DESIGN AND ANALYSIS OF DISTRIBUTED PRIMITIVES FOR MOBILE AD HOC NETWORKS

A Dissertation

by

YU CHEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2005

Major Subject: Computer Science

DESIGN AND ANALYSIS OF DISTRIBUTED PRIMITIVES FOR MOBILE AD HOC NETWORKS

A Dissertation

by

YU CHEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Jennifer L. Welch
Committee Members,	Riccardo Bettati
	Jianer Chen
	Weiping Shi
Head of Department,	Valerie E. Taylor

August 2005

Major Subject: Computer Science

ABSTRACT

Design and Analysis of Distributed Primitives for Mobile Ad Hoc Networks.

(August 2005)

Yu Chen, B. Eng.; M.S., Zhejiang University, P. R. China

Chair of Advisory Committee: Dr. Jennifer L. Welch

This dissertation focuses on the design and analysis of distributed primitives for mobile ad hoc networks, in which mobile hosts are free to move arbitrarily. Arbitrary mobility adds unpredictability to the topology changes experienced by the network, which poses a serious challenge for the design and analysis of reliable protocols. In this work, three different approaches are used to handle mobility. The first part of the dissertation employs the simple technique of ignoring the mobility and showing a lower bound for the static case, which also holds in the mobile case. In particular, a lower bound on the worst-case running time of a previously known token circulation algorithm is proved. In the second part of the dissertation, a self-stabilizing mutual exclusion algorithm is proposed for mobile ad hoc networks, which is based on dynamic virtual rings formed by circulating tokens. The difficulties resulting from mobility are dealt with in the analysis by showing which properties hold for several kinds of mobile behavior; in particular, it is shown that mutual exclusion always holds and different levels of progress hold depending on how the mobility affects the token circulation. The third part of the dissertation presents two broadcasting protocols which propagate a message from a source node to all of the nodes in the network. Instead of relying on the frequently changing topology, the protocols depend on a less frequently changing and more stable characteristic — the distribution of mobile hosts. Constraints on distribution and mobility of mobile nodes are given which guarantee that all the nodes receive the broadcast data.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Dr. Jennifer L. Welch, for her inspiring and encouraging guidance in the exciting field of distributed computing and her invaluable comments during the work on this dissertation. I truly appreciate her patience, encouragement, advice and research support throughout my Ph.D study. The methodology and philosophy that I learned in my research will definitely benefit my career for life.

I want to express my gratitude to the members of my advisory committee, Dr. Riccardo Bettati, Dr. Jianer Chen, and Dr. Weiping Shi, for their valuable comments and earnest help. Special thanks to Dr. Nitin H. Vaidya for many helpful suggestions.

I also thank the fellow students in my research group, Guangtong Cao, Cheng Shao, Nicholas Neuman, Sangeeta Bhattacharya, Rajan Chandra and Vijay Balasubramanian for their collaborations.

Finally, I would like to thank my parents and my family. I could not have gone through this long journey without their constant love and support. I love them so much.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Mobile ad hoc networks consist of mobile hosts, which are free to move arbitrarily. The communication between the mobile hosts depends on their positions and transmission ranges, so the communication topology may change with time as the hosts move into and go out of each other's transmission range. The technology of mobile ad hoc networking is becoming increasingly prevalent and it has been an active research area.

A major obstacle in design and analysis of distributed primitives for mobile ad hoc networks is the movement of mobile nodes. Arbitrary mobility adds unpredictability to the topology changes, which is a big challenge in the analysis of protocols' performance and design of reliable protocols. Due to the complication introduced by arbitrary topology changes, there is little theoretical analysis of algorithms for mobile scenarios. The most common way to examine a protocol's behavior is simulation, which should match as closely as possible the reality. However, the results in [1] indicate that significant divergences exist between the most popular simulators, including OPNET Modeler[2], NS-2 [3] and GloMoSim [4]. Thus simulation results alone might not give enough meaningful information about protocols' performance — sometimes theoretical analysis is required to complement simulation. Furthermore, unpredictable topology changes prevent most protocols from providing fully reliable service. Most of the protocols designed for mobile ad hoc networks use a best-effort strategy: protocols try their best to provide services, but service qualities are not guaranteed. In our work, we use three approaches to handle the complications introduced by mobility.

Our first approach is to reduce the problem in mobile scenarios to a simpler and solv-

The journal model is *IEEE Transactions on Automatic Control.*

able case, in which the analysis can still provide meaningful information. In chapter IV we focus on a token circulation algorithm, LR, which shows quite good performance in terms of *round length* in simulations [5], where round length is the number of node visits made by the token in order to visit all the nodes of the network. The goal is to provide an idea of how bad the performance of LR could be in mobile scenarios. Our analysis is done for static cases (the results appeared in [6]). We give a loose upper bound and a rigorous worst-case analysis on the round length. Since the performance cannot be better with mobility, this analysis gives us an idea of the algorithm's performance in the worst case in the presence of mobility: in particular, the worst case behavior with mobility cannot be better than the worst case for static networks.

Our second approach is to guarantee some aspects of the service unconditionally and provide different level of the others under different mobility conditions [7] [8]. In our work on self-stabilizing mutual exclusion (chapter V), the mutual exclusion property is always guaranteed without constraints on mobility and different levels of progress are guaranteed under different levels of constraints on mobility. A preliminary version of this algorithm appeared in [7] and the journal version has been accepted by [9].

Our third approach to handle mobility is to eliminate, based on the specific conditions of the network, as much as possible the impact on the protocols of the changing communication topology. Usually protocols designed for mobile networks depend on the communication topology, and its unpredictability prevents them from providing reliable services. In chapter VI, we focus on broadcasting problem in dense mobile networks, in which mobile nodes keep moving but their distribution is fairly stable. Instead of relying on the frequently changing communication topology, our algorithms depend on a less frequently changing and more stable characteristic — the distribution of mobile nodes. We also provided specific constraints on the distribution and mobility of mobile nodes to guarantee that all the nodes receive the broadcast data.

CHAPTER II

RELATED WORK

The technology of mobile ad hoc networking is becoming increasingly prevalent and it has been an active research area. Much of the work in this area has focused on routing and medium access control protocols ([10], [11], [12], [13], [14]). Past work on distributed services focused on non-fault-tolerant algorithms (e.g. leader election [15][16], token circulation [5] and mutual exclusion [17]). A survey of distributed algorithms for mobile ad hoc networks can be found in [8]. In this chapter, we provide the related work on token circulation, mutual exclusion and broadcasting respectively.

A. Related Work on Token Circulation

Token circulation can be used to implement totally ordered message delivery in a group — all nodes in a group receive all messages in an identical order. One approach is to assign each message a globally unique sequence number, which can be maintained by a token. In [18, 19], a token carries a sequence number and is circulated through all the nodes. When a node receives the token, it gets the sequence number from the token and assigns it to the pending message which is sent to the group members; the sequence number in the token is incremented by one. Total order can also be achieved by storing messages in the token — the order in which messages are added to the token determines the order in which they are delivered to the nodes [20, 21].

Several distributed token circulation algorithms for mobile ad hoc networks are studied in [5], in which the LR algorithm is introduced. Some of the proposed algorithms are aware of, and adapt to changes in, the ad hoc network topology. Comparison between the proposed algorithms is performed using simulation results obtained from a detailed simulation model (with ns-2 simulator). Our analysis results focus on the LR algorithm.

The results were added to [5] and can be found in its journal version [6].

B. Related Work on Mutual Exclusion

Mutual exclusion can be solved by circulating a token throughout the system in an appropriate way; when a processor has the token, it can enter the critical section. If the token circulation algorithm ensures that no more than one token exists in the system and every processor gets it infinitely often, clearly this simple approach to solving mutual exclusion is correct. Self-stabilizing mutual exclusion [22] and token circulation algorithms [23][24] have been designed for static networks. However, after they converge, they do not guarantee mutual exclusion in the presence of ongoing topology changes.

One paper on self-stabilization for mobile ad hoc networks is [25], in which a self-stabilizing group communication using random walk is proposed. In contrast to the *probabilistic* protocol in [25], our algorithm is *deterministic*. Both algorithms use tokens or agents to control processors' behavior, but they differ in many aspects, including how the tokens or agents are forwarded, how membership is maintained and how to handle mobility. Other work on self-stabilization for mobile situations includes a "weakly" self-stabilizing leader election algorithm in [26]. Self-stabilization can be used to handle the topology changes in dynamic distributed system ([27], [28], [29]). A comprehensive bibliography on self-stabilization is given in [30]. A survey of fault-tolerant algorithms for mobile ad hoc networks can be found in [8].

A preliminary version of our algorithm appears in [7]. Simultaneously and independently, [31] presented a non-fault-tolerant mutual exclusion algorithm in which the ring is also built dynamically according to the current physical proximity of nodes. The journal version has been accepted by [9].

C. Related Work on Broadcasting

Traditional broadcast algorithms for radio networks (e.g., [32]) usually rely on knowledge of the network topology. Mobility is not considered and the algorithms are not adaptive to topology changes. The best known broadcasting protocol [32] for undirected radio networks in which nodes do not transmit until they receive a message has time complexity $O(n \log n)$, where n is the number of nodes. If we apply these algorithms in dense networks, the complexity will be very high due to the large number of nodes. A lower bound of $\Omega\left((n \log n)/(\log \frac{n}{D})\right)$ on time complexity is also provided in [32], where D is the diameter of the network. Denoting the area of the network by $\mathcal{A}$ and the transmission range by R, our approaches achieve $O\left(\mathcal{A}/R^2\right)$ time complexity for a special case group of networks with certain properties.

