Distributed Token Circulation in Mobile Ad Hoc Networks

Navneet Malpani, Yu Chen, Nitin H. Vaidya and Jennifer L. Welch

Abstract

This paper presents several distributed algorithms that cause a token to continually circulate through all the nodes of a mobile ad hoc network. An important application of such algorithms is to ensure total order of message delivery in a group communication service. Some of the proposed algorithms are aware of, and adapt to changes in, the ad hoc network topology. When using a token circulation algorithm, a *round* is said to complete when every node has been visited at least once. Criteria for comparing the algorithms include the average time required to complete a round, number of bytes sent per round, and number of nodes visited per round. Comparison between the proposed algorithms is performed using simulation results obtained from a detailed simulation model (with ns-2 simulator). We also give a rigorous worst-case analysis of the proposed LR algorithm, which gives the best overall performance in the simulation.

Index Terms

Mobile ad hoc networks, token circulation, distributed system.

An earlier version of this paper appeared as "Distributed Token Circulation in Mobile Ad Hoc Networks," *Proc. 9th International Conference on Network Protocols (ICNP)*, November 2001.

Navneet Malpani is with the Intel Corporation, 505 E. Huntland Dr., Suite 550, Austin, TX 78752

Yu Chen and Jennifer L. Welch are with the department of Computer Science, Texas A&M University, College Station, TX 77843-3112, {ychen, welch}@cs.tamu.edu

Nitin H. Vaidya is with the Electrical and Computer Engineering Department and Coordinated Science Lab, University of Illinois at Urbana-Champaign, Urbana, IL 61801, nhv@uiuc.edu

I. Introduction

This paper presents several distributed algorithms that cause a token to continually circulate through all the nodes of a mobile ad hoc network. An important application of such algorithms is to ensure total order of message delivery in a group communication service [5].

Mobile ad hoc networks are formed by a collection of, potentially mobile, wireless nodes; communication links form and disappear as nodes come into and go out of each other's communication range. Such networks have many practical applications, including home networking, personal area networking, search-and-rescue, and military operations. Wireless networking has received a boost from the development of standards such as IEEE 802.11 and Bluetooth. These wireless technologies can potentially be utilized to implement wireless ad hoc networks.

Mobile ad hoc networking has been an active research area. Much of this activity has focussed on the design of routing and medium access control protocols, since efficiency of these protocols can have a significant impact on performance. However, there has been less work on the development of distributed services for mobile ad hoc networks, such as group communication.

A group communication service forms an important building block for applications in dynamic distributed systems and is useful in many applications that involve collaborations among a group of people (e.g., a whiteboard application). A group communication service can be used by an application designer as a high-level service, allowing the application to remain oblivious to details of the dynamic network environment. The key features of a *group communication* service are: (1) maintaining information regarding group membership (who is in a group and who is not), and (2) letting nodes within a group communicate with each other in an *ordered* manner – many types of orders are useful, including *total* order (wherein all nodes in a group receive all messages in an identical order) [5].

The group communication problem becomes especially difficult in a mobile ad hoc environment wherein links can repeatedly fail and recover. There has been significant research activity on group communication in traditional wired networks (see Section II). The vast body of this research is an indicator of the significance of the group communication service paradigm.

Group communication services have been successfully used in the past as building blocks and abstractions for implementing distributed tasks. Past work on total ordering has yielded several approaches which use a *token* to implement the total order. These algorithms have two flavors:

- As exemplified by the algorithms in [24], [3], totally ordered message delivery is achieved by continually circulating a *token* through all the nodes of the network in a *virtual ring*. The token circulates around the virtual ring carrying a sequence number. When a node receives the token, it assigns sequence numbers (carried with the the token) to its messages, and then multicasts the messages to the group members. The sequence number carried in the token is incremented once for each message sent by the node holding the token. Since the messages are assigned globally unique sequence numbers, total order can be achieved. (Additional mechanisms are needed depending on the desired level of reliability.)
- In the above approach, each message is multicast by the sender node, tagged with a sequence number obtained from the token. An alternative approach [15], [10] is to store the messages in the token itself – since the token visits all nodes in a virtual ring, the messages will eventually reach all the nodes, the order in which messages are added to the token determining the order in which they are delivered to the nodes. Clearly, this approach would result in large tokens (since messages are carried in the token itself).

Both these approaches depend on the existence of a virtual ring in the network. But the prior work has not sufficiently addressed the issue of determining efficient embeddings of rings (around which a token may be circulated) in networks with dynamically changing topology. Past theoretical work on ring embeddings assumes specific target topologies (e.g., [27]); we are not aware of any work on embedding rings in arbitrary dynamic topologies.

In this paper, we will consider mechanisms for finding approximations to a virtual ring that change dynamically as the topology changes and that are efficient according to certain metrics. Since token circulation around a virtual ring is a useful component of many existing group communication mechanisms for wired networks, we will consider ways of improving the performance of such mechanisms in mobile ad hoc networks.

The rest of this paper is organized as follows. Section II summarizes the related work, Section III describes the performance measures that we study, the algorithms are presented in Section IV, the simulation results appear in Section V, an analysis of the LR algorithm is given in Section VI, and Section VII concludes the paper.

II. Related Work

As mentioned earlier, the majority of the past research in the area of ad hoc and packet radio networks has focused on routing and medium access control protocols.

Group communication has been well-studied for static, wired networks, with results ranging from commercial systems to theoretical impossibility proofs. Originally, group communication services were designed for local area networks (e.g., Isis [5]). In considering how to extend such services to large-scale distributed systems, the key new issue is how to handle partitions, which are now much more likely to occur due to failures of links and nodes [2]. Many papers have presented algorithms to deliver messages in various consistent orders within groups, either on top of a layer that deals with membership and view issues, or intertwined with them (e.g., [3], [12], [16], [9], [20]).

Baldi and Ofek [4] compare sending multicast messages over a tree versus a ring embedded in a network for real-time systems. They do not discuss how to find the embeddings. For their comparisons, the ring is obtained by going twice around a spanning tree of the network and ignoring repeated nodes. Their results show that the ring is actually better than the tree in some situations. Their results indicate that it is worth investigating ring embeddings in ad hoc networks.

A few papers have looked at the problems involved in group communication for mobile cellular environments, which have mobile hosts and mobile support station infrastructure. Cho and Birman [8] describes enhancements to the ISIS group communication system to handle mobile clients. An algorithm to ensure message delivery in causal order is described by Prakash et al. [23]. El-Gendy et al. [14] present a model based on the two-tiered approach for providing group communication. The multicast problem for mobile hosts has been studied in [28], [1].

There is less work on token passing in mobile ad hoc networks (one exception is [13]). However, Prakash and Baldoni [22] describe a multi-level architecture for use in various types of mobile environment, including ad hoc networks, and show how a three-round group membership protocol can be used to construct groups, i.e., for group membership. Their paper does not address how to achieve totally ordered message delivery.

[25] presents an algorithm that circulates a *software agent* (analogous to our token) to collect information about network topology. The procedure used by agents to travel through the network is analogous to the LF algorithm described later in this paper. It is worth noting that the proposed LR algorithm (described later) performs better than algorithm LF.

III. Performance Measures

Before presenting our performance measures, we define some terminology used later in the paper. When we say that a token *visits* a node, it implies that the token is received by the group communication service running on that node. On the other hand, when we say a token is *routed* by a node, it means that the network layer at that node simply relayed, or forwarded, the token to another node. A token may be *routed* by a node without *visiting* that node. We define a *round* to be a minimal length execution sequence in which each node is visited at least once.

The following are the performance measures that we will apply to our algorithms.

