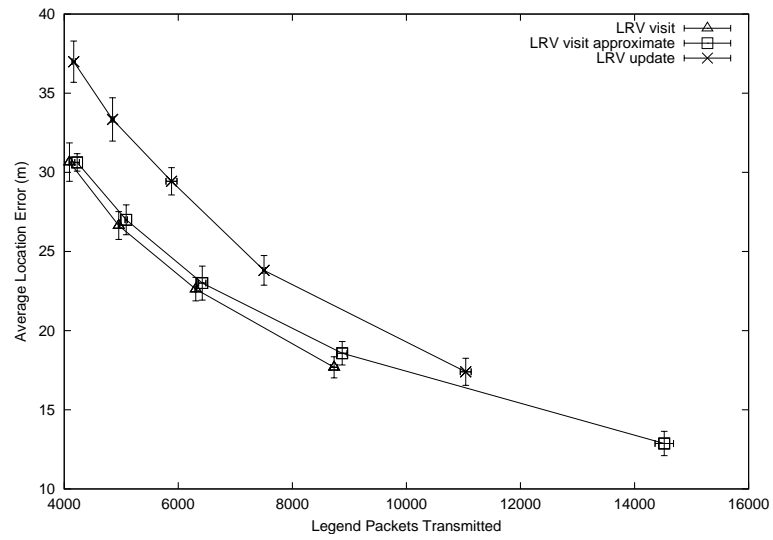


FIG. 7.9. Update/Visit Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 15 m/s)

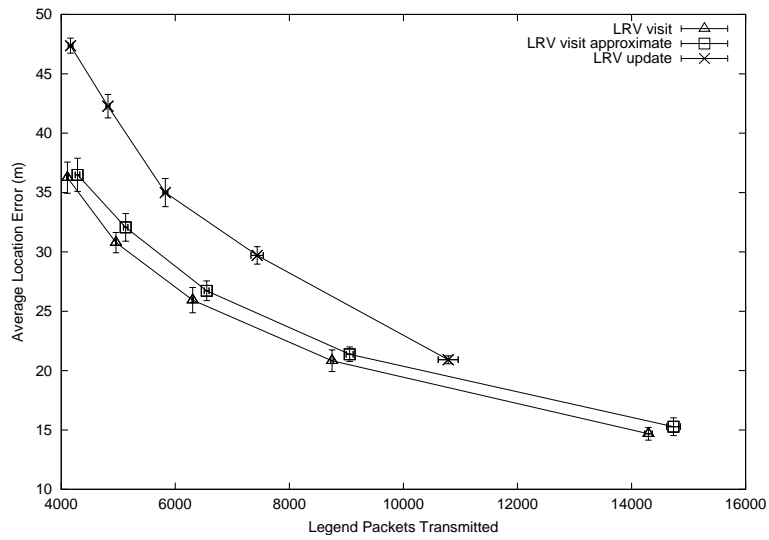


FIG. 7.10. Update/Visit Performance vs. Overhead: Average Location Error vs. Legend Packets Transmitted (average node speed 20 m/s)

Chapter 8

CONCLUSIONS

In this study, we took the idea of a legend—a data structure that is passed around a network to share information—and developed three different ways for such a legend to traverse an ad hoc network. We then compared the traversal methods in simulation. The results of this study show that, of the three traversal methods studied, the LRV method performs best. Although the TB method showed statistically similar performance under a few network conditions, the LRV method had the best overall performance.

The legend is a useful all-to-all broadcast protocol for a mobile ad hoc network. In addition to providing location information, the legend can be used for gathering and sharing any kind of information among the nodes of the network. In addition, by adjusting the legend pause time, the legend can easily fit applications where different amounts of overhead or performance are tolerable. This adjustability gives the legend the versatility to be used for all kinds of information sharing applications.

In addition to comparing three traversal methods, we explored several ways to improve their performance. We also designed a way to make the legend traversal reliable in an unreliable network. Therefore nodes in a network can depend on the legend for accurate, up-to-date information with adjustable overhead. This efficient algorithm for performing an all-to-all broadcast will directly help other protocols that require information from other nodes in a network.