Broadcasting protocols [33, 34, 35, 36, 37, 38] designed for mobile ad hoc networks have been heavily studied recently. Their performances are evaluated by simulations and little theoretical analysis is provided. A comparison of broadcasting techniques for mobile ad hoc networks is presented in [39]. Location information is used in area-based broadcasting in [33]: an intermediate node retransmits only when significant additional area could be reached. But applying this method in a dense network will result in many collisions since nodes at the boundary of the sender's transmission distance will retransmit simultaneously.

Position information has been used in routing protocols for mobile ad hoc networks; examples are [40, 41, 42, 43, 44, 45, 46, 47, 48]. Most of the existing position-based routing algorithms forward the packet on *straight lines* between source and destination: when a node receives a packet, it selects a neighbor [41] or all neighbors [43, 44] in the general direction of the destination to forward the packet. But routing along the shortest path is not always the best option since partitions may occur due to battery overuse along these popular shortest paths. Another problem introduced in this method is called *local maximum*

— in some situations forwarding along straight lines is not possible due to obstacles or holes in distribution of mobile nodes. In order to handle the *local maximum* problem, face routing (e.g., [47]) has been proposed to route packets around the obstacles or holes. Non-straight trajectories are considered in trajectory-based routing [48], in which packets follow a trajectory established at the source, and each intermediate node takes a greedy decision of the next hop based on local position information. Since the trajectory is established before packet propagation, knowledge about the distribution of mobile nodes and the battery states of nodes is important to design an efficient trajectory. Discussion on how to handle the dynamic situation is provided in [48]. A survey of position-based routing in mobile ad hoc networks can be found in [14]. The most important differences between our approach and the existing work are that, in our approach the location-based paths are decided *online* during message forwarding and *no knowledge about the distribution of mobile nodes is required a priori*. These properties are desirable in mobile ad hoc networks, in which global knowledge might not be available and the system keeps changing.

CHAPTER III

SYSTEM MODEL

Mobile ad hoc networks consist of computing entities or mobile hosts, which are free to move arbitrarily. Mobile ad hoc networks have several salient characteristics, such as dynamic topologies, bandwidth-constrained links and limited power supplies. Mobile hosts are equipped with wireless transmitters and receivers, and the communication between the mobile hosts depends on their positions and transmission ranges. The communication ability between pairs of mobile hosts can be represented by a directed graph: each vertex v_i represents a mobile processor p_i and there is a directed link from vertex v_i to vertex v_j if and only if mobile processor p_j is within the transmission range of mobile processor p_i.

An important characteristic of the wireless medium is that if more than one neighbor of a node are transmitting at the same time, then a collision occurs. Different systems have different ability to detect collisions. The wireless communication can be synchronous or asynchronous. In our work on broadcasting problem (Chapter VI), we assume nodes can distinguish between the background noise and the interference noise, that is, a mechanism is available to detect collision. We also assume synchronous communication, in which the communication is structured into time-slots. Details of the system model of our broadcasting protocols is presented in Chapter VI.

Basic communication service between adjacent processors can be provided on the top of the physical layer. In our work on token circulation and mutual exclusion (Chapter IV and Chapter V), the distributed services assume the existence of a basic communication service between adjacent processors and the physical details of the wireless communication are hidden from the distributed protocols. In Chapter V, we list more specific assumptions required for our mutual exclusion algorithm.

We use the message passing model, in which each processor has an incoming queue.

Messages in transit are modeled as being in the incoming queues of the receiving processors. Given a mobile ad hoc network, a *configuration* of the network at some point in time can be described by the local state of each processor, the state of the incoming queue of each processor and the communication topology. There are two kinds of *events* in mobile ad hoc networks: *local computation events* and *topology changes*. A local computation event is initiated by the receipt of a message, the receipt of a request from the application, or a timeout expiring if timers are available. A local computation event changes the state of the processor at which it occurs, and enqueues messages in the incoming queues of neighboring processors. A topology change event changes the topology. An *execution* of an algorithm in a mobile ad hoc network is an alternating sequence of configurations and timed events $c_0, e_1, c_1, e_2, \ldots$ satisfying the following: (1) The real times of the events form a monotonically increasing sequence; if the execution is infinite, then the times increase without bound. (2) Messages are generated, sent and received in ways that are consistent with the assumptions made for the particular algorithm. (3) Each configuration after the first follows appropriately from the previous, according to the specification of the intervening event.

CHAPTER IV

TOKEN CIRCULATION IN MOBILE AD HOC NETWORKS

A. Introduction

In this chapter, we present theoretical analyses of a distributed *token circulation* algorithm that causes a token to continually circulate through all the nodes of a network. When a token is circulated, a *round* is said to complete when every node has been visited by this token at least once. The *length* of a round is the number of node visits made by the token in one round. We give a rigorous worst-case analysis of the Local-Recency (LR) token circulation algorithm in static networks, which shows quite good performance in simulations. The results appeared in [6].

B. Analysis of Local Recency (LR) Algorithm

The LR algorithm is introduced in [5]. In LR, the token contains a timestamp for each mobile node indicating when the mobile node received the token most recently. The token is forwarded to the neighbor which has been visited least recently. The pseudocode is presented in Figure 1.

```
$LR\_initialize(token\ t)$:
  1  $t.ts = [0, 0, \cdots, 0]$;
$LR\_forward(token\ t)$:
  2  $t.ts[i] = max(t.ts) + 1$;
  4  $next$ = index of $t.ts$ entry among all neighbors
          of $p_i$ with minimum value (break ties by id);
  5  send $t$ to $p_{next}$;
```

Fig. 1. Token circulation algorithm: LR

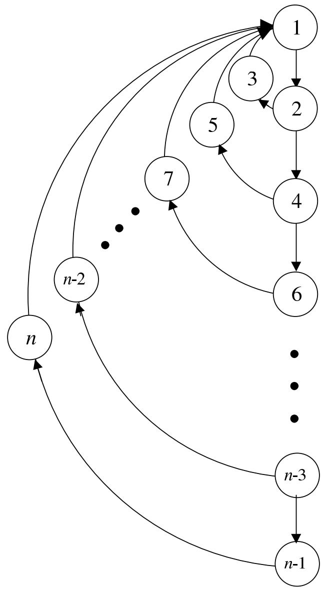


Fig. 2. A directed graph with exponential round length

Compared to all the algorithms presented in [5], LR shows quite good performance in both static and dynamic 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 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 [49]. In particular, on the graph in Figure 2, the LR algorithm has round length $2^{(n+1)/2} - 1$. 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 $(5n - 3)/4$ for all the rounds; if $n = 4k + 1$, $k > 0$, the length of the first round is $(5n - 1)/4$ and the length of the subsequent rounds is $(7n - 15)/4$.

The rest of this chapter 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 edges on the longest shortest path) is denoted by D.

In [5], we saw that every node is visited infinitely often. Here we prove the round length is $O(n \cdot \Delta^{D+1})$.

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* $O(n \cdot \Delta^{D+1})$.

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

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

□

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 $k \geq 1$, $\forall i, j \geq k$, $\rho_i = \rho_j$. Furthermore, the round 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 3 depicts a unit disk graph that satisfies FP with $r = 16$, $p = 27$, and $t = 2$. Unit disk graphs are widely employed to model ad-hoc networks — all the nodes are assumed to have identical transmission range and two nodes are connected if and only if they are in each other's transmission range. The dotted circles in Figure 3 indicate the transmission range of nodes. 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, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 25, 11, **10**, **27**, **3**, 4, 26, 8, 7, 6, 5, 17, 18, 19, 20, 21, 22, 23, 9, **10**, 11, 25, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 24, 2, 1, 12, 15, 14, 13, **16** and has length 82. 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 $r_i = r + (i-1) \cdot m$

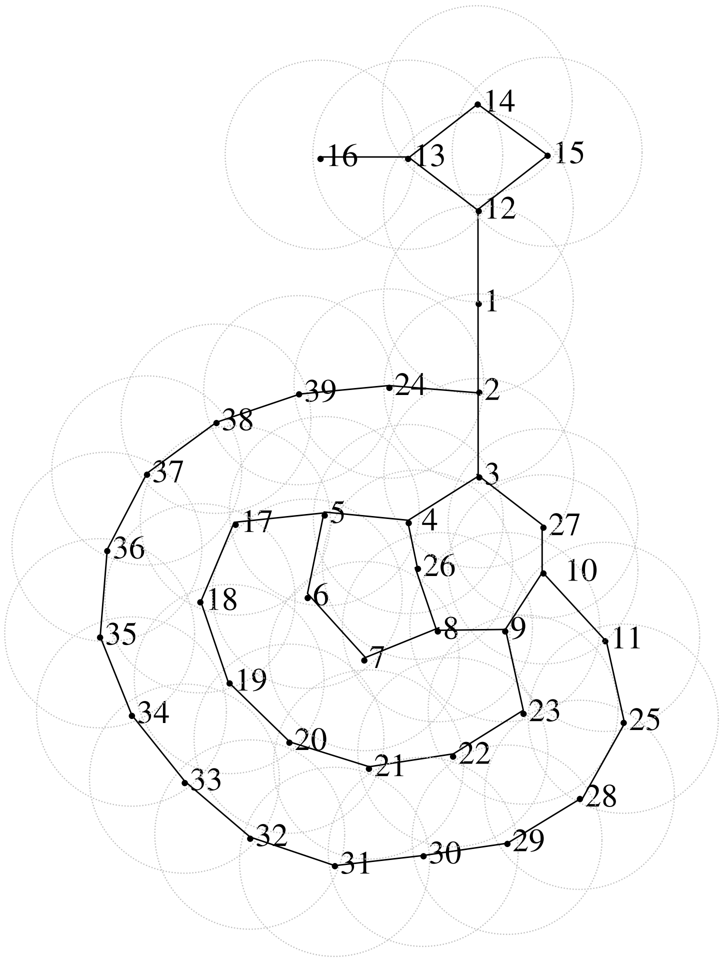


Fig. 3. A unit disk graph satisfying FP

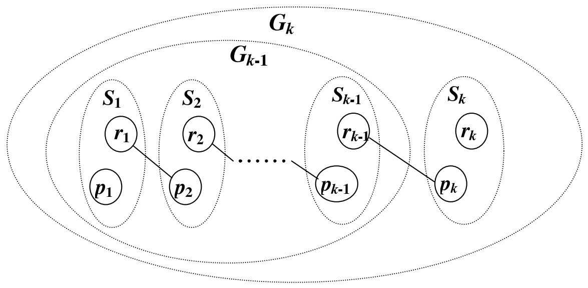


Fig. 4. Construction of G_k

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 4). 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 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 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 [5], 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 $\varphi_k = \sigma_k(p_k \rightarrow \langle p_k, r_{k-1}, \varphi_{k-1}, p_k\rangle)$ 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. $\square$