- **Round length:** The *length* of a round is the number of node visits made by the token in one round. Note that, in general, a given node may be visited multiple times in a round. (Of course, by the definition of a round, there must be at least one node that is visited exactly once within a round.)
- **Message overhead** measured as *number of bytes sent per round:* This overhead measures the overhead due to all packets transmitted to complete one round. If the packet takes multiple hops to reach a destination, the bytes required to transmit the packet on each hop are counted. If any packet is lost and needs to be retransmitted, the retransmissions are counted as well. Similarly, the overhead of sending control packets in the medium access

control protocol is also included in the message overhead (in our simulations, we use the IEEE 802.11 wireless medium access protocol).

- **Time overhead** measured as *time required to complete a round*: The time required to complete a round is the duration of time from when the last node in the previous round is visited until when the last node in the current round is visited.

IV. Token Circulation Algorithms

Let us first consider what would happen if the token circulation algorithm were to ignore the network topology and choose an arbitrary order to visit the nodes. For instance, in a ring consisting of nodes 1, 3, 5, 2, 6, 4, 1, if the nodes are visited in the order $1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, \ldots$, then between any two consecutive visits, the token takes several hops – in this case, although the length of the round is 6 (which is optimal), message overhead is large (since each node visit requires the token to take several hops). On the other hand, visiting the nodes in the order 1,3,5,2,6,4,1,3,5,2,6,4... still results in the optimal length, but lower message overhead. Thus, the latter visit order should be preferred. However, if the visit order is chosen without taking the topology into account, in general, the algorithm will not typically choose the best possible order of visits.

The above example suggests that it is useful to utilize network topology information in determining the order in which nodes are visited. However, knowledge of the network topology, particularly in mobile environments, is expensive to achieve. Therefore, in this paper, we explore token circulation algorithms which use only local neighborhood information, and also consider an algorithm that does not use any topology information.

One simple approach for keeping track of neighbors is by means of "hello" messages (e.g., [6]) – each node periodically broadcasts a hello message, the period being referred to as the *hello interval*. Each node i assumes that a node j is its neighbor if node i has recently received a hello message from j. On the other hand, if node i does not receive a hello from node k for a "hello threshold" number of consecutive hello intervals, then i assumes that k is not its neighbor.

The hello mechanism may be implemented as a part of the group communication service, or alternatively, this information may be obtained by the network layer and made available to the group communication service via a system call.

The algorithms that we have explored are characterized by the following parameters, which control how the node holding the token determines the next node to be visited by the token – the letters in the parentheses in each item below will be used to form the abbreviated names of proposed algorithms, as described later:

- *Local (L) versus Global (G):* In *local* algorithms, the next node to send the token to is chosen from amongst the nodes that are believed to be neighbors of the node possessing token. A *global* algorithm may direct the token towards any node in the network.
- *Recency (R) versus Frequency (F):* In case of *recency* algorithms, the decision on the next recipient of the token is based on how *recently* the nodes have had the token. In case of *frequency* algorithms, the decision is based on how *frequently* the nodes have had the token.
- *Visiting "next" node on route to a desired destination (N):* This variation is only relevant for *global* algorithms. Once a desired destination has been determined by a global algorithm, there are two possibilities: (a) the token is sent directly to the chosen destination – other nodes on the route to this destination will route the token, but the token will not *visit* these intermediate nodes. (b) Alternatively, the current token-holder may send the token to the next node on the route to the chosen destination – effectively, the token will visit all nodes on the route to the chosen destination. In order to implement the second mechanism, the network layer must be able to provide to the application layer the identity of the next node on the route to the desired destination.

Figure 1 shows six algorithms that are obtained using the above variations. The first six algorithms follow the same framework: The token carries with it some "count" information for each node in the system. When a node receives the token, it chooses the next recipient of the token using this count information, updates its own count information in the token, and sends the token to the chosen next recipient. The criteria for choosing the next token recipient and updating the

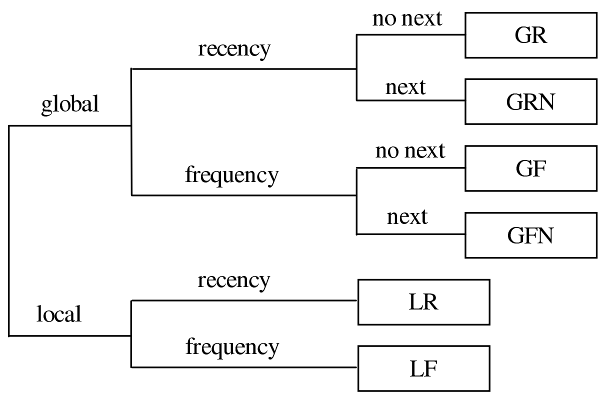

Fig. 1. Decision tree with algorithms at the leaves

count depend on the particular algorithm. We first summarize the possibilities for these two procedures, followed by a more detailed discussion of each algorithm evaluated in the paper.

- *Updating the counts:* The token carries a vector containing a *count* for each node. The *recency* and *frequency* algorithms update the count information in the token differently, as described next.

  For *recency* algorithms, the count for node i (as stored in the token) represents the last "time" when node i was visited. "Time" in this case represents the total number of visits made by the token. The token carries a *time* variable, in addition to the *counts*. The *time* variable is initialized to 0, and then incremented by 1 each time the token visits a node. Consider a node v. Assume that on a particular token visit to node v, the *time* variable in the token is incremented to 15. Then, the *count* for node v (as stored in the token) will be updated to be 15 as well.

  For *frequency* algorithms, the count for node i is the number of times the token has visited node i. Each node maintains a local variable to store the number of token visits made to that node. Each time the token visits a node, the node increments its local counter by 1, and the new value is also stored in the node's *count* in the token. Thus, the token, at all times, contains the number of visits made by the token to each node.

- *Choosing the next token recipient:* For choosing the next recipient, local algorithms are only allowed to consider nodes that are currently believed to be the token-holder's neighbors, whereas global algorithms are allowed to consider all nodes. In most of our algorithms, the next recipient is the node, among those allowed to be considered, with the smallest count value, ties being broken either arbitrarily or by using some other criteria (specified later). The exception to this rule is the global algorithm that visits intermediate nodes, in which case the next destination is actually the *neighbor* of the current token-holder that is on the path to the node with the smallest count.

The token contains one count for each node in the network; each count can potentially grow without bound, although overflow is unlikely with, say, 64 bits allocated per count.

Now we discuss each proposed algorithm individually.

A. Algorithm Local-Frequency (LF)

The Local-Frequency (LF) algorithm keeps track of how many times each node has been visited, and sends the token to the least-frequently visited neighbor of the token-holder. To implement this algorithm, the *count* for each node, as stored in the token, contains the number of past token visits to that node.

Note that, since the token-holder may not have a precise knowledge of its neighbors, occasionally the chosen node may no longer be its neighbor. To protect against the potential loss of the token in such cases, we use a TCP connection to deliver the token. The TCP protocol, running on top of a unicast routing protocol for ad hoc networks, will eventually deliver the token to the intended recipient (provided that the recipient is not partitioned away). This approach is used for all our algorithms. (In our simulations, the Dynamic Source Routing [18] protocol is used for routing in ad hoc networks.)

The following argument proves that if there is no mobility and the topology is connected, then the LF algorithm ensures that every node is visited infinitely often, i.e., there is no starvation. Suppose in contradiction that there is starvation in some execution. Let S be the set of starved nodes and F be the set of non-starved nodes (those that get the token infinitely often). Note that

F cannot be empty; if it were, S would contain all the nodes and the token would have nowhere to be. Consider the situation after the last time that any node in S gets the token. Since the topology graph is connected, there exists a node x that has at least one node in S as a neighbor. Eventually when x gets the token, it sees one of its neighbors, y, in S as its least-frequently visited neighbor and sends the token to y, a contradiction.

However, the LF algorithm has the unfortunate property that the round length can increase without bound in certain network topologies, even if there is no mobility. For example, consider the network shown in Figure 2. Suppose that, initially, the token resides at node 1. Assume that the LF algorithm breaks ties in favor of the neighboring node with the smallest identifier. In this case, it is easy to verify that the length of a round will grow unboundedly with time, when using the LF algorithm. In particular, it can be shown by induction on i that the length of round $2i$ is $3 \cdot 2^i$ and the length of round $2i + 1$ is $3 \cdot 2^{i+1} - 5$.