Though we have found an efficient way for a legend to traverse a MANET, there remain several issues regarding legends to be further researched. So far we have focused on the traversal methods themselves; we chose not to send data communication in the

network in order to simplify matters. However, the effect of data traffic on the legend's performance must be explored.

In this thesis, we have explored an environment with one legend in a fairly small network of 50 nodes. While the legend performs quite well in these conditions, we desire a legend service that can scale to larger networks. One idea is to have multiple simultaneous legends, possibly behaving in a hierarchical manner. Such scalability ideas is an area of future work.

Additional applications of the legend service, other than distributing location information, should be explored. Any other information would probably be shared in addition to location information, because the location information itself makes the legend traversal more efficient. With the legend, an all-to-all broadcast operation is no longer an infeasible operation in a MANET. Possible applications include sharing topology information to ease the task of routing and stand-alone all-to-all broadcasting applications. Analysis of such applications is needed to determine the efficiency of the legend for uses other than those studied in this thesis.

REFERENCES

- [1] X. Jiang and T. Camp. An information dissemination protocol for an ad hoc network. In *Proceedings of the 23rd IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 337–345, 2004.
- [2] T. Camp, J. Boleng, and L. Wilcox. Location information services in mobile ad hoc networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 3318–3324, 2002.
- [3] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 120–130, 2000.
- [4] S. Lindsey and C. Raghavendra. Energy efficient broadcasting for situation awareness in ad hoc networks. In *Proceedings of the IEEE International Conference on Parallel Processing (ICPP)*, pages 149–155, 2001.
- [5] Y. Ko and N.H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 66–75, 1998.
- [6] I. Stojmenovic, M. Russell, and B. Vukojevic. Depth first search and location based localized routing and QoS routing in wireless networks. In *Proceedings of the IEEE International Conference on Parallel Processing*, pages 173–180, 2000.
- [7] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network Magazine*, 15(6):30–39, 2001.

- [8] I. Stojmenovic. Position based routing in ad hoc networks. *IEEE Communications Magazine*, 40(7):128–134, July 2002.
- [9] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. MIT Press, 2001.
- [10] M. Dorigo and L. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
- [11] D. Camara and A. Loureiro. A novel routing algorithm for ad hoc networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, 2000.
- [12] The VINT Project. The network simulator - ns-2. <http://www.isi.edu/nsnam/ns/>. Page accessed on April 26th, 2004.
- [13] W. Navidi, T. Camp, and N. Bauer. Improving the accuracy of random waypoint simulations through steady-state initialization. In *Proceedings of the 15th International Conference on Modeling and Simulation (MS)*, pages 319–326, 2004.
- [14] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1312–1321, 2003.
- [15] J. Yoon, M. Liu, and B. Noble. Sound mobility models. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 205–216, 2003.
- [16] J. Boleng. Normalizing mobility characteristics and enabling adaptive protocols for ad hoc networks. In *Proceedings of LANMAN 2001: 11th IEEE Workshop on Local and Metropolitan Area Networks*, pages 9–12, March 2001.

- [17] A. Tanenbaum. *Computer Networks, Fourth Edition*. Prentice Hall, 2003.

Appendix A

TRAVERSAL METHOD PSEUDOCODE

Traversal Methods:

LAR Location-Aided Routing algorithm

LRV Least Recently Visited algorithm

TB TraceBack algorithm

A.1 Common Pseudocode

Common Variables:

<i>GLT</i>	Global Location Table (table entries include ts, v_bit, x, y, indexed by node IDs)
<i>LLT</i>	Local Location Table (table entries include ts, x, and y, indexed by node IDs)
<i>NT</i>	Neighbor Table (hash table mapping node IDs to time stamps)
<i>ts</i>	Time Stamp (time entry was last updated)
<i>v_bit</i>	Visited Bit (has entry been updated?)
<i>x</i>	Position in the x-direction
<i>y</i>	Position in the y-direction
<i>prev</i>	Previous node that last had the legend
<i>send_feedback</i>	Send feedback bit (should node that received the legend send feedback to sender?)
<i>legend_timer</i>	A node's timer to check that the legend was received correctly
<i>hello_timer</i>	A node's timer to periodically send hello packets
<i>pause_timer</i>	A node's timer to restart the legend after pausing
<i>traversal_method</i>	The traversal algorithm to be used (LAR, LRV or TB)
<i>next_dest</i>	The next destination for the legend
<i>id</i>	The unique identifier of a node
<i>USE_PROMISCUOUS_MODE</i>	Promiscuous mode bit (should nodes update their LLTs when they overhear the legend being transmitted?)
<i>CHECK_NEIGHBORS</i>	Check neighbors bit (should nodes remove old entries in their NT?)
<i>HELLO_INTERVAL</i>	Period of time between sending hello packets
<i>NEIGHBOR_INTERVAL</i>	Period of time before a node assumes its neighbor did not receive the legend
<i>PAUSE_LEGEND</i>	Pause legend bit (should the legend pause periodically?)
<i>PAUSE_TIME</i>	Period of time the legend is paused
<i>CURRENT_TIME</i>	The current simulation time in seconds

Common Subroutines:

Initialize all the Global Location Table (legend) entries and all the nodes' Local Location Table and Neighbor Table entries.

INITIALIZE():

```

1 // Initialize GLT
2 foreach node entry  $i$  in the GLT
3   Set  $GLT[i].v\_bit \leftarrow false$ ;
4   Set  $GLT[i].ts \leftarrow 0$ ;
5   Set  $GLT[i].x \leftarrow \infty$ ;
6   Set  $GLT[i].y \leftarrow \infty$ ;
7 // Initialize LLT
8 foreach node  $I$  in the network
9   Set  $I.send\_feedback \leftarrow false$ ;
10  Set  $I.NT \leftarrow \{\}$ ;
11  Set  $I.hello\_timer(HELLO\_INTERVAL + \text{small random jitter})$ ;
12  foreach node entry  $j$  in  $I$ 's LLT
13    Set  $I.LLT[j].ts \leftarrow 0$ ;
14    Set  $I.LLT[j].x \leftarrow \infty$ ;
15    Set  $I.LLT[j].y \leftarrow \infty$ ;
16 // Start legend
17 if  $A.id = 0$ 
18    $A.SEND\text{-}LEGEND\text{-}PACKET()$ ;
```

Nodes periodically create and send hello packets to their neighbors, containing their own location.

HELLO-TIMER-EXPIRES:

```

19 if node  $A$ 's hello timer expires
20   Create a hello packet  $h$ ;
21   Set  $h.ts \leftarrow CURRENT\_TIME$ ;
22   Set  $h.x \leftarrow A.x$ ;
23   Set  $h.y \leftarrow A.y$ ;
24   Broadcast  $h$  to all neighbors of  $A$ ;
25   Set  $A.hello\_timer(HELLO\_INTERVAL)$ ;
```

Nodes will update their Local Location Tables and Neighbor Tables when they receive a hello packet from one of their neighbors.

RECEIVE-HELLO-PACKET:

```

26 if node  $A$  receives hello packet  $h$  from its neighbor  $B$ 
27     // Update  $B$ 's entry in  $A$ 's LLT
28     Set  $A.LLT[B.id].ts \leftarrow h.ts$ ;
29     Set  $A.LLT[B.id].x \leftarrow h.x$ ;
30     Set  $A.LLT[B.id].y \leftarrow h.y$ ;
31     // Update  $A$ 's neighbor table
32     if  $B.id \notin A.NT$ 
33         Insert  $B.id$  into  $A.NT$ ;
34     Set  $A.NT[B.id] \leftarrow h.ts$ ;

```

Upon receiving the legend, if the node is the intended recipient, the node will bring both the Global Legend Location Table and its own Local Location Table up-to-date. Then it will send the legend onward.