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 3 satisfies conditions FP_1 and FP_2 with $m = 39$, $t = 2$, and $|\sigma| = 82$. If G_k is constructed from this graph, then by Theorem 4, $|\varphi_k| = \Theta(t^k) = \Theta(t^{\frac{n}{m}}) = \Theta(2^{\frac{n}{39}})$. Note G_k can also be a unit disk graph. Let us compare this bound to the one obtained in Theorem 2, which is $O(n \cdot \Delta^{D+1})$, 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 $7k + 6$, this bound becomes $O(n \cdot 3^{7k+7}) = O(n \cdot 3^{\frac{7}{39}n+7})$. Thus we can see that the general upper bound is a large over-estimate for this family of graphs.

CHAPTER V

SELF-STABILIZING MUTUAL EXCLUSION IN MOBILE AD HOC NETWORKS

A. Introduction

The technology of mobile ad hoc networking is becoming increasingly prevalent and it has been an active research area. But much of the work in this area has focused on routing and medium access control protocols, and there is less work on distributed services. Furthermore, past work on distributed services focused on non-fault-tolerant algorithms.

An important technique for designing algorithms that tolerate arbitrary transient faults is self-stabilization, first introduced by Dijkstra in [22]. A system is defined as self-stabilizing if, starting from an arbitrary configuration, it is guaranteed to converge to a "legitimate" configuration or a "legitimate" execution in finite time. The system in the legitimate configuration (or execution) exhibits desired behavior. The arbitrary starting configuration could be the result of a transient failure that corrupted some aspect of the system state. A lot of work has been done on self-stabilization for static networks. However, it is inefficient to apply most existing self-stabilizing algorithms for static networks directly in a mobile ad hoc network for the following reason. Traditionally, a topology change has been considered as a kind of transient fault. But most known self-stabilizing algorithms require no subsequent failures (i.e., static topology) in order to converge to a legitimate configuration. If such an algorithm is applied in a mobile ad hoc network, which typically experiences frequent topology changes, it would essentially start over in trying to converge every time a topology change occurs, which is quite inefficient and might cause the algorithm never to reach a legitimate configuration. Thus a self-stabilizing algorithm for mobile ad hoc network needs to be designed specifically so that it exhibits correct behavior in the presence of topology changes, once it has converged.

In this chapter, we are interested in self-stabilizing distributed services for mobile ad hoc networks. We focus on the mutual exclusion problem. Informally, a mutual exclusion algorithm should satisfy two properties: *mutual exclusion* — there is no more than one processor in the critical section; and *progress* — some or all processors enter the critical section infinitely often. Our algorithm also provides a dynamic membership service so that processors can alternate participating in and not participating in the competition to enter the critical section. Mutual exclusion provides a mechanism for shared access to a resource, which is worth considering for mobile ad hoc networks, where mobile nodes may need to share common resources. Mutual exclusion can be used in the implementation of group communication, which is an important building block for applications that involve groups of cooperating hosts, such as mobile conferences, rescue missions and battlefield operations. Mutual exclusion is useful in many high level applications in mobile ad hoc networks such as E-learning in [50] to guarantee the consistency of shared objects.

Mutual exclusion and progress conditions cannot both be guaranteed in mobile ad hoc networks with arbitrary mobility for the following reason. Since reliable message delivery cannot be guaranteed in such a network, it is possible that some processors stop receiving messages from others even when they are always in the same connected component. If a processor makes the decision whether to enter the critical section based on communication with others, the progress condition cannot be guaranteed; if a processor can enter the critical section without communicating with others, mutual exclusion may be violated. Arbitrary mobility has prevented most protocols from providing fully reliable service in mobile ad hoc networks, and usually they provide a best-effort service instead. Our strategy to solve the mutual exclusion problem for mobile ad hoc networks is to guarantee mutual exclusion without constraints on the reliability of message delivery, and guarantee different levels of progress under different levels of constraints on mobility.

Our algorithm is based on the token circulation algorithm LR discussed in chapter IV, in which the token is forwarded by the current token holder to its neighboring processor that was visited by the token least recently. A distinguished processor in the system periodically generates a token. Upon receipt of a token, a processor checks the token state and its local state to decide whether it can enter the critical section; the check must ensure that only one processor is in the critical section even when there are multiple tokens in the system. Thus mutual exclusion can always be guaranteed, while progress depends on the frequency with which processors receive tokens.

B. Motivation

A simple solution for mutual exclusion is to circulate a token throughout the network and only the processor which holds the token enters the critical section. If there is exactly one token in the system and this token visits every processor that wants to enter the critical section infinitely often, this simple idea solves the mutual exclusion task. But it is not self-stabilizing because it cannot recover from a configuration with zero or multiple tokens.

One idea to make this simple idea self-stabilizing is as follows. In order to recover from a configuration with no token, a distinguished processor keeps generating tokens, which are circulated throughout the network. Upon receipt of a token, a processor checks the token state and its local state to decide whether it can enter the critical section. Such check should ensure that eventually only one processor is in the critical section even when there are multiple tokens in the system. An example is the self-stabilizing mutual exclusion algorithm on a ring in the shared memory model presented in [22]. A message passing version of this algorithm is presented in [51], which works on a fixed ring with FIFO links. In this algorithm, each token and each processor has a token ID. The distinguished processor, p_0, keeps generating tokens, setting the tokens' ID as its token ID. The tokens are forwarded along the ring. When processor p_i, $i \neq 0$, receives a token with ID not equal to p_i's token ID, p_i sets its token ID to the token's ID and enters the critical section. When a token comes back to p_0 after traveling around the whole ring, p_0 checks the token's ID and its token ID. If the token's ID is equal to its token ID, p_0 increases its token ID by one and enters the critical section.

In this chapter, we consider mutual execution in mobile ad hoc networks. If the composition of the network is fixed and known, we can predefine a *virtual ring*. Each token is circulated in the network by some token circulation algorithm, but it visits the processors according to their positions in the virtual ring: the first processor in a token's virtual ring is said to be visited by this token when it receives this token for the first time or the last processor visited by this token is the last processor in the virtual ring; each of the other processors p_i is said to be visited by a token if and only if the last processor visited by this token is the processor previous to p_i in the virtual ring. We notice the virtual link between two successive processors on the virtual ring is non-FIFO because of the changing topology — different tokens may travel along different paths from one processor to another. Applying the idea in [51] on such a virtual ring to solve mutual exclusion does not work with non-FIFO virtual links. A counter-example is shown in Fig. 5, in which the system has converged at time 0. But at time 3, processor p_1 and p_2 enter the critical section at the same time.

So we cannot apply the idea in [51] to mobile ad hoc networks directly. What is more, a fixed virtual ring is quite inefficient in mobile ad hoc networks as it may bear little resemblance to the actual topology. Finally, in a mobile ad hoc network, a processor p_i may be prevented from receiving a token due to an adversarial mobility pattern, and since all the processors that follow p_i in a virtual ring should be visited after p_i is visited, they will not be visited by any token and cannot enter the critical section any more, even if they keep receiving tokens.

In this chapter we propose an algorithm based on the ideas in [22] and [51]. In order

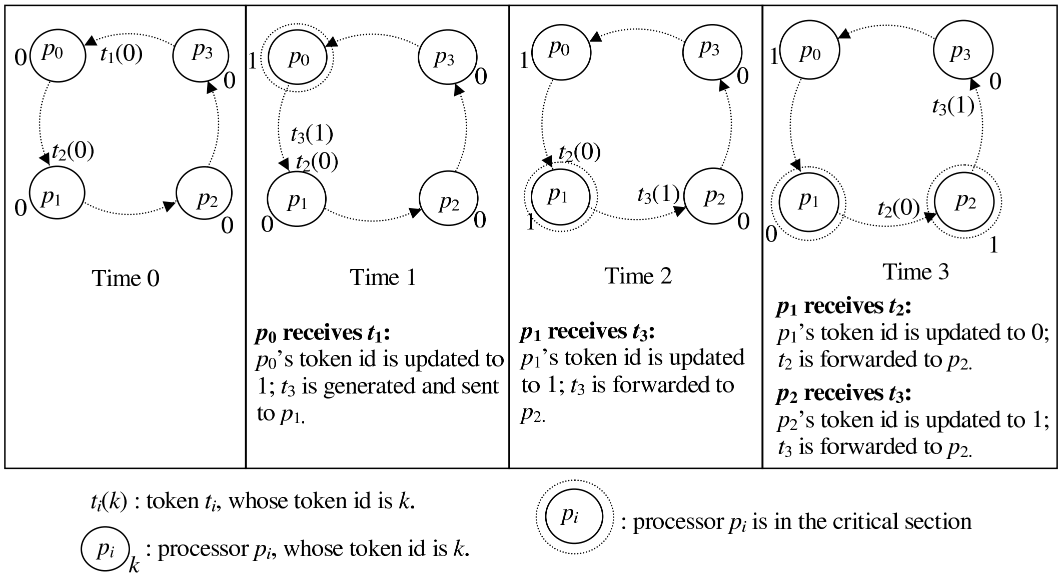

Time 0
Time 1
Time 2
Time 3
$t_1(0)$
$t_2(0)$
$t_3(1)$
p_0 receives t_1:
p_0's token id is updated to 1; t_3 is generated and sent to p_1.
p_1 receives t_3:
p_1's token id is updated to 1; t_3 is forwarded to p_2.
p_1 receives t_2:
p_1's token id is updated to 0; t_2 is forwarded to p_2.
p_2 receives t_3:
p_2's token id is updated to 1; t_3 is forwarded to p_2.
$t_i(k)$: token t_i, whose token id is k.
: processor p_i, whose token id is k.
: processor p_i is in the critical section