B. Algorithm Local-Recency (LR)

The Local-Recency (LR) algorithm is similar to LF, except that the least *recently* visited neighbor of the token-holder is chosen as the next recipient of the token. To implement this algorithm, the *count* for each node, as stored in the token, contains the “time” (as defined earlier) when the node was last visited by the token.

A similar argument to that for LF shows that there is no starvation in the case of static connected topologies. In fact, the behavior of LR is much better than that of LF on the static graph in Figure 2 – it ensures a round length that is never more than seven.

In any static connected graph, a round length of at most $2n$, where n is the number of nodes in the graph, can be achieved if the nodes are visited according to a spanning tree of the graph,

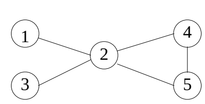

Fig. 2. A network topology for which the LF round length is unbounded

backtracking where necessary. The LR algorithm attempts to improve on this round length by taking advantage of cycles to avoid the backtracking. Our simulation results in Section V show that, in most topologies, the LR algorithm succeeds in improving on the $2n$ bound. However, there do exist graphs (see Section VI) on which the LR algorithm has a round length exponential in n.

C. Global Algorithms

In these algorithms, the token is sent to the node that has been visited the least recently (in algorithm GR) or least frequently (in algorithm GF) among *all* the nodes in the system, not just among the token-holder's neighbors.

When using the GR algorithm, ties will occur only during the first round, and subsequently, the count values for the different nodes will always be distinct. However, ties may potentially occur at any time when using the GF algorithm. If ties are broken arbitrarily when using GF as well, then GF and GR would perform similar to each other. To explore algorithm behavior further, in our simulation of GF, when breaking such ties, we favor nodes that are the token-holder's neighbors. In more detail, preference is given to neighbor nodes over non-neighbor nodes, with remaining ties broken arbitrarily. Thus, this tie-breaker procedure for GF needs the hello mechanism to maintain neighborhood information.

When using both the GR and GF algorithms, the number of nodes visited in each round (i.e., round length) is equal to the number of nodes in the network. It is easy to see that, with the GR algorithm, the token visits nodes in the same order in each round. On the other hand, this is not necessarily true for the GF algorithm. For instance, consider nodes A, B and C that are fully connected. In this case, when using algorithm GF, the token may visit the nodes in the order ABCBACABCBAC... – here, in some rounds the token visits the nodes in order ABC while in other rounds, the order is BAC.

D. Global Algorithms with Next

These algorithms first determine the node with smallest count value from among all nodes in the network. Recall that the counts are included in the token. Then, the token is sent to the neighbor of the token-holder on the route to the node with the smallest count. These algorithms require the ability to query the network layer to determine the neighbor on the route to a given destination. We present simulation results only for GRN, since our experiments showed that GFN on the average performs very poorly. The reason for the poor performance of GFN is that, since the intermediate nodes are also visited, there typically is a node, say x, whose frequency value becomes much higher than the rest. Then the other nodes are visited many times before x is visited again, causing a large round length.

E. Algorithm Iterative Search (IS)

We also considered an algorithm (named *Iterative Search*) that tries to *learn* from the past to improve future performance of the algorithm. Such algorithms can improve performance when the time spent by the network in a given topology increases. Our algorithm tries to find a Hamiltonian path in the network if there exists one.

Pseudo-code for the iterative search algorithm appears in Appendix A. Here we explain the algorithm briefly.

Each node in the network maintains information about the paths that the token takes when it passes through all of its neighbors. The information is in terms of a single distance value for each of its neighbors. The distance value represents a likelihood of the path to the neighbor being part of a Hamiltonian path. The lower the value, more the likelihood. When a node x receives the token from a neighbor y and if x thinks that this edge lies on a Hamiltonian path, then it will record the distance for y as 0, otherwise it will record the distance indicated to it by y (this distance will be some positive value greater than 0). When a node x forwards a token to another of its neighbors, say z, x will record the distance for z as 0 unless it knows that z no longer lies on a Hamiltonian path. If z is not on a Hamiltonian path, then it will compute the distance which x and z will record. If a Hamiltonian path exists in the network, each node will eventually have two of its neighboring distance values equal to 0. One of the neighbors will be the one from which the node receives the token and the other one will be the one to which the node will forward the token.

Initially the distance value of all the neighbors at each node is set to a large value (MAX-VALUE). The node that starts the token passing chooses any one of its neighbors and updates the distance to that neighbor to be 0. The node which receives the token, updates the distance for the node from which it received it, and sees if there is any neighbor which has not been visited. If there exists one then the distance to that node is set to 0 and the token is sent to that node. If there exists no such node and the round is not yet over (i.e., there is at least one unvisited node unreachable from it) the node will increment the distance by 1 for the node from which it received the token and send it to it. The token will have the new distance. The distance value is incremented at each hop by 1 until at least one of the unvisited nodes is reachable. The distance value at each node is updated appropriately. This process continues round after round until a Hamiltonian path is found (if there exists one). In our simulations for static topologies, the algorithm found a Hamiltonian path for all the 50 different scenarios.

A good token circulation algorithm for a mobile ad hoc network should show good performance in the presence of mobility. Existing heuristics for the Hamiltonian circuit problem are designed for static networks and no simulations or proofs show they work well in mobile environments. Although we do not have a proof of correctness for this algorithm, the simulation results are very encouraging. In the mobile case, we simulated two versions of this algorithm, an ideal one in which nodes had perfect knowledge of their neighbors, and a realistic one in which the nodes relied on hello messages to learn their neighbors. Since the IS algorithm did not perform particularly well in the mobile case without perfect knowledge of neighbors, we omit a detailed discussion of the IS algorithm here.

V. Simulation Results

In this section, we present performance evaluation results for the algorithms discussed above. Of these seven algorithms, the results for the GFN algorithm are not shown, for reasons stated in Section IV.

The performance evaluation is done with the ns-2 simulator [26] with CMU extensions [6]. We consider a system consisting of 20 nodes. The transmission range of each node is 250 m and the bandwidth of the channel is 2Mbps. To model mobility of the nodes, we used the *random waypoint* mobility model from [6]. In each mobility scenario generated using this model, the 20 nodes are initially placed in randomly chosen positions in a 1000m $\times$ 300m box. Then, the nodes follow randomly chosen paths. For our experiments, we used node speeds of 6, 12, 18 and 24 m/s and the duration of the simulation was varied inversely with the speed, with the duration for the slowest speed (6m/s) being 50 seconds.

Each algorithm runs as an application on top of TCP, the Dynamic Source Routing (DSR) protocol [18], and IEEE 802.11 MAC. By using TCP as the transport protocol, we ensure that the token does not get lost due to route failures or transmission errors. To facilitate implementation of the GRN algorithm, we augmented DSR such that the token circulation algorithm can obtain the next node on the route to any given destination (the GRN algorithm does *not* need to make use of hello messages for this purpose).

We implemented the “hello” protocol to maintain neighborhood information – this protocol is used for LF, LR, GF and for one of the simulation runs for the Iterative Search algorithm, but not for GR, GRN and the other simulation run for the Iterative Search algorithm. The hello threshold of 3 was used in all simulations. The hello interval was varied as explained later.

In our performance evaluation, we measured average values of the metrics discussed in Section III. Specifically, the metrics are the average time in seconds per round, the average number of bytes transmitted per round (including any hello packets, TCP packets, and medium access control packets), and the average number of nodes visited per round. The following parameters were varied:

- **Hello interval:** In our simulations, hello interval values of 0.1, 0.3, 0.5 and 0.7 second are used. As described earlier, some of our algorithms find it useful to know the neighbors of the node that holds the token. Of course, since the nodes are mobile, it is not possible to maintain perfect knowledge of the neighborhood. The accuracy of this information may affect the overhead of the token circulation mechanisms.