$A::$ HANDLE-LEGEND():

```

35 Set  $A.prev = id$  of node that sent legend packet;
36 // Update  $A$ 's entry in the GLT
37 Set  $GLT[A.id].v\_bit \leftarrow true$ ;
38 Set  $GLT[A.id].ts \leftarrow CURRENT\_TIME$ ;
39 Set  $GLT[A.id].x \leftarrow A.x$ ;
40 Set  $GLT[A.id].y \leftarrow A.y$ ;
41 // Update the GLT with  $A$ 's more recent information and vice versa
42 foreach node entry  $i \in GLT \cup A.LLT$ 
43     if  $A.LLT[i].ts > GLT[i].ts$ 
44         Set  $GLT[i].ts \leftarrow A.LLT[i].ts$ ;
45         Set  $GLT[i].x \leftarrow A.LLT[i].x$ ;
46         Set  $GLT[i].y \leftarrow A.LLT[i].y$ ;
47     else
48         Set  $A.LLT[i].ts \leftarrow GLT[i].ts$ ;
49         Set  $A.LLT[i].x \leftarrow GLT[i].x$ ;
50         Set  $A.LLT[i].y \leftarrow GLT[i].y$ ;
51  $A.SEND-LEGEND-PACKET()$ ;

```


If the node overheard the legend sent to another node, it will cancel its legend timer because it knows the legend has been received correctly since it last sent the legend. If the node is operating in promiscuous mode, it will update its own Local Location Table with information from the Global Legend Location Table.

A::OVERHEAR-LEGEND():

```

52 Cancel A.legend_timer; // heard legend forwarded
53 if USE_PROMISCUOUS_MODE
54     // Update A's LLT with the GLT's more recent information
55     foreach node entry i in the GLT
56         if GLT[i].ts > A.LLT[i].ts
57             Set A.LLT[i].ts ← GLT[i].ts;
58             Set A.LLT[i].x ← GLT[i].x;
59             Set A.LLT[i].y ← GLT[i].y;

```

The node with the legend will remove old entries from its neighbor table before using its traversal algorithm to pick the next destination for the legend.

A::CHECK-NEIGHBORS():

```

60 if CHECK_NEIGHBORS
61     // Remove expired neighbor table entries
62     foreach entry i ∈ A.NT
63         if CURRENT_TIME − A.NT[i] > 2* HELLO_INTERVAL
64             Remove i from A.NT;

```

Upon receiving a feedback packet, a node will cancel its legend timer because it knows that the next destination received the legend correctly.

RECEIVE-FEEDBACK-PACKET:

```

65 if node A receives a feedback packet
66     Cancel A.legend_timer;

```

When a node's legend timer expires, the node will remove the attempted next destination node from its Neighbor Table if applicable and will then pick a new destination and resend the packet.

LEGEND-TIMER-EXPIRE:

```

67 if node A's legend timer expires
68     if A.next_dest ∈ A.NT;
69         Remove A.next_dest from A.NT;
70     Set A.send_feedback ← false; // don't need to send feedback again
71     A.SEND-LEGEND-PACKET();

```

A node with the legend will pause the legend by setting its pause timer, without sending the legend on. If needed, it will send a feedback packet to the previous node.

A::PAUSE-LEGEND():

```

72 Set A.pause_timer(PAUSE_TIME);
73 if A.send_feedback
74     Send feedback packet directly to neighbor A.prev;
75     Set A.send_feedback ← false;

```

When a node's pause timer expires, the node will pick a new destination and send off the legend.

PAUSE-TIMER-EXPIRE:

```

76 if node A's pause timer expires
77     Set A.send_feedback ← false; // don't need to send feedback after pause
78     A.SEND-LEGEND-PACKET();

```

A.2 LAR Pseudocode

LAR Variables:

<i>tries</i>	Number of times the algorithm has tried to send the node the legend, stored in the GLT
<i>MAX_TRIES</i>	Maximum number of times the algorithm will try to send a node the legend in a given loop
<i>LAR_INTERVAL</i>	Period of time before a node assumes its non-neighbor did not receive the legend

LAR Subroutines:

Initialize the LAR-specific variables.