Fig. 5. A counter-example on a virtual ring with non-FIFO virtual links

to handle the mobility, we need to deal with:

- dynamic virtual ring, which is updated periodically to reflect the changing topology,
- non-FIFO virtual links on the virtual ring, and
- partitions, so that a disconnected processor will not prevent others from entering the critical section.

C. System Model

In this chapter, we focus on the distributed services that receive and respond to the requests from the application. The distributed services run on top of the basic communication service between adjacent processors as discussed in chapter III. We make the following additional assumptions:

- $Assumption_0$: There is a distinguished processor p_0 (we will discuss relaxing this assumption in section V.H).

- $Assumption_1$: An upper bound N on the number of processors is known, and each processor has a unique ID.

  For simplicity, we assume processors' IDs are in $[0, N-1]$ and processor p_i's ID is i; the correctness of the algorithm does not rely on it. Note that we do not require the processors know the IDs of others in advance.

- $Assumption_2$: There is an upper bound, d, on the message delay between two adjacent processors if the message is forwarded successfully. Messages may be lost but not corrupted.

  We do not assume the network is always connected. Later we will see that message loss and the connectivity of the network will only affect the progress of the algorithm (see section V.G.6) but not mutual exclusion. The assumption of no message corruption is required because otherwise the mutual exclusion problem cannot be solved even in a non-self-stabilizing way.

- $Assumption_3$: There is an upper bound, g, on the number of messages generated by a processor in each time unit.

A transient fault may cause the system to enter an arbitrarily corrupted configuration. The property of self-stabilization models the ability of a system to recover from a corrupted configuration under the assumption that the transient faults do not continue to occur. Our algorithm can recover from an arbitrary configuration, which may be caused by any transient fault, under the above assumptions.

A self-stabilizing algorithm requires variables to be bounded since memory space is finite. In a non-self-stabilizing algorithm, an unbounded variable can be simulated by using large but finite memory: since the algorithm starts from a correctly initialized state, the maximum value that would ever be assigned to the variable in, say, the next few hundred years can be estimated and sufficient memory to store that value can be allocated. But this approach does not work for self-stabilizing algorithms, since in an arbitrary configuration, the variable could take on its maximum value, no matter how much space is allocated, thus experiencing overflow.

$Assumption_1$ is useful to set bounds on the size of variables related to processors' identification. $Assumption_2$ and $Assumption_3$ enable the algorithm to rely on a maximum number of messages existing in the system, thus we can assign IDs to messages from a bounded range.

There are timers on each processor such that each timer has a value which is decremented at the rate of real time, and if a timer is set for time interval T at time t, then at time $t + T$ an event is initiated by the timeout expiring. A *timed event* is a pair consisting of an event and and a real number, which represents the real time at which the event occurs. We have the following assumption on the local computation.

- $Assumption_4$: At each processor no more than one event is activated at the same time and the time of the local computation is negligible.

  Note that one consequence of this assumption is that the time a processor stays in the critical section is negligible. This assumption is made for simplicity. We will discuss relaxing it in section V.H.

D. Problem Definition

We define the *mutual exclusion task* similarly to [22], in which "processor p_i being in the critical section in configuration c" refers to a predicate whose specifics will depend on the particular algorithm.

In some situations, processors may want to alternate between participating in and not participating in the competition to enter the critical section. In this chapter, we consider a system in which the composition of the competing processors is changing as indicated by the processors receiving $join$ or $leave$ requests from the application. We call a processor in an execution *active* (*inactive* respectively) if it experiences only a finite number of $join$ and $leave$ requests and the last one is a $join$ ($leave$ respectively) request. Three versions of the *dynamic mutual exclusion task*, corresponding to different progress conditions, are defined as follows:

Definition 1 ***(Dynamic No Deadlock Mutual Exclusion.)*** *The task of dynamic no deadlock mutual exclusion is a set of executions, such that for every execution in the set,* $Mutual\ Exclusion$*,* $Eventual\ Stopping$ *and* $No\ Deadlock$ *are satisfied:*

- $Mutual\ Exclusion$*: There is no more than one processor in the critical section in any configuration.*
- $Eventual\ Stopping$*: Every inactive processor enters the critical section only finitely often.*
- $No\ Deadlock$*: Some active processor enters the critical section infinitely often if the execution is infinite.*

Variations with increasingly strong progress conditions are:

- ***Dynamic No Lockout Mutual Exclusion****, in which* $No\ Deadlock$ *is replaced with*
    - $No\ Lockout$*: Every active processor enters the critical section infinitely often if the execution is infinite.*
- ***Dynamic Bounded Waiting Mutual Exclusion****, in which* $No\ Deadlock$ *is replaced with*
    - $Bounded\ Waiting$*: There exists time interval* T*, such that every active processor enters the critical section exactly once in every time interval* T*.*

This chapter describes an algorithm for mobile ad hoc networks that is self-stabilizing for $Mutual\ Exclusion$ and $Eventual\ Stopping$, and that is self-stabilizing for $No\ Deadlock$, $No\ Lockout$, and $Bounded\ Waiting$ with respect to various conditions (which are detailed later).

Definition 2 (Self-Stabilization). *An algorithm is self-stabilizing for property P (with respect to condition C), if, starting from an arbitrary configuration, every infinite execution of the algorithm (satisfying C) has a suffix satisfying P.*

In this chapter, property P can be one of $No\ Deadlock$, $No\ Lockout$ and $Bounded\ Waiting$, and condition C captures the assumptions about the network behavior.

E. Algorithm Overview

Here we give an informal description of our algorithm. Communication in our algorithm is based on token circulation, where each token is a message. We have two kinds of tokens: *mutual exclusion token* and $join\_request$ *token*, denoted by m-*token* and j-*token* respectively; the specific usage of each kind of token is described later in this section. Both kinds of tokens are routed using a self-stabilizing version of the token circulation algorithm LR analyzed in chapter IV. The reason for using LR (instead of a more traditional routing algorithm) is that we have developed a simple self-stabilizing version of LR, whereas it is much more problematic to design a self-stabilizing version of traditional routing algorithms. Our version of LR uses bounded timestamps and enforces a lifetime on tokens, by discarding any token that has been forwarded more than a certain number of hops. Because of the bounded lifetime, we can assign IDs to tokens from a bounded range and we define the operations on token IDs to be the operations on integers mod this bound. Note in section V.C, we explained why the values of a variable should be from a bounded range.

The execution of our algorithm is divided into phases. Informally, a phase is a bounded subsequence of execution in which all the connected members enter the critical section exactly once. The special processor p_0 maintains the membership, which is organized in two variables: new_set, a set of processors that are waiting to join the system, and $ring$, a list of members that have joined the system. The membership is updated only at the beginning of each phase.

- In each phase, p_0 repeatedly generates $m\text{-}token$s which carry new_set and $ring$.
    - Processors in new_set initialize their local states upon receipt of an $m\text{-}token$.
    - Members in $ring$ are visited by an $m\text{-}token$ according to their positions in $ring$. When a member is visited by an $m\text{-}token$,
        * it checks the token's state and its local state to see whether it is in the critical section, and updates its local state accordingly (the condition and update is designed to handle non-FIFO virtual links, see section V.F for details);
        * if it wishes to leave the system, it indicates this in the $m\text{-}token$.
    - Each $m\text{-}token$ records the order of processors that have forwarded it.
- When a processor wishes to join the system, it periodically sends a $j\text{-}token$ to processor p_0. The sending is repeated in order to increase the likelihood that the message gets through even if some copies are lost due to mobility.
- Processor p_0 starts a new phase when all the processors in new_set have initialized their states and all the members in $ring$ have entered the critical section, or a certain amount of time has elapsed since the current phase began. A member is considered to be disconnected if it did not enter the critical section in the previous phase. The member list is updated by p_0 as follows:

- $ring$ is updated to be the currently reachable members, which are the processors in new_set that have initialized their local state and the processors in $ring$ that have entered the critical section in last phase, minus the leave requests. The order of members in $ring$ is decided by the order of processors that have forwarded the m-$tokens$. Since a poorly chosen virtual ring requires a long time for a token to visit all the nodes, the idea behind this heuristic is to have the visiting order match more closely the actual network topology in order to increase efficiency,
- new_set is updated to be the senders of j-$tokens$ received by p_0 in the last phase.

F. Algorithm

Here we introduce the underlying token circulation algorithm in section V.F.1, the data structures in section V.F.2, the interfaces to the application in section V.F.3, and the mutual exclusion algorithm in section V.F.4.