  An issue of interest to us is the frequency with which the hello messages are transmitted. Greater frequency results in greater accuracy in the neighborhood information, but also greater overhead of the hello messages. The issue of hello frequency has been previously studied in the context of unicast routing in ad hoc networks [6], however, here we consider the impact of hello frequency (or hello interval) on the overhead of token circulation algorithms.
- **Speed:** The speed at which the nodes move in a random mobility pattern.

In the following, we first present simulation results in the case when there is no mobility and the topologies are connected. The reason to consider static scenarios is to obtain some intuition about the behavior of the algorithms in the simpler case. Then, we consider the simulation results when there is mobility and discuss how mobility affects the behavior of the algorithms.

A. Static topologies

In this case, we generated many connected random graph topologies for the 20 nodes and simulated the various algorithms. Since the topology is static, the routes, once determined using DSR, do not break during the simulation. Figures 3, 4 and 5 shows the results we obtained. These are plots, for each algorithm, of the number of nodes visited, number of bytes sent, and amount of time elapsed per round, averaged over 50 different scenarios.

For the number of nodes visited per round, the GF and the GR algorithms perform the best; this is of course by definition, since we are only counting the number of visited nodes and not the number of nodes that relay the token. (The GF and GR algorithms pay a cost in terms of bytes and time for having the perfect round length.) Good performance in terms of round length

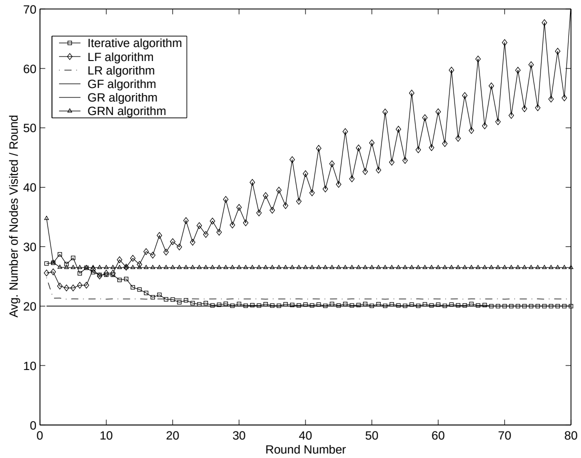

Fig. 3. Average number of nodes visited during each round for 50 static topologies

is exhibited by the Iterative Search algorithm, which converges to the optimal round length after some time, and by the LR algorithm, which within one round converges to close to the optimal round length. In Section IV we mentioned that the LF algorithm had the unfortunate property that the round length can increase without bound in certain topologies. Our simulation results indicate that this property of the LF algorithm occurs in many graphs rather than on a small set of graphs.

For the time and number of bytes per round, our results show the same trends per algorithm as for the round length. The Iterative Search algorithm exhibits the best performance, followed by the LR algorithm. Again the simulation results of the LF algorithm indicate the unfortunate property of unbounded round length in certain topologies. The main difference, as noted above, is that the GF and GR algorithms are no longer optimal.

Among the global algorithms, the GF algorithm performs the best, since ties in this algorithm are broken by preferring the neighboring node. Thus, this global algorithm benefits by also making use of local neighborhood information. The GRN algorithm's performance is also comparable to the local algorithms and the GF algorithm due to the fact that the intermediate nodes are visited.

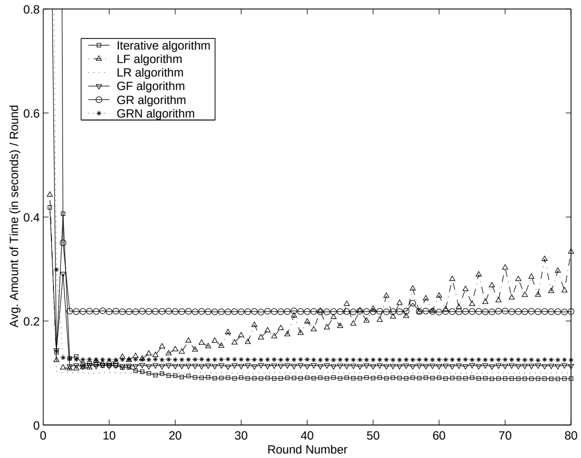

Fig. 4. Average time for each round for 50 static topologies

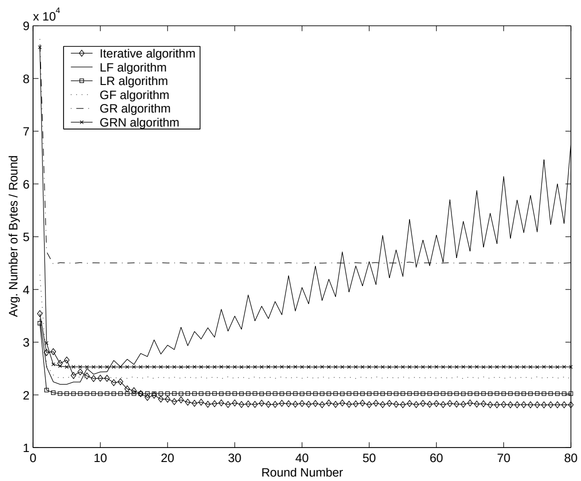

Fig. 5. Average bytes for each round for 50 static topologies

B. Dynamic topologies

For the simulation of dynamic topologies we have two sets of plots:

1) First, we vary the speed of the nodes (6, 12, 18, and 24 m/sec) and find the average amount of time, average number of bytes and the average number of nodes visited per round for all the scenarios with the different speeds.
2) Second, we vary the hello intervals (0.1, 0.3, 0.5, and 0.7 seconds) and find the average amount of time, average number of bytes and the average number of nodes visited per

round for all the scenarios with the different hello intervals.

In both cases, the number of scenarios simulated is 30. Recall that the duration of the simulation was varied inversely with the speed, with the duration for the slowest speed (6 m/s) being 50 seconds.[1]

When we simulated mobility, the LR algorithm continues to perform well in all situations, similar to the static topology cases. We found that the behavior of the other algorithms became somewhat unpredictable in some cases – the simulation results are presented in Figures 6 through 11, and discussed later. We attribute this aspect of our results to three factors:

1) The effect of uncertainty in the topology knowledge due to the hello protocol: Since hello packets are sent at finite intervals, and the hello threshold must necessarily be non-zero, there is a delay before a node learns that a new link has been formed, or an existing link is broken. In addition, since hello messages are sent unreliably, the loss of several hello messages consecutively can lead a node to believe that a link is broken, when it really is not broken.
2) The effect of the TCP timeout intervals when partitions occur: This phenomenon is similar to what has been observed in previous studies of TCP on ad hoc networks [17]. In some of our mobility patterns, partitions occasionally occur, and last for non-negligible intervals of time. Since the global algorithms may choose any node in the network as the destination, the token-holder may choose an unreachable node as the destination – the TCP connection attempting to send the token to this node will timeout, backoff the timeout interval, and retry. Multiple timeouts and retries may occur if the partition lasts for a long interval. Now when eventually the partitions do merge, the TCP timeout intervals may have become very large, and it takes a while for the TCP connection to send the token again, resulting in

[1]For both versions of the Iterative Search algorithm, an additional 20 seconds, during which no topology changes took place, was appended to the simulation time. Since this algorithm relies more on past history than the others, we thought this would give the algorithm a better opportunity to converge; however, it still did not perform particularly well in the realistic case when hello messages were used.

a loss of time. Even in case of local schemes, this situation can occur, because the local topology information is not accurate at all instants of time – specifically, it takes some time for a node to determine that its link with another node is broken. Thus, a node may attempt to send a packet to a node that is actually partitioned away, even when a local scheme is used. However, the likelihood of these events when using local schemes is much lower than when using the global schemes.