INITIALIZATION:

```

79 // Initialize GLT
80 foreach node entry i in the GLT
81     Set GLT[i].tries ← 0;
82 INITIALIZE();

```

A node receiving a legend packet will handle the legend and send an ACK packet or set the feedback bit as appropriate to confirm receipt of legend if it is the legend's destination, or overhear it if not.

RECEIVE-LEGEND-PACKET:

```
83 if node A receives the legend
84     if the legend packet's destination is A
85         if A received legend from LAR
86             Send ACK packet to A.prev using LAR;
87             Set A.send_feedback  $\leftarrow$  false; // A.prev can receive ACK
88         else
89             Set A.send_feedback  $\leftarrow$  true;
90             A.HANDLE-LEGEND();
91     else // legend is not destined for A
92         A.OVERHEAR-LEGEND();
```

The node with the legend will pick the legend's next destination according to the LAR traversal method, and will then send it on. If the next destination is not a neighbor, the node will send the legend via LAR and will also send a feedback packet to the previous node, unless an ACK or feedback packet has already been sent to it or it has already had a chance to hear the legend forwarded. Otherwise, the legend is sent directly. For added reliability, nodes set a timer to resend the legend if they do not hear the legend forwarded or receive an ACK or feedback packet before it expires. Note: a node will pause the legend instead of sending it on if it is time to reset the loop.

A::SEND-LEGEND-PACKET():

```

93  if all nodes in GLT have been visited or tried at least MAX_TRIES times
94      A.RESET-LOOP();
95      if PAUSE_LEGEND
96          A.PAUSE-LEGEND();
97      return;
98  // Find legend's next destination
99  A.CHECK-NEIGHBORS();
100 Set A.next_dest ← A.GET-NEXT-DESTINATION-FOR-LAR();
101 // Send legend to next destination
102 Increment GLT[A.next_dest].tries;
103 if A.next_dest ∈ A.NT // destination is a neighbor
104     Send legend packet directly to neighbor A.next_dest;
105     Set A.send_feedback ← false; // previous node can hear legend forwarded
106     Set A.legend_timer(NEIGHBOR_INTERVAL);
107 else // destination is not a neighbor
108     Send legend packet to A.next_dest using LAR;
109     if A.send_feedback = true
110         Send feedback packet directly to neighbor A.prev;
111         Set A.send_feedback ← false;
112     Set A.legend_timer(LAR_INTERVAL);

```

The LAR traversal method picks the closest unvisited node to send the legend to. If the closest unvisited node is not a neighbor of the current node, the legend will be sent to the closest node that has not been tried more than a given threshold times.

A::GET-NEXT-DESTINATION-FOR-LAR():

```

113 Set next_dest ← closest unvisited node to A;
114 if next_dest ∉ A.NT // not neighbor
115     Set next_dest ← closest unvisited node to A
                                that has not been tried MAX_TRIES times;
116 return next_dest;

```

When a node receives an ACK packet via LAR, it will cancel its legend timer because it knows that the next destination received the legend via LAR correctly.

RECEIVE-ACK-PACKET-FROM-LAR:

```

117 if node A receives an ACK packet from LAR
118     Cancel A.legend_timer;

```

A node with the legend will reset the loop by clearing all the visit bits and number of tries in the Global Legend Location Table.

A::RESET-LOOP():

```

119 foreach node entry i in GLT
120     Set GLT[i].v_bit ← false;
121     Set GLT[i].tries ← 0;

```

A.3 LRV Pseudocode

LRV Variables:

hops_since_last_pause

Number of hops the legend has traveled since last pausing

MAX_HOPS_BETWEEN_PAUSES

Number of hops the legend should take before pausing

LRV Subroutines:

Initialize the LRV-specific variables.