1. Token Circulation Algorithm: LR_L

The non-self-stabilizing LR algorithm was discussed in Chapter IV, in which there is a unique token t that contains an array ts of N integers (the "*timestamp array*") initialized to all 0's; $ts[i]$ is a $timestamp$ indicating when processor p_i received the token most recently. In LR, a token is forwarded to a neighboring processor that was visited by this token least recently.

In our algorithm we use a variant of LR, denoted by LR_L, which is similar to LR except that the timestamps in LR_L are in the range $[0, L]$, the operation "+" on elements of $t.ts$ is addition mod $(L + 1)$, and only tokens with maximum timestamp less than L

are forwarded to the next processor (Fig. 6.) The value of L should be chosen based on the expected network behavior to guarantee that most of the tokens can be forwarded to every processor within L hops (see section V.G.6 for further discussion). If token t is not routed to the next processor, we say t is no longer in the system. In the sequel, we refer to t's *lifetime* as the time token t has stayed in the system. In this chapter, we consider a self-stabilizing algorithm, which may start from an arbitrary configuration. The following lemma shows that the lifetime of every token forwarded by LR_L is bounded no matter what values are initially in the timestamp array.

Lemma 5 *Consider any execution σ, in which tokens are forwarded for LR_L. The lifetime of any token in σ is no more than $M_{lt} = L \cdot d$. In more detail, in any configuration c, token t has been in the system for no more than $max(t.ts) \cdot d$, and t can be in the system for no more than $(L - max(t.ts)) \cdot d$ hops after c.*

Proof. Denote the initial timestamp array of token t by $t.ts^0$. From the code we can see $max(t.ts)$ is increased by 1 each time t is forwarded, so t cannot be forwarded for more than L hops, and t's lifetime is no more than $L \cdot d$ by $Assumption_2$. Suppose in configuration c, t is in p_i's incoming queue. Then t has been forwarded for $max(t.ts) - max(t.ts^0)$ hops, and it cannot be forwarded for more than $L - max(t.ts)$ hops after c. By $Assumption_2$, t has stayed in the system for no more than $max(t.ts) \cdot d$ time, and t can stay in the system for no more than $(L - max(t.ts)) \cdot d$ after c time. □

2. Data Structures

The denotations in TABLE I are used in our algorithm, where L is the upper bound on the number of hops taken by tokens that is enforced by LR_L. The choice of L only affects the progress of the algorithm (see section V.G.6), but not mutual exclusion. Note M_{tidr} is

```
$LR_L\_initialize(token\ t)$:
  1 $t.ts = [0, 0, \cdots, 0]$;

$LR_L\_forward(token\ t)$:
  2 $t.ts[i] = max(t.ts) + 1$;
  3 **if** $(max(t.ts) \geq L)$ **then return**;
  4 $next$ = index of $t.ts$ entry among all neighbors
        of $p_i$ with minimum value (break ties by ID);
  5 send $t$ to $p_{next}$;
```

Fig. 6. Token circulation algorithm: LR_L

the maximum difference between any two token IDs at any given time after convergence, while M_{tid} is the bound on the $token\_id$ type. The reset values of each timer are listed in Table II (the explanation of each timer is listed in Table III.) Recall that N, g and d were introduced in section V.C.

Table I. Denotations

Denotation	Value	Explanation
M_{lt}	$L \cdot d$	upper bound on token's *lifetime*
M_{tidr}	$(3g+1)M_{lt} + 2$	maximum difference between any two token ids after convergence, i.e. range of token IDs (difference is defined in this section)
M_{tid}	$2M_{tidr} + 2$	upper bound on $token\_id$ variables

We define the $token\_id$ type to be the set of integers in $\{0, 1, \ldots, M_{tid}\}$. The operations on variables of type $token\_id$ variables are the operations on integers $mod\ (M_{tid}+1)$. In particular, given two $token\_id$ variables, tid_1 and tid_2, the operation $[tid_1, tid_2]$ returns $\{tid_1, tid_1+1, \ldots, tid_2\}$ if $tid_1 \leq tid_2$, and $\{tid_1, tid_1+1, \ldots, M_{tid}\} \cup \{0, 1, \ldots, tid_2\}$ otherwise. We define the *difference* between tid_1 and tid_2 as $min\{t_{max} - t_{min}, t_{min} - t_{max} + M_{tid} + 1\}$, where $t_{max} = max\{tid_1, tid_2\}$ and $t_{min} = min\{tid_1, tid_2\}$. Note that the difference between tid_1 and tid_2 is the same as the difference between tid_2 and tid_1.

Table II. Timers' reset values

$Timers$		Reset values	Requirement on reset values
$timers$ on p_0	$timer_0^{phase}$	TM_{phase}	$> M_{lt}$
	$timer_0^{gen}$	TM_{gen}	
$timers$ on p_i, $i \neq 0$	$timer_i^{app}$	TM_{app}	$> (3M_{lt} + 3TM_{phase})$
	$timer_i^{join}$	TM_{join}	$> (M_{lt} + 3TM_{phase})$
		TM_{join_re}	$< TM_{join}$

For example, if $M_{tid} = 12$, then $[5, 9] = \{5, 6, 7, 8, 9\}$, $[9, 5] = \{$9, 10, 11, 12, 0, 1, 2, 3, 4, 5$\}$, the difference between 5 and 9 is 4 and the difference between 2 and 9 is 5.

The fields on tokens and the variables at processors are listed in Table III.

3. Interfaces to the Application

The application process on processor p_i submits a $join$ or $leave$ request to the mutual exclusion service by setting $member_i$ to $true$ or $false$ (see events $join_i$ and $leave_i$ in section V.F.5.) A processor cannot join the system if it has currently joined, that is, $member_i = true$ (see precondition of $join_i$ in section V.F.5), and it cannot leave if it is not in the system, that is, $member_i = false$ (see precondition of $leave_i$ in section V.F.5). The time interval between two requests is no less than TM_{app}, which is enforced by using $timer_i^{app}$ (see line 2.2 of $leave_i$ and the precondition of $join_i$ in section V.F.5). Such an interval is required because when a processor leaves the system, the system needs time to erase the information related to it, so that when it rejoins the system, it will not update its state based on false information.

The application decides whether it can enter the critical section based on the part of the configuration accessible to p_i, that is, p_i's local state and the state of its incoming queue (see Predicate cs_i for p_i to enter the critical section, section V.F.5). Processor p_0 enters the critical section at the beginning of each phase (see cs_0 and the precondition of event

$start\_phase_0$). Processor p_i, $i \neq 0$, enters the critical section if p_i has joined the system (see cs_i^1), it is p_i's turn to be visited (see cs_i^2), and p_i has a correct token ID (see cs_i^3). We notice in a $receive_i^m$ event following any configuration in which p_i is in the critical section, line 9.2 is $false$, line 9.8 is $true$, and line 9.13 is $true$.

4. Mutual Exclusion Algorithm

Mutual exclusion is achieved by four kinds of events on processor p_0 and three kinds of events on the other processors (see code in section V.F.5). Processor p_0 keeps generating $m\text{-}token$s in event $generate_0^m$. A new phase is started by p_0 in event $start\_phase_0$. Event $receive_0^j$ is activated when p_0 receives a $j\text{-}token$, and event $receive_0^m$ is activated when it receives an $m\text{-}token$. Processor p_i, $\neq 0$, sends $j\text{-}token$s to p_0 in event $send_i^j$, which are forwarded to p_0 among processors in event $receive_i^j$. Event $receive_i^m$ is activated when p_i receives an $m\text{-}token$.

Now we explain how these events interact with each other to achieve mutual exclusion. Processors join the system by sending join requests. Upon receipt of a $join$ request from the application, p_i sets $status_i$ to $joining$, indicating it is waiting to join the system, and sends $j\text{-}token$s (lines 7.1 - 7.6), which are forwarded to p_0 by processors using LR_L (lines 8.1 - 8.2). Upon receipt of such a token (line 6.1 - 6.4), p_0 records the information in its local variable $join\_set_0$, which will be used to update the membership at the begining of the next phase. Note in event $send_i^j$, a processor resends the join request in every TM_{join_re} time interval if it does not receive an $m\text{-}token$ within time TM_{join} by using the timers.

Now we consider the execution of each phase. In each phase, p_0 repeatedly generates $m\text{-}token$s (lines 3.1 - 3.10). Each $m\text{-}token$ has an id and carries membership information in fields new_set and $ring$. Note that since p_0 updates its local token id and membership only at the beginning of each phase, all the tokens generated in the same phase have the same token id and membership information.

We now explain how a processor p_i, $i \neq 0$, responds to the receipt of an $m\text{-}token$ t (lines 9.1 - 9.24).

- If p_i is waiting to join the system and its join request was received by p_0 in the last phase, line 9.2 is true when p_i receives the first $m\text{-}token$ that is generated in the current phase. In this case, p_i takes following actions (line 9.3-9.6): it initializes its local token id tid_i as $t.id$ and sets $status_i$ to $joined$, indicating it has initialized its local state and joined the system; it also adds its id to $t.visit\_set$ (line 9.5), ensuring that it will ignore token t if it receives t again. (As we explain below, once p_i joins the system, p_i will be moved from new_set to $ring$ in the next phase, and line 9.2 will be false in the next phase.)
- Otherwise, we say the processor is "visited" by $m\text{-}token$ t if line 9.8 is true. Note that this condition is true only if p_i is a member in $ring$ and it enforces the property that each token visits processors according to their positions in the $ring$. In line 9.9 p_i adds its id to $t.visit\_set$, ensuring that it will ignore token t if it receives t again. Processor p_i updates its local state and checks whether it is in the critical section. If $t.id$ is 0 and p_i's token id is currently out of the range $\{M_{tid}, 0, \ldots, M_{tidr}\}$, then an inconsistency in the state of the system has been detected, and p_i's token id is reset to 0 (lines 9.10 - 9.12). If $t.id$ is one more than p_i's token id, then p_i enters the critical section (lines 9.13 - 9.15). After leaving the critical section, p_i sets its token id to be $t.id$ (line 9.14), which will prevent p_i from entering the critical section again due to tokens generated in the current phase. The intuition of how it works follows. Formal proofs are provided in section V.G.
  - Consider some phase b in which p_0 keeps generating $m\text{-}token$s with id 0. All the visited processors have local token id in range $\{M_{tid}, 0, \ldots, M_{tidr}\}$ at line 9.12 and those that have token id M_{tid} at line 9.12 will enter the critical section

and update their token id to 0 in line 9.14 since $t.id = 0$. Thus all the visited processors will have token ids in $[0, M_{tidr}]$ at the end of phase b.