3) The chaotic nature of the algorithms themselves. This chaotic nature is easy to see in case of algorithm LF, which relies on information about how *frequently* the neighboring nodes have been visited by the token. In case of LF, small changes in the topology have big effects on the round length. For instance, in Figure 2, we saw that the round length using algorithm LF grows without bound. However, if an edge is added between nodes 1 and 3, then the round length quickly stabilizes to five, which is optimal. As the topology changes with node movement, the system could be switching back and forth between topologies with bounded and unbounded round lengths with respect to LF.

Effect of speed: Figures 6, 7, and 8 show the plots of the average time per round, average number of bytes per round, and average round length versus speed for all the algorithms. For time and bytes, the algorithms from best to worst are ordered: LR, ideal IS, LF, GRN, GF, IS with hello, and GR. More specifically, the average time per round of LR, LF, and ideal IS increases very slowly with speed, while GRN, GF, IS with hello, and GR exhibit increasingly more non-monotonic behavior as speed increases. Similarly, the average number of bytes per round of LR, LF, ideal IS, and GF increases very slowly with speed, GRN and IS with hello exhibit higher and non-monotonic behavior, and GR shows significantly worse behavior that increases with speed. For round length, the ranking is consistent except for the fact that GR and GF by definition are optimal. More specifically, GR and GF have the optimal round length of 20, the round lengths of LR, LF, and ideal IS are all in the low 20's regardless of the speed, GRN's behavior is non-monotonic, and IS with hello shows significantly worse behavior that increases with speed. However, it is worth noting that this metric alone is not adequate to measure the

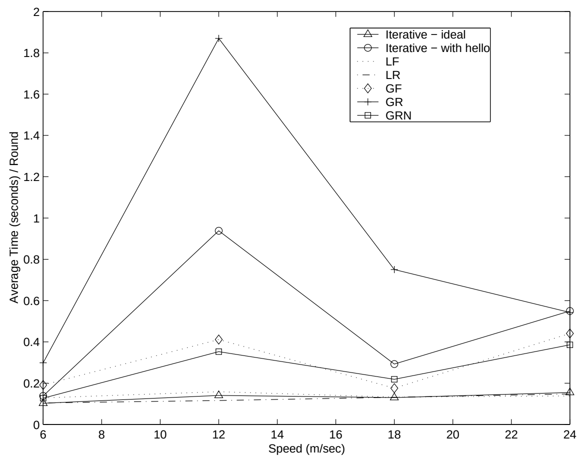

Fig. 6. Average time per round vs. speed for dynamic topologies

algorithm behavior, since GR and GF have higher time and byte overheads. We conjecture that the non-monotonicity exhibited in these plots is due to the factors discussed above.

Effect of hello interval length: Figures 9, 10, and 11 show the plots of the average time per round, average number of bytes per round, and average round length versus hello interval length for all the algorithms. The rankings of the algorithms is essentially the same as that observed when speed was varied.

In general, the larger the hello interval, the fewer the number of bytes that will be sent for hello messages. However, a larger hello interval means that the neighbor information can be more out-of-date, thus possibly incurring more bytes on behalf of the algorithms. This complex interaction contributes to the non-monotonic behavior observed in our simulations.

Our simulations indicate that the LR algorithm gives the best overall performance.

VI. Analysis of Algorithm Local-Recency

In Section V, we saw that the LR algorithm performs well in both static and dynamic network simulations. In particular, the round length is very close to the optimal round length of n. However, our attempts to prove a good upper bound on the behavior of the LR algorithm, in the static case, were not successful. Surprisingly, we are able to exhibit a class of graphs for which

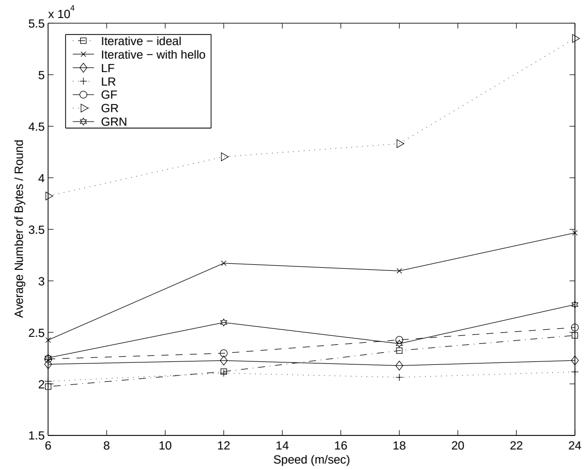

Fig. 7. Average bytes per round vs. speed for dynamic topologies

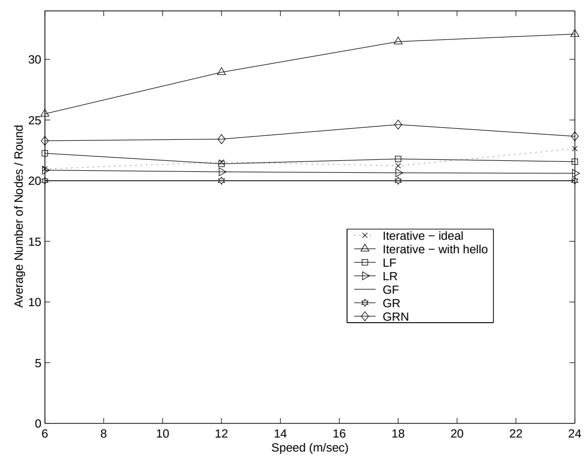

Fig. 8. Average round length vs. speed for dynamic topologies

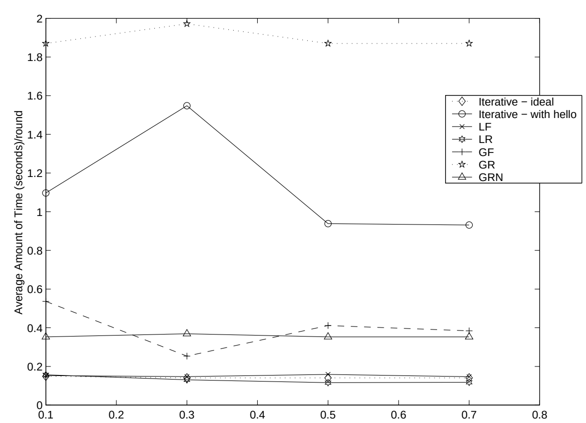

Fig. 9. Average time vs. hello interval

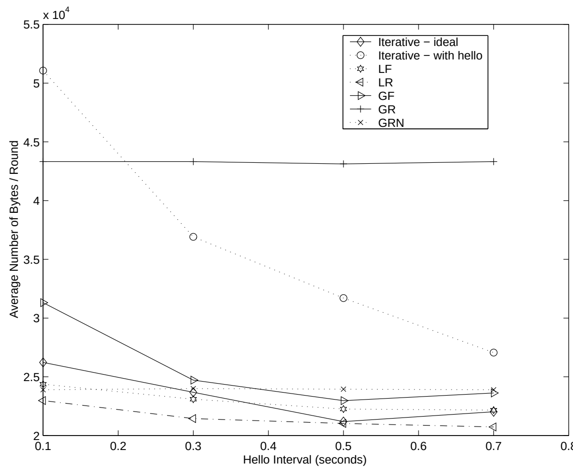

Fig. 10. Average bytes vs. hello interval

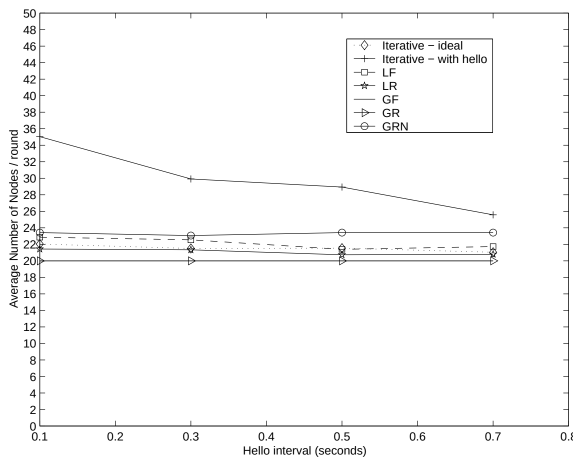

Fig. 11. Average round length vs. hello interval

the worst-case round length of LR is exponential in the number of nodes in the graph.