INITIALIZATION:

122 // *Initialize GLT*

123 Set $GLT.hops_since_last_pause \leftarrow 0$;

124 INITIALIZE();

A node receiving a legend packet will handle the legend and maintain the number of hops the legend has taken since its last pause if it is the legend's destination, or overhear it if not.

RECEIVE-LEGEND-PACKET:

125 **if** node A receives the legend

126 **if** the legend packet's destination is A

127 Increment $GLT.hops_since_last_pause$;

128 Set $A.send_feedback \leftarrow true$;

129 $A.HANDLE-LEGEND()$;

130 **else** // legend is not destined for A

131 $A.OVERHEAR-LEGEND()$;

The node with the legend will pick the legend's next destination according to the LRV traversal method, and will then send it on. The legend is sent directly. For added reliability, nodes set a timer to resend the legend if they do not hear the legend forwarded or receive an ACK or feedback packet before it expires. Note: LRV will pause the legend instead of sending it on if the legend has traveled enough hops since its last pause.

```

A::SEND-LEGEND-PACKET():
132 if PAUSE_LEGEND AND A.NEED-PAUSE()
133     A.PAUSE-LEGEND();
134     return;
135 // Find legend's next destination
136 A.CHECK-NEIGHBORS();
137 Set A.next_dest ← A.GET-NEXT-DESTINATION-FOR-LRV();
138 if A.next_dest < 0
139     A.PAUSE-LEGEND();
140     return;
141 // Send legend to next destination
142 Send legend packet directly to neighbor A.next_dest;
143 Set A.send_feedback ← false; // previous node can hear legend forwarded
144 Set A.legend_timer(NEIGHBOR_INTERVAL);

```

The LRV traversal method picks the closest neighbor that has never been visited by the legend. If no such neighbor exists, then the least recently visited neighbor is chosen.

```

A::GET-NEXT-DESTINATION-FOR-LRV():
145 if ∃ an entry i such that  $i \in A.NT$  and  $GLT[i].v\_bit = false$ 
146     Set next_dest ← closest i;
147 else
148     Set lrv_neighbor ← -1;
149     Set min_ts ← ∞;
150     foreach entry  $i \in A.NT$ 
151         if  $NT[i] < min\_ts$ 
152             Set min_ts ←  $NT[i]$ ;
153             Set lrv_neighbor ← i;
154     Set next_dest ← lrv_neighbor;
155 return next_dest;

```

A node using the LRV traversal method will pause the legend everytime the legend visits a certain threshold number of nodes since the last pause.

A::NEED-PAUSE():

```

156 if GLT.hops_since_last_pause ≥ MAX_HOPS_BETWEEN_PAUSES
157     Set GLT.hops_since_last_pause ← 0;
158     return true;
159 else
160     return false;

```

A.4 TB Pseudocode

TB Variables:

traceback: Node that originally sent legend to current node in a given loop

TB Subroutines:

Initialize the TB-specific variables.

INITIALIZATION:

```

161 // Initialize LLT
162 foreach node I in the network
163     Set I.traceback ← -1;
164 INITIALIZE();

```


A node receiving a legend packet will handle the legend and maintain the node's traceback link if it is the legend's destination, or overhear it if not.

RECEIVE-LEGEND-PACKET:

```

165 if node A receives the legend
166     if the legend packet's destination is A
167         Set A.send_feedback  $\leftarrow$  true;
168         if GLT[A.id].v_bit = false
169             Set A.traceback  $\leftarrow$  A.prev;
170             A.HANDLE-LEGEND();
171     else // legend is not destined for A
172         A.OVERHEAR-LEGEND();

```

The node with the legend will pick the legend's next destination according to the TB traversal method, and will then send it on. The legend is sent directly. For added reliability, nodes set a timer to resend the legend if they do not hear the legend forwarded or receive an ACK or feedback packet before it expires. Note: a node will pause the legend instead of sending it on if it is time to reset the loop.