  - In phase $b + 1$, p_0 keeps generating token with id 1 and all the processors with token id 0 update their token id to 1 when visited by such a token. Processors with other token ids will not change their token id since the bounded token lifetime guarantees that no token with id in $[2, M_{tidr} + 1]$ exists in phase $b + 1$, and only tokens with id in this range can change a processor's token id that is in $[1, M_{tidr}]$ (see lines 9.10 and 9.13). So at the end of phase $b + 1$, all the visited processors have id in $[1, M_{tidr}]$.
  - Generally speaking, at the begining of phase $b + k$, $1 \leq k \leq M_{tidr}$, all the processors have token ids in $[k - 1, M_{tidr}]$. In phase $b + k$, p_0 keeps generating tokens with id k. All the processors with id $k - 1$ updates their token ids to k when visited by a token generated in the current phase, while others remain unchanged since no token with id in range $[k + 1, M_{tidr} + 1]$ exists due to the bounded token lifetime. Thus at the end of phase $b+k$, all the visited processors have token ids in $[k, M_{tidr}]$. So all the visited processors have the same token id at the end of phase $b + M_{tidr}$; we say the system converges at this point.
  - After convergence, at the begining of phase $b + k$, $k > M_{tidr}$, all the members have token id $k - 1$. In this phase p_0 keeps generating tokens with id k, and the members enter the critical section only when they are visited by a token generated in the current phase for the first time. Since all these tokens visit processors in the same order, specified by $ring$, $Mutual\ Exclusion$ is guaranteed, even if the virtual links are non-FIFO.

- When a processor wishes to leave the system, it indicates this in the $m\text{-}token$s that pass through it (line 9.20). The order of processors that have forwarded the token is updated in line 9.22. At last, the token is forwarded to the next processor using LR_L (line 9.24).

When p_0 receives an $m\text{-}token$, it only processes the token if it has the same id as p_0's token id, since only these tokens can enable a processor to enter the critical section after the system converges (lines 5.1 - 5.11). The information about reachable members, routing order and leave requests are collected in line 5.3-5.5. If all the members (those in $ring$) have entered the critical section and all the waiting processors (those in new_set) have initialized their local state, that is, line 5.6 is true, then p_0 sets $timer_0^{phase}$ to 0 in line 5.7 to start a new phase (see the precondition of event $start\_phase_0()$). Otherwise the token is forwarded to the next processor in line 5.9.

A new phase is started in event $start\_phase_0$ by p_0 (lines 4.1 - 4.11). There are two situations for the precondition to be true. The first is line 5.7 as mentioned above. The other is where timer $timer_0^{phase}$ goes off; the purpose of $timer_0^{phase}$ is to prevent a phase from never ending due to partition. In this event, p_0 updates the membership: new_set consists of the processors whose join requests were received in the last phase (line 4.2), and $ring$ is the currently reachable members, (those in new_set that have initialized their local state and those in $ring$ that have entered the critical section), minus the leave requests (line 4.3). Processor p_0 also figures out if the system has become disconnected, by checking whether any of the processors in the old group were not visited yet did not send a leave request (line 4.1). Since $m\text{-}token$s generated in the last phase continue circulating, a disconnected processor may be visited by an $m\text{-}token$ which enables it to enter the critical section in the new phase. In order to avoid the situation that a disconnected processor enters the critical section at the same time as a member enters, p_0 waits until all the $m\text{-}token$s are discarded to generate new tokens for the new phase (lines 4.6-4.9). Note that a disconnected member can rejoin the system by sending $j\text{-}token$s — as mentioned above, processor p_i resends

Predicate cs_i for p_i to enter the critical section:

- $i = 0$: $cs_0 \equiv (timer_0^{phase} = 0)$
- $i \neq 0$: $cs_i \equiv cs_i^1 \wedge\ cs_i^2 \wedge\ cs_i^3$, where cs_i^1,cs_i^2 and cs_i^3 are defined as follows, in which $m\text{-}token\ t$ is the first token in $queue_i$.

$cs_i^1 \equiv (member_i = true) \wedge (status_i = joined)$

$cs_i^2 \equiv p_i$ is the first processor in $t.ring$ that is not in $t.visit\_set$

$cs_i^3 \equiv t.id = (tid_i + 1)\ mod\ (M_{tid} + 1)$

Fig. 7. Predicate cs_i for p_i to enter the critical section

/* Event $join_i$ occurs when p_i receives a request from the application to join the system */

Event $join_i()$ on p_i, $i \neq 0$:

pre: Receive $join$ request from the application, and $(member_i = false) \wedge (timer_i^{app} = 0)$

action:

```
1.1  $member_i = true$;
```

Fig. 8. Event $join_i()$ on p_i, $i \neq 0$

$j\text{-}token$s in event $sent_i^j$ if it does not receive an $m\text{-}token$ within time TM_{join}.

5. Code

The predicate for p_i to enter the critical section and the code of each event on the processors are listed in Fig. 7 - Fig.16.

/* Event $leave_i$ occurs when p_i wants to leave the system. p_i can rejoin only after time TM_{app} */

Event $leave_i()$ **on** p_i**,** $i \neq 0$**:**

pre: Receipt of $leave$ request from the application, and $(member_i = true)$

action:

```
2.1  $member_i = false$;
2.2  $timer_i^{app} = TM_{app}$;
```

Fig. 9. Event $leave_i()$ on p_i, $i \neq 0$

/* p_0 continues generating $m\text{-}token$s in event $generate_0^m$ */

Event $generate_0^m()$ **on** p_0:

pre: $timer_0^{gen} = 0$

action:

```
 3.1  create new $m\text{-}token$ data structure $t$;
 3.2  $LR_L\_initialize(t)$;
 3.3  $t.id = tid_0$;
 3.4  $t.ring = ring_0$;
 3.5  $t.new\_set = new\_set_0$;
 3.6  $t.visit\_set = \{p_0\}$;
 3.7  $t.route\_list = \langle p_0 \rangle$;
 3.8  $t.leave\_set = \emptyset$;
 3.9  $LR_L\_forward(t)$;
3.10  set $timer_0^{gen} = TM_{gen}$;
```

Fig. 10. Event $generate_0^m()$ on p_0

/* A new phase is started by p_0 in event $start\_phase_0$ */

Event: $start\_phase_0()$ **on** p_0**:**

pre: $timer_0^{phase} = 0$

action

```
  /* Update membership */
 4.1  $discon_0 = !((ring_0 - visit\_set_0) \subseteq leave\_set_0)$;
 4.2  $new\_set_0 = join\_set_0$;
 4.3  $ring_0$ = elements of $(visit\_set_0 - leave\_set_0)$ in the order in which they appear in
$route\_list_0$ (if there exist elements in $(visit\_set_0 - leave\_set_0 - route\_list_0)$, append them
to the end);

  /* Update token id and reset variables */
 4.4  $tid_0 = (tid_0 + 1)\ mod\ (M_{tid} + 1)$;
 4.5  $join\_set_0 = leave\_set_0 = \emptyset$, $visit\_set_0 = \{p_0\}$;
          $\boxed{\text{Critical Section}}$

  /* Clean up tokens if there are disconnected members: $timer_0^{gen}$ is set to $M_{lt}$ to ensure $p_0$
stops generating tokens for $M_{lt}$ time */
 4.6  if $(discon_0 == true)$ then
 4.7      set $timer_0^{gen} = M_{lt}$;
 4.8  else
 4.9      set $timer_0^{gen} = 0$;
4.10  endif

  /* Reset phase timer */
4.11  set $timer_0^{phase} = TM_{phase}$;
```

Fig. 11. Event: $start\_phase_0()$ on p_0:

/* Event $receive_0^m$ occurs when p_0 receives a $m\text{-}token$. */

Event $receive_0^m(m\text{-}token\ t)$ **on** p_0**:**

pre: The first token in $queue_0$ is $m\text{-}token\ t$.

action:

```
 5.1  remove $t$ from $queue_0$;

 5.2  if $(t.id == tid_0)$ then
        /* Update membership information */
 5.3    $visit\_set_0 = visit\_set_0 \cup t.visit\_set$;
 5.4    $route\_list_0 = t.route\_list$;
 5.5    $leave\_set_0 = leave\_set_0 \cup (t.leave\_set \cap ring_0)$;

        /* If the token has visited all the members, start a new phase, otherwise forward the
 token to the next processor using $LR_L$*/
 5.6    if $((ring_0 \cup new\_set_0) \subseteq visit\_set_0)$ then
 5.7      set $timer_0^{phase} = 0$;
 5.8    else
 5.9      $LR_L\_forward(t)$;
5.10    endif
5.11  endif
```

Fig. 12. Event $receive_0^m(m\text{-}token\ t)$ on p_0

/* Event $receive_0^j$ occurs when p_0 receives a join request. Membership is updated accordingly */