If the topology graph is directed, then the LR algorithm can have exponential behavior[19]. In particular, on the graph in Figure 12, the LR algorithm has round length $2^{\frac{n+1}{2}} - 1$. This is the same graph on which a random walk requires exponential expected time to visit every node. However, when this graph has undirected edges, the LR round length is only $O(n)$: if $n = 4k-1$, $k \geq 0$, the round length is $\frac{5n-3}{4}$ for all the rounds; if $n = 4k+1$, $k > 0$, the length of the first round is $\frac{5n-1}{4}$ and the length of the subsequent rounds is $\frac{7n-15}{4}$.

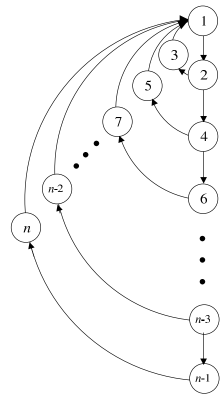

Fig. 12. A directed graph with exponential round length

The rest of this section is devoted to showing exponential upper and lower bounds on the worst-case round length of the LR algorithm when the topology graph is undirected.

Let $G(V,E)$ be a connected undirected graph. In the sequel, the degree of node p is denoted by $degree(p)$, the set of neighbors in $V-S$ of nodes in S is denoted by $N(S)$, the maximum degree of G (the maximum number of neighbors of all the nodes in G) is denoted by Δ and the diameter of G (the number of vertices on the longest shortest path) is denoted by D.

In Section IV-B, we saw that every node is visited infinitely often. Here we prove the round length is $O(n \cdot \Delta^D)$.

Lemma 1: We have the following properties of LR's execution on any graph:

(a) If a node p is visited $degree(p)+1$ times in a segment of the execution, then all the neighbors of p are visited in this segment.

(b) If p is visited no more than k times in an execution, then every neighbor q of p is visited no more than $(k+1)\cdot degree(q)$ times in this execution.

Proof: Let σ be a segment of the execution in which node p is visited by the token $(degree(p)+1)$ times. So the sequence of events in σ is $\langle \cdots, v_1, \cdots, v_2, \cdots, v_{degree(p)+1}, \cdots \rangle$, where each v_i is the event that p gets the token for the ith time.

Suppose p's timestamp is updated to t when it is visited in event v_1. Given a state of the system, we divide the neighbors of p into two subsets, N_1 for the nodes with timestamps larger than t and N_2 for the others. Notice the only way for a node to move from N_2 to N_1 is to be visited by the token. At the beginning of σ, we have $N_2 = N(\{p\})$ and $|N_2| = degree(p)$.

Each time p gets the token, p chooses the next token holder from N_2 until N_2 is empty. Thus $|N_2|$ is decreased by 1 each time p is visited. So it takes at most $degree(p)$ times to empty N_2. Thus when p is visited for the $(degree(p) + 1)$th time, N_2 is empty, that is, every neighbor of p is visited in σ. Thus (a) is proved.

Consider the sequence of visited nodes σ' in an execution in which p is visited k times. The occurrences of p divide σ' into $k+1$ subsequences. By (a), each neighbor q of p occurs no more than $degree(q)$ times in each subsequence, thus in σ', q appears no more than $(k+1) \cdot degree(q)$ times. So (b) is proved. ■

Theorem 2: Every round of LR on any graph has length $O(n \cdot \Delta^D)$.

Proof: Consider any execution α of LR and any round. Suppose p is the last node of this round. Notice that this is the first time p is visited in this round. From (a) in Lemma 1, each neighbor q' of p is visited no more than $degree(q') \leq \Delta$ times in this round. By (b) in Lemma 1, each neighbor q'' of q' is visited no more than $(\Delta + 1) \cdot degree(q'') \leq \Delta^2 + \Delta$ times. Similarly each node q at distance k from p is visited no more than $\Delta^k + \Delta^{k-1} + \cdots + \Delta$ times in this round. Thus the length of the round is no more than

$$\begin{aligned}
& 1+ \\
& (\Delta) \cdot |N(p)|+ \\
& (\Delta + \Delta^2) \cdot |N(N(p))|+ \\
& (\Delta + \Delta^2 + \Delta^3) \cdot |N(N(N(p)))|+ \\
& \cdots \\
& (\Delta + \Delta^2 + \cdots + \Delta^{D-1}) \cdot |N(\cdots N(N(p)) \cdots)| \\
\leq \;& n \cdot (1 + \Delta + \cdots + \Delta^{D-2} + \Delta^{D-1}) \\
= \;& O(n \cdot \Delta^D)
\end{aligned}$$

■

Theorem 2 gives a fairly loose bound. It is possible that no graph actually exhibits such bad behavior. We next show a family of graphs on which the LR round length is exponential.

A graph is said to have a *fixed point round* if the execution of the LR algorithm on that graph, when considered as a sequence of rounds $\rho_1, \rho_2, \ldots$, has the property that for some $k \geq 1$, $\rho_k = \rho_{k+1} = \ldots$. Furthermore, the sequence ρ_k is said to be the fixed point round.

Let S be a graph on the set of nodes $\{1, \ldots, m\}$, we define S to satisfy the condition FP if there are two nodes r and p in S and an integer $t > 1$, such that if the token starts at r, S has a fixed point round σ satisfying:

- FP_1: The last node of σ is r.
- FP_2: p occurs t times in σ, and every neighbor of p occurs between every two consecutive occurrences of p in σ, and either before the first occurrence of p or after the last occurrence of p in σ.

The reason for these conditions will be explained shortly.

Figure 13 depicts a unit disk graph that satisfies FP with $r = 16$, $p = 27$, and $t = 2$. If the token is started at 16, the fixed point round is 13, 12, 1, 2, **3**, **27**, **10**, 9, 8, 26, 4, 5, 6, 7, 8, 9, 23, 22, 21, 20, 19, 18, 17, 5, 4, **3**, 2, 24, 25, 11, **10**, **27**, **3**, 4, 26, 8, 7, 6, 5, 17, 18, 19, 20, 21, 22, 23, 9, **10**, 11, 25, 24, 2, 1, 12, 15, 14, 13, **16** and has length 58. The bold nodes in this sequence are r, p and the neighbors of p.

We now consider the construction of a family of graphs, G_k, $k = 1, 2, \cdots$. Informally, G_k consists of k copies of a graph S satisfying FP, hooked together in series, with node r in one copy connected to node p in the next copy.

More formally, for $i \geq 1$, define graph S_i to be isomorphic to S, with each node j, $1 \leq j \leq m$, in S replaced with node $m \cdot (i-1)+j$ in S_i. Define node $r_i = r+(i-1) \cdot m$ and $p_i = p+(i-1) \cdot m$. Note that each S_i satisfies FP with respect to p_i, r_i and t, and the fixed point round of S_i is σ_i, the result of replacing each occurrence of node j in σ with node $m \cdot (i-1) + j$.

For $k \geq 1$, we define G_k to be $\bigcup_{i=1}^{k} S_i$ together with the additional edges (r_i, p_{i+1}), $1 \leq i \leq k-1$ (see Figure 14). The number of nodes in G_k is $n = k \cdot |S| = k \cdot m$, where m is the number of nodes in S.