A::SEND-LEGEND-PACKET():

```

173 if all nodes in GLT have been visited
174     A.RESET-LOOP();
175     if PAUSE_LEGEND
176         A.PAUSE-LEGEND();
177     return;
178 // Find legend's next destination
179 A.CHECK-NEIGHBORS();
180 Set A.next_dest  $\leftarrow$  A.GET-NEXT-DESTINATION-FOR-TB();
181 if A.next_dest < 0
182     A.PAUSE-LEGEND();
183     return;
184 // Send legend to next destination
185 Send legend packet directly to neighbor A.next_dest;
186 Set A.send_feedback  $\leftarrow$  false; // previous node can hear legend forwarded
187 Set A.legend_timer(NEIGHBOR_INTERVAL);

```

The TB traversal method picks the closest unvisited neighbor. If no such neighbor exists, then the traceback neighbor, being the neighbor that originally sent the current node the legend in the current loop, is chosen. If the traceback neighbor is no longer a neighbor, then the loop is reset and if pausing is being used, the legend is paused, otherwise the algorithm starts over.

```

A::GET-NEXT-DESTINATION-FOR-TB():
188 if  $\exists$  an entry  $i$  such that  $i \in A.NT$  and  $GLT[i].v\_bit = false$ 
189     Set  $next\_dest \leftarrow$  closest  $i$ ;
190 else if  $A.traceback \in A.NT$  // traceback link good
191     Set  $next\_dest \leftarrow A.traceback$ ;
192 else // traceback link broken
193     A.RESET-LOOP();
194     if PAUSE_LEGEND
195         Set  $next\_dest \leftarrow -1$ ;
196     else
197         Set  $next\_dest \leftarrow A.GET-NEXT-DESTINATION-FOR-TB()$ ;
198 return  $next\_dest$ ;

```

A node with the legend will reset the loop by clearing all the visit bits in the Global Location Table (legend). It will also set the current node's traceback link to -1.

```

A::RESET-LOOP():
199 foreach node entry  $i$  in  $GLT$ 
200     Set  $GLT[i].v\_bit \leftarrow false$ ;
201 Set  $A.traceback \leftarrow -1$ ;

```

Appendix B

CORRECTNESS PROOF FOR THE LRV TRAVERSAL METHOD

Given a static, connected network of at least two nodes with a legend that traverses it endlessly following the LRV traversal method, we desire to prove that every node in the network will receive the legend endlessly.

Lemma: If one node, C , in a static network receives the legend endlessly, then all the neighbors of C receive the legend endlessly.

Proof by Contradiction: Assume that A is a mobile node with n neighbors that receives the legend endlessly, and further assume that B is a neighbor of A that is not visited endlessly by the legend. If A maintains its neighbor table in a queue, and that queue is sorted from least-recently-visited time to most-recently-visited time, the LRV algorithm is the act of removing a neighbor from the head of the queue, sending the legend to it, and replacing the neighbor at the end of the queue. All of A 's neighbors, including B , are in this queue. Because the queue is of finite length, n , and A receives the legend endlessly, neighbor B will eventually come to the head of the queue. At this point, a contradiction is reached: A must either send the legend to B or violate the least recently visited algorithm.

Theorem: Every node in a static, connected network is visited endlessly by the legend.

Proof by Induction: Our assumption that the legend traverses endlessly in the network directly implies that there is at least one node that receives the legend endlessly, because if every node was only visited by the legend a finite number of times, then the legend

wouldn't travel endlessly. Given one node, E , that receives the legend endlessly, all of E 's neighbors receive it endlessly by the preceding lemma. If all of E 's neighbors receive the legend endlessly, then all of E 's 2-hop neighbors receive it endlessly, also by the lemma above. Assume that all of E 's k -hop neighbors receive the legend endlessly. If so, then all of E 's $k + 1$ -hop neighbors receive it endlessly, by applying the preceding lemma to each of E 's k -hop neighbors. Therefore, by induction, the theorem is true for all k . Because the network is connected, all the nodes are within some k hops of E , and therefore all of them receive the legend endlessly.