Event: $receive_0^j(j\text{-}token\ t_j)$ **on** p_0**:**

pre: The first token in $queue_0$ is $j\text{-}token\ t_j$.

action

```
6.1 remove $t_j$ from $queue_0$;
6.2 **if** $(t_j.pid \notin (ring_0 \cup new\_set_0))$ **then**
6.3     $join\_set_0 = join\_set_0 \cup \{t_j.pid\}$;
6.4 **endif**
```

Fig. 13. Event: $receive_0^j(j\text{-}token\ t_j)$ on p_0

/* In event $send_i^j$, p_i continues generating $j\text{-}token$s to increase the likelihood that such request arrives at p_0 */

Event $send_i^j()$ **on** p_i**,** $i \neq 0$:

pre: $(member_i = true) \wedge (timer_i^{join} = 0)$

action:

```
7.1 create new $j\text{-}token$ data structure $t_{join}$;
7.2 $LR_L\_initialize(t_{join})$;
7.3 $t_{join}.pid = p_i$;
7.4 $LR_L\_forward(t_{join})$;
7.5 $status_i = joining$;
7.6 set $timer_i^{join} = TM_{join\_re}$;
```

Fig. 14. Event $send_i^j()$ on p_i, $i \neq 0$

/* Event $receive_i^j$ occurs when p_i receives a join request. The join request is forwarded to p_0 using LR_L */

Event $receive_i^j(j\text{-}token\ t_j)$ **on** p_i**,** $i \neq 0$:

pre: The first token in $queue_i$ is $j\text{-}token\ t_j$

action:

```
8.1  remove $t_j$ from $queue_i$;
8.2  $LR_L\_forward(t_j)$;
```

Fig. 15. Event $receive_i^j(j\text{-}token\ t_j)$ on p_i, $i \neq 0$

G. Correctness

Here we show the correctness by proving that eventually tokens, non-members and members will all exhibit good behaviors. The organization of the proof is shown in Fig. 17. We explain in more detail below.

In section V.G.1, we show $Eventual\ Stopping$ is true from the initial configuration of any execution (c_0 in Fig. 17), and introduce the term "phase". Informally, a phase is a bounded subsequence of an execution in which the membership and tid_0 remain unchanged (they are updated only at the beginning of each phase) and each connected member enters the critical section exactly once.

We discuss the properties of $m\text{-}token$s in section V.G.2. In an arbitrary configuration, the IDs of tokens may spread throughout the range $[0, M_{tid}]$. We show that eventually they will converge to a narrow range; this property is called $token\_safe$. We prove it by defining property $conv_{token}$ of a configuration and showing that eventually the execution will reach a configuration satisfying $conv_{token}$ (c_1 in Fig. 17) and after that $token\_safe$ is always guaranteed.

/* Event $receive_i^m$ occurs when p_i receives a $m\text{-}token$. p_i checks its local state and the token's state to see whether it can enter the critical section */

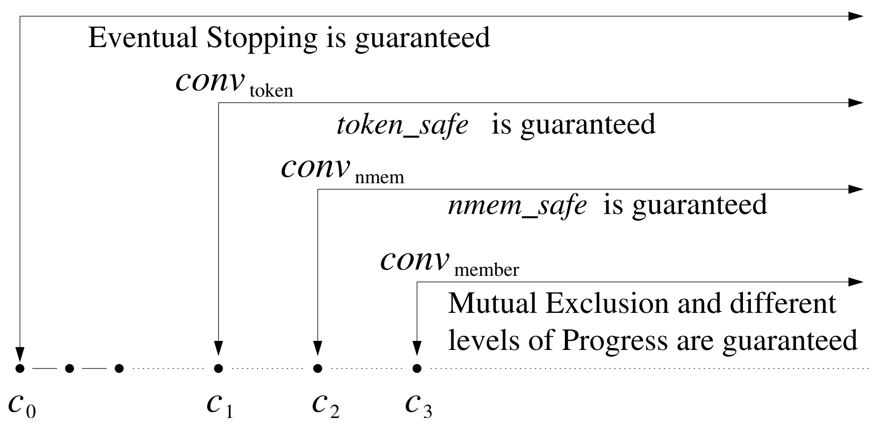
Event $receive_i^m(m\text{-}token\ t)$ **on** p_i**,** $i \neq 0$:

pre: The first token in $queue_i$ is $m\text{-}token\ t$.

action:

```
9.1  remove $t$ from $queue_i$;
9.2  if(($member_i$ == $true$)&&($status_i$ == $joining$)&& ($p_i \in t.new\_set$)&&($p_i \notin$
     $t.visit\_set$)) then
9.3      $tid_i = t.id$;
9.4      $status_i = joined$;
9.5      $t.visit\_set = t.visit\_set \cup \{p_i\}$;
9.6      set $timer_i^{join} = TM_{join}$;
9.7  else
9.8      if (($member_i$ == $true$) && ($status_i$ == $joined$) && ($p_i$ is the first processor in
           $t.ring$ not in $t.visit\_set$)) then
9.9          $t.visit\_set = t.visit\_set \cup \{p_i\}$;
9.10         if ($t.id$ == 0)&&($tid_i \notin \{M_{tid}, 0, \ldots, M_{tidr}\}$) then
9.11             $tid_i = 0$;
9.12         endif
9.13         if ($t.id$ == $(tid_i + 1) mod (M_{tid} + 1)$) then
                 $\boxed{\text{Critical Section}}$
9.14             $tid_i = t.id$;
9.15         endif
9.16         set $timer_i^{join} = TM_{join}$;
9.17     endif
9.18 endif
9.19 if ($member_i \neq true$) then
9.20     append $p_i$ to $t.leave\_set$;
9.21 else if ($p_i \notin t.route\_list$) then
9.22     append $p_i$ to $t.route\_list$;
9.23 endif
9.24 $LR_L\_forward(t)$;
```

Fig. 16. Event $receive_i^m(m\text{-}token\ t)$ on p_i, $i \neq 0$



Eventual Stopping is guaranteed
conv token
token_safe is guaranteed
conv nmem
nmem_safe is guaranteed
conv member
Mutual Exclusion and different levels of Progress are guaranteed
c0
c1
c2
c3

Fig. 17. Organization of proof

The properties of non-members are discussed in section V.G.3. In an arbitrary configuration, there may exist false information related to processors which are no longer members, based on which these processors may violate mutual exclusion when they join the system. We show that eventually a processor can join the system only when no information related to it exists; this property is called $nmem\_safe$. We prove it by first defining property $conv_{nmem}$, in which we define the appropriate levels of information in a configuration related to a non-member according to the time it has left the system. Then we show that any execution will eventually reach a configuration satisfying $conv_{nmem}$ (c_2 in Fig. 17)and after that $nmem\_safe$ is always guaranteed.

Then in section V.G.4, we focus on the properties of members in an execution in which all the configurations are both $token\_safe$ and $nmem\_safe$. We define property $conv_{member}$ of a configuration related to members' state. We prove eventually the execution will reach a configuration satisfying $conv_{member}$ (c_3 in Fig. 17), and we say the execution has converged if it has reached such a point.

In section V.G.5, we prove $Mutual\ Exclusion$ in any execution that has converged. We also provide a discussion of different levels of progress achieved by the algorithm given different performance of token circulation in section V.G.6.

1. Properties of Execution

Since $member_i$ is set to $false$ when p_i receives a $leave$ request from the application (see section V.F.5), p_i cannot enter the critical section before the next $join$ request. Thus $Eventual\ Stopping$ can be guaranteed.

Theorem 6 *Given any execution σ, $Eventual\ Stopping$ is guaranteed in σ.*

Our algorithm is based on circulating m-$token$s, which are forwarded by LR_L, but *visit* processors based on their member list. A formal definition of "visiting" is given as follows, from which we see that an m-$token$ t visits processors in $t.ring$ according to their positions in $t.ring$ (see line 9.8), and each processor in $t.ring$ or $t.new\_set$ is visited by t at most once (see line 9.5 and line 9.9).

Definition 3 ***(Visiting).*** *Given an m-$token$ t, if t is the first token in $queue_i$ in a configuration c, p_i is visited by t in c if and only if line 9.2 or line 9.8 is $true$.*

We divide the execution into "phases", which are defined as follows (Fig. 18):

Definition 4 (phase). *Given an execution σ, σ is divided into phases:*

- *phase 0 starts at the beginning of σ, and ends just before the first $start\_phase_0$ event in σ; and*
- *phase i, $i > 0$, starts immediately after the end of phase $i-1$ and ends just before the second occurrence after phase $i-1$ of a $start\_phase_0$ event.*

In the sequel we call the m-$token$s with ID equal to p_0's token ID as "current" tokens. We denote the $start\_phase_0$ event of phase k by e_s^k, the configuration previous to e_s^k by c_s^k, the first $generate_0^m$ event of phase k by e_g^k and the configuration following e_g^k by c_g^k. We have the following properties for each phase:

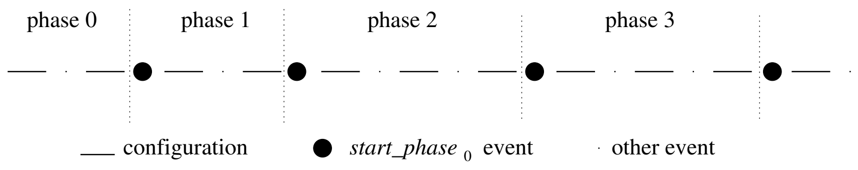


Fig. 18. Definition of phases

Lemma 7 *Given any execution σ and any $k \geq 0$,*

(1) *In phase k, tid_0, $ring_0$ and new_set_0 remain unchanged; their values in phase k are denoted by tid_0^k, $ring_0^k$, and $new\_set_0^k$.*

(2) *All tokens generated in the same phase have the same values for id, $ring$ and new_set.*

(3) *The IDs of the m-$tokens$ generated in a phase are one larger than those of the m-$tokens$ generated in the previous phase, that is, $tid_0^k = tid_0^{k-1} + 1$.*

(4) *The upper bound on the phase length is TM_{phase}.*

Proof. All m-$tokens$ generated by p_0 are generated with $tid = tid_0$, $ring = ring_0$ and $new\_set = new\_set_0$ in the $generate_0^m$ event. Since the only update to $ring_0$, new_set_0 and tid_0 is line 4.1-4.4 in a $start\_phase_0$ event, in which tid_0 is increased by one, (1), (2) and (3) are proved.