For each $k \geq 1$, consider the execution of the LR algorithm on graph G_k in which the token

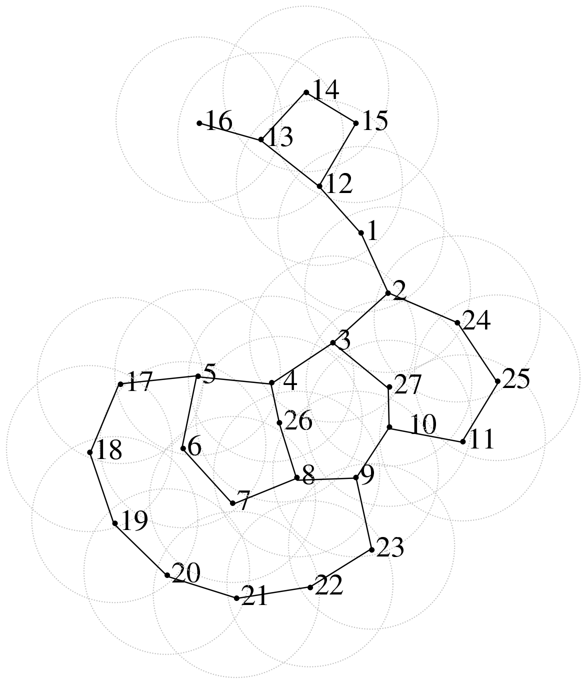

Fig. 13. A unit disk graph satisfying FP

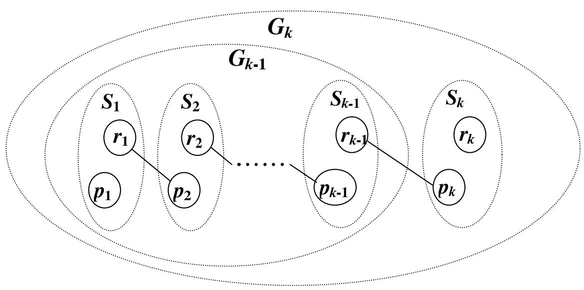

Fig. 14. Construction of G_k

starts at node r_k. Let φ_k denote the resulting fixed point round, if one exists. In the statement of the next theorem, we use the notation $\alpha(x \to \beta)$ to represent the result of replacing every occurrence of x in the sequence α with the sequence β. For example, if $\alpha = \langle a, b, c, a \rangle$, $x = a$ and $\beta = \langle b, c \rangle$, then $\alpha(x \to \beta) = \alpha(a \to \langle b, c \rangle) = \langle b, c, b, c, b, c \rangle$.

Theorem 3: For all $k \geq 1$, if the token starts at r_k, the fixed point round φ_k for LR on graph G_k ending at node r_k exists. Furthermore the round sequence is $\varphi_1 = \sigma$ and $\varphi_k = \sigma_k(p_k \to \langle p_k, r_{k-1}, \varphi_{k-1}, p_k \rangle)$, for $k \geq 2$.

Proof: For the base case, G_1 equals S, and thus the theorem is true. Assume the theorem is true for $k-1$ and show it holds for k.

First notice that G_k can be viewed as G_{k-1} connected to S_k by the single edge (r_{k-1}, p_k). Thus if we eliminate successively repeating nodes, any sequence of visited nodes of LR on G_k starting at r_k is a sequence of visited nodes of LR on G_{k-1} starting at r_{k-1} when restricted to the nodes of G_{k-1}. In Section 4.1, we saw that each node in the network is visited infinitely often by LR, so this sequence of LR on G_{k-1} is infinite. Thus there exist fixed point rounds on G_{k-1} ending at r_{k-1} when restricted to the nodes of G_{k-1} by the inductive hypothesis.

Similarly if we eliminate successively repeating nodes, any sequence of visited nodes of LR on G_k starting at r_k is a sequence of visited nodes of LR on S_k starting at r_k when restricted to the nodes of S_k, which is infinite. So by condition FP, there exist fixed point rounds on S_k satifying FP_1 and FP_2 when restricted to the nodes of S_k by the property C of S_k. By FP_2 we see that in such fixed point rounds on S_k, between any two times p is visited, all of p's neighbors are visited, that is, starting from the second occurrence of p in the fixed point rounds, each time p gets the token, all the neighbors of p have been visited since the last time p gets the token.

In the following, we consider the execution of LR on G_k in which the sequence on G_{k-1} is the fixed point rounds, and the sequence on S_k is the sequence after the second occurrence of p in the fixed point rounds.

Whenever the token comes to p_k from some node in S_k, by the condition FP_2 on S_k, every neighbor of p_k has been visited more recently than r_{k-1}, so p_k forwards the token to r_{k-1}, which passes the token to the nodes in G_{k-1}.

The inductive hypothesis implies that r_{k-1} occurs only once in φ_{k-1}, namely at the end. Thus, by the time the token returns to r_{k-1}, it has visited every node except r_{k-1} in G_{k-1} along φ_{k-1}, so the token is then sent to p_k. This fact shows $\varphi_k = \sigma_k(p_k \to \langle p_k, r_{k-1}, \varphi_{k-1}, p_k \rangle)$ is true for G_k. Since φ_{k-1} and σ_k are fixed by the inductive hypothesis and S_k's property, φ_k is fixed. ■

The next theorem gives a tight asymptotic bound on the length of the fixed point round of

G_k. Recall that m is the number of nodes in each S_i, so $n = m \cdot k$ is the number of nodes in G_k. Thus the bound is exponential in the number of nodes.

Theorem 4: $|\varphi_k| = \Theta(t^{\frac{n}{m}})$.

Proof: For $k = 1$, $|\varphi_1| = |\sigma|$. For $k > 1$, $|\varphi_k| = |\sigma| + t \cdot (|\varphi_{k-1}| + 2)$ by Theorem 3, since each of the t occurrences of p_k in σ_k, which has the same length as σ, is replaced with $\langle p_k, r_{k-1}, \varphi_{k-1}, p_k\rangle$. The solution to this recurrence is $|\varphi_k| = (|\sigma| + 2t)(1 + t + t^2 + \ldots + t^{k-2}) + |\sigma| \cdot t^{k-1}$, $k > 1$. Thus the length of the fixed point round of G_k is $\Theta(t^k) = \Theta(t^{\frac{n}{m}})$. ■

The graph S in Figure 13 satisfies conditions FP_1 and FP_2 with $m = 27$, $t = 2$, and $|\sigma| = 58$. If G_k is constructed from this graph, then by Theorem 4, $|\varphi_k| = \Theta(t^k) = \Theta(t^{\frac{n}{m}}) = \Theta(2^{\frac{n}{27}})$. Let us compare this bound to the one obtained in Theorem 2, which is $O(n \cdot \Delta^D)$, where Δ is the maximum degree and D is the diameter. Since the maximum degree of G_k is 3 and the diameter of G_k is $7k + 5$, this bound becomes $O(n \cdot 3^{7k+5}) = O(n \cdot 3^{\frac{7}{27}n+5})$. Thus we can see that the general upper bound is a large over-estimate for this family of graphs.

VII. Conclusion

We have studied the problem of circulating a token throughout all the nodes of a mobile ad hoc network, a problem of interest for implementing totally ordered message delivery in a group communication service. We have described several distributed algorithms for this problem and compared them by simulation. The overall best algorithm, according to the metrics that we measured, was the Iterative Search algorithm in the static case and the LR (Local-Recency) algorithm in the dynamic case. This difference in performance in the static and the dynamic case clearly shows us that some algorithms that perform well in static networks are not well suited for mobility.

Work is in progress to identify characteristics of graphs on which LR has linear round length; the counter-example graphs found so far have a complex recursive construction. We do not have an analysis that shows properties of the algorithm in the presence of mobility — an adversarial mobility pattern can potentially prevent a node from receiving a token. Theoretical analysis of the protocol in the presence of mobiilty is a topic for future work (incidentally, as yet there is little, if any, theoretical complexity analysis of algorithms for mobile scenarios in ad hoc networks).

Additional work is needed to integrate token circulation as described here with the mechanisms of a complete group communication service. On the theoretical side, if upper bounds on the overhead of these algorithms and/or lower bounds on the achievable overhead could be obtained, they could be compared with simulated performance of the proposed approaches.