A $start\_phase_0$ event is activated if and only if $timer_i^{phase} = 0$, in which $timer_i^{phase}$ is set to TM_{phase}. After that, $timer_i^{phase}$ keeps going down until it goes off or it is set to 0 by line 5.7 in a $receive_0^m$ event, so it takes at most TM_{phase} for a $start\_phase_0$ event to be activated again. Thus (4) is proved. □

2. Safe Properties of m-$tokens$

In this section, we discuss the properties related to m-$tokens$. In an initial configuration, the ID of a token may be any one of $[0, M_{tid}]$. We show in Lemmas 9 and 10 that eventually the IDs of tokens will converge to some subset of $[0, M_{tid}]$ (property $conv_{token}$) and starting from that point, there always exists an upper bound M_{tidr} on the difference of token IDs in a configuration (property $token\_safe$.) An execution in which all the configurations satisfy this bound exhibits good behaviors: a processor's token ID remains unchanged when certain conditions hold (Lemma 11), all the current tokens are generated in the current phase (Lemma 12), and all the members are those visited by a current token in the previous phase (Lemma 13).

First we define properties $conv_{token}$ and $token\_safe$.

Definition 5 *($conv_{token}$). A configuration c satisfies property $conv_{token}$ if and only if*

(1) *the number of m-tokens is no more than $g \cdot M_{lt}$;*

(2) *for every m-token t, we have $t.id \in [tid_0 - (g \cdot M_{lt} + g \cdot d \cdot max(t.ts) + 1)\ , tid_0]$;*

(3) *any two m-tokens with the same token ID have the same $ring$ and new_set;*

(4) *any m-token t with ID tid_0 satisfies $(t.ring = ring_0) \wedge (t.new\_set = new\_set_0)$.*

Definition 6 *($token\_safe$). A configuration is $token\_safe$ if and only if:*

(1) *for any m-token t, we have $t.id \in [\ tid_0 - M_{tidr}\ , tid_0]$;*

(2) *all the m-tokens with the same id have the same $ring$ and new_set.*

We will show in Lemmas 9 and 10 that, starting from an arbitrary configuration, eventually the execution converges to a configuration satisfying $conv_{token}$, after which $token\_safe$ is always guaranteed. The proof of this lemma is based on the fact that the number of phases is limited in each bounded time interval.

Lemma 8 *Given times T_1 and T_2, $T_1 < T_2$, if the number of m-tokens at T_1 is no more than k, then the number of $start\_phase_0$ events in $[T_1, T_2]$ is no more than $(\ k + g \cdot (T_2 - T_1)$ $+ \lceil (T_2 - T_1)/TM_{phase} \rceil\ )$.*

Proof. A $start\_phase_0$ event is activated only when $timer_0^{phase} = 0$, which occurs only in two situations:

- $timer_0^{phase}$ is set to 0 in a $receive_0^m$ event. The number of such events is no more than the number n_r of *m-token*s received by p_0 in $[T_1, T_2]$, which either exist at T_1 or are generated during $[T_1, T_2]$. By $Assumption_3$, we have $n_r \leq k + g \cdot (T_2 - T_1)$.
- $timer_0^{phase}$ goes off because the timer ticked down to 0. Since $timer_0^{phase}$ is never set to any values other than TM_{phase} and 0, and it is only set to 0 in a $receive_0^m$ event, at least TM_{phase} time elapses between any two consecutive times that the timer goes off due to ticking down, so it goes off at most $\lceil (T_2 - T_1)/TM_{phase} \rceil$ times in $[T_1, T_2]$.

So in $[T_1, T_2]$ the number of $start\_phase_0$ events is no more than $k + g \cdot (T_2 - T_1) + \lceil (T_2 - T_1)/TM_{phase} \rceil$. □

Lemma 9 *Starting from an arbitrary configuration c_0, within time $2M_{lt}$ there is a configuration satisfying $conv_{token}$.*

Proof. By Lemma 5, there is a configuration c_1 in which all the *m-token*s are generated by p_0 after c_0, and a later configuration c_2 in which all the *m-token*s are generated by p_0 after c_1. We will prove c_2 satisfies $conv_{token}$. Suppose c_2 is in phase k. By Lemma 5, c_2 occurs within $2M_{lt}$ from c_0.

By Lemma 5, all the *m-token*s in a configuration c after c_1 are generated within time M_{lt} before c, thus the number of *m-token*s in c is no more than $g \cdot M_{lt}$ by $Assumption_3$. So (1) of definition 5 is proved for c_2. Given any *m-token* t in c_2, assume t first occurs in configuration c_t in phase k_t. Since c_t is after c_1, the number of *m-token*s in c_t is no more than $g \cdot M_{lt}$. Let T be the time interval between c_2 and c_t. We have $T \leq d \cdot max(t.ts) \leq M_{lt}$ by Lemma 5, and $k - k_t \leq g \cdot M_{lt} + g \cdot T + \lceil T/TM_{phase} \rceil \leq g \cdot M_{lt} + g \cdot d \cdot max(t.ts) + 1 \leq$

$2 \cdot g \cdot M_{lt} + 1 < M_{tid}$ by Lemmas 5 and 8. Thus $m\text{-}token$s with the same ID are generated in the same phase by (3) of Lemma 7 and (2), (3) and (4) of definition 5 follow. □

Lemma 10 *Given any execution starting from a configuration c_0 satisfying $conv_{token}$, every configuration c in σ satisfies $token\_safe$.*

Proof. We denote the time interval between c and c_0 by T and consider any $m\text{-}token$ t in c. Assume c is in phase k.

If t is generated between c and c_0, say at time T_t in phase k_t, we have $T - T_t \leq M_{lt}$ by Lemma 5. Since all the tokens at T_t either exist in c_0 or are generated within M_{lt} before T_t by Lemma 5, the number of tokens at T_t is no more than $2g \cdot M_{lt}$ by $Assumption_3$. By Lemma 8, we have $k - k_t \leq 2g \cdot M_{lt} + g \cdot M_{lt} + 1 \leq M_{tidr}$. Thus in this case we have $t.id = tid_0^{k_t} \in [tid_0^k - M_{tidr}, tid_0^k]$ by Lemma 7. Otherwise we denote $max(t.ts)$ in c_0 by $max_{ts}^{c_0}$ and in c by max_{ts}^{c}, and $g \cdot M_{lt} + g \cdot d \cdot max_{ts}^{c_0} + 1$ by x. We have $T \leq d \cdot (L - max_{ts}^{c_0})$ $\leq M_{lt}$ by Lemma 5 and $k \leq g \cdot M_{lt} + g \cdot T + \lceil T/TM_{phase} \rceil \leq g \cdot M_{lt} + g \cdot d \cdot (L - max_{ts}^{c_0})$ $+ 1 \leq M_{tidr}$ by Lemma 8. By (2) of $conv_{token}$, we have $t.id \in [tid_0^0 - x, tid_0^0]$, which is a subset of $[tid_0^0 - x, tid_0^0 + k] = [tid_0^k - k - x, tid_0^k] \subseteq [tid_0^k - M_{tidr}, tid_0^k]$ because $k + x$ $\leq M_{tidr}$. Thus (1) is true in both cases.

Now we prove (2). First we consider the case that every $m\text{-}token$ in c is generated after c_0. Assume $m\text{-}token$ t is generated in phase k_t. We know $k_t \in [k - M_{tidr}, k]$ from above. Because $M_{tidr} \leq M_{tid}$, $m\text{-}token$s generated in different phases have different token IDs by Lemma 7, thus (2) is true.

Otherwise we know $k = tid_0^k - tid_0 \leq M_{tidr}$ from above. So in c, all the $m\text{-}token$s existing in c_0 have ID in $R_1 = [tid_0^0 - M_{tidr}, tid_0^0]$. By (2) of definition 5 all the $m\text{-}token$s generated after c_0 have IDs in $R_2 = [tid_0^0, tid_0^k] \subseteq [tid_0^0, tid_0^0 + M_{tidr}]$. Consider all the $m\text{-}token$s with ID $l \in R_1 \cup R_2$ in c. If $l \in [tid_0^0 - M_{tidr}, tid_0^0 - 1]$, all these $m\text{-}token$s exist in c_0 and they have the same $ring$ and new_set by (3) of definition 5. If $l = tid_0^0$, these $m\text{-}token$s either exist in c_0 or are generated in phase 0 with $ring = ring_0^0$ and $new\_set = new\_set^0$, so (2) is true in this case by (4) of definition 5; otherwise they are generated in the same phase after c_0 and (2) is still true. □

An execution in which all the configurations satisfy $token\_safe$ exhibits good behaviors. We state the conditions under which a processor's token ID remains unchanged in Lemma 11.

Lemma 11 *Let σ be an execution in which all the configurations are $token\_safe$. For each processor p_i and all $k \geq 0$, we have:*

(1) *If $(tid_