Finally, the algorithms presented here are not tolerant of token loss (due to node failure or message loss) or of long-term partitions of the ad hoc network. One way to handle token loss is with a self-stabilizing algorithm. Self-stabilizing token circulation algorithms have been developed for static networks [11], [21] and for mobile networks [13], [7]. Integrating self-stabilization into the algorithm of this paper is left for future work. Handling partitions is part of the purpose of the membership maintenance aspect of a group communication service and often depends on the application semantics.

Acknowledgments: The research reported here was conducted while all authors were at Texas A&M University and was supported in part by National Science Foundation grants CCR-9972235 and CCR-0098305. We thank Kera Alexander for asking the question, Jianer Chen, David Karger, Idit Keidar and Nancy Lynch for helpful conversations, and Gavin Holland for assistance with the software.

APPENDIX

Variables in *token*:

- *visited*[]: array of booleans, one entry per node, indicates whether that node has been visited yet in the current round; initially all false
- *counting*: boolean indicating whether counts are to be calculated; initially false
- *count*: integer; initially 0

Local variables of node $i, 0 \leq i < n$:

- *neighborCount*[]: array of counts for i's neighbors; initially all $\perp$
- *curSender*: id of node (other than self) that sent the token to i most recently
- *prevSender*: id of node (other than self) that sent the token to i previously

---

1. when node i receives *token* from node j:
2. *neighborCount*[j] := *token.count*
3. **if** (*token.visited*[i] = **false**) **then**
4. *token.visited*[i] := **true**
5. **if** ($i \neq j$) **then**

```
        prevSender := curSender; curSender := j;
      endif
    endif
    next := getNext()
    neighborCount[next] := token.count
    send token to next

function getNext() returns node id
  N := set of ids of all neighbors of i
  UV := {k : token.visited[k] = false}
  UVN := N ∩ UV // unvisited neighbors
  if (|UV| = 0) then // all nodes are visited
    token.visited[k] := false, 0 ≤ k < n
    token.counting := false
    token.count := 0
    return i
  else if (|UVN| = 0) then // backtrack
    token.counting := true
    token.count := 1 + max({token.count}∪
                   {neighborCount [k] : k ∈ N})
    return curSender
  else // there is an unvisited neighbor
    token.counting := false
    token.count := 0
    if (∃k ∈ N s.t. neighborCount[k] = ⊥) then
      return any such k
    else
      m := min({neighborCount[k] : k ∈ UVN})
      S := {k ∈ UVN : neighborCount[i] = m}
      if prevSender ∈ S then return prevSender
      else return any k ∈ S endif
    endif
  endif
```

REFERENCES

[1] A. Acharya and B. R. Badrinath. A framework for delivering multicast messages in networks with mobile hosts. *ACM – Baltzer Journal on Mobile Networks and Applications*, 1:199–219, 1996.

[2] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Membership algorithms for multicast communication groups. In *Proc. International Workshop on Distributed Algorithms*. Haifa, Israel, 1992, pp. 292-312.

[3] Y. Amir, L. Moser, D. Agrawal, and P. Ciarfella. Fast message ordering and membership using a logical token-passing ring. In *Proc. of the 13th IEEE International Conference on Distributed Computing Systems*. Pittsburgh, PA, 1993, pp. 551-560.

[4] M. Baldi and Y. Ofek. Ring versus tree embedding for real-time group multicast. In *Proc. IEEE INFOCOM '99*. New York, NY, March 1999, pp. 27-45.

[5] K. P. Birman and R. V. Renesse. *Reliable Distributed Computing with the ISIS Toolkit*. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[6] J. Broch, D. A. Maltz, D. B. Johnson, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *ACM/IEEE Int. Conf. on Mobile Computing and Networking*. Dallas, TX, October 1998, pp. 85-97.

[7] Y. Chen and J. L. Welch. Self-stabilizing mutual exclusion using tokens in mobile ad hoc networks. In *the 6th Annual International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM'2002)*, 2002.

[8] K. Cho and K. P. Birman. A group communication approach for mobile computing. Technical Report TR94-1424, Department of Computer Science, Cornell University, Ithaca, NY, May 1994.

[9] F. Cristian. Synchronous and asynchronous group communication. *Communications of the ACM*, 39:88–97, 1996.

[10] F. Cristian and F. Schmuck. Agreeing on processor group membership in asynchronous distributed systems. Technical Report CSE95-428, Department of Computer Science, University of California at San Diego, 1995.

[11] A. Datta, C. Johnen, F. Petit, and V. Villain. Self-stabilizing depth-first token circulation in arbitrary rooted networks. In *SIROCCO'98, The 5th International Colloquium On Structural Information and Communication Complexity Proceedings*, 1998.

[12] D. Dolev, D. Malki, and R. Strong. An asynchronous membership protocol that tolerates partitions. Technical Report CS94-6, Institute of Computer Science, Hebrew University of Jerusalem, 1994.

[13] S. Dolev, E. Schiller, and J. Welch. Random walk for self-stabilizing group communication in ad-hoc networks. *21st IEEE Symposium on Reliable Distributed Systems*, October 2002.

[14] M. A. El-Gendy, H. Baraka, and A. H. Fahmy. Migrating group communication protocols to networks with mobile hosts. In *Proc. IEEE Midwest Symposium on Systems and Circuits*. University of Notre Dame, Notre Dame, IN, 1998, pp. 7-12.

[15] A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partitionable group communication service. In *Proc. of the 16th Annual ACM Symposium on Principles of Distributed Computing*. Santa Barbara, CA, 1997, pp. 53-71.

[16] R. Friedman and R. V. Renesse. Strong and weak virtual synchrony in Horus. Technical Report TR95-1537, Department of Computer Science, Cornell University, Ithaca NY, 1995.

[17] G. Holland and N. H. Vaidya. Analysis of TCP performance over mobile ad hoc networks. In *Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*. Seattle, WA, 1999, pp. 219-230.

[18] D. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*. Kluwer Academic Publishers, Boston, 1994, pp. 153-181.

[19] D. Karger. personal communication, 2001.

[20] I. Keidar and D. Dolev. Efficient message ordering in dynamic networks. In *Proc. ACM Symposium on Principles of Distributed Computing*. Philadelphia, PA, 1996, pp. 68-76.

[21] F. Petit and V. Villain. Color optimal self-stabilizing depth-first token circulation for asynchronous message-passing systems. In *Proc. of the ISCA 10th Conf. Parallel and Distributed Computing Systems*, Oct, 1997.

[22] R. Prakash and R. Baldoni. Architecture for group communication in mobile systems. In *Proc. IEEE Symp. on Reliable Distributed Systems*. Purdue University, Lafayette, IN, October 1998, pp. 235–242.

[23] R. Prakash, M. Raynal, and M. Singhal. An efficient causal ordering algorithm for mobile computing environments. In *Proc. of 16th ICDCS*. Hong Kong, 1996, pp. 744-751.

[24] B. Rajagopalan and P. McKinley. A token-based protocol for reliable, ordered multicast communication. In *Proceedings of the 8th IEEE Symposium on Reliable Distributed Systems*. Seatle, WA, October 1989, pp. 84-93.

[25] R. RoyChoudhury, S. Bandyopadhyay, and K. Paul. A distributed mechanism for topology discovery in ad hoc wireless networks using mobile agents. In *Workshop on Mobile Ad Hoc Networking and Computing (Mobi Hoc 2000)*, 2000.

[26] V. P. Team. *The network simulator – ns-2*. VINT Project Team, Available at http://www.isi.edu/nsnam/ns/, November 2000.

[27] Y. C. Tseng, S. H. Chang, and J. P. Sheu. Fault-tolerant ring embedding in a star graph with both link and node failures. *IEEE Trans. Parallel and Distributed Systems*, 8:1185–1195, 1997.

[28] G. Xylomenos and G. C. Polyzos. IP multicast for mobile hosts. *IEEE Communications Magazine*, 35:54–58, 1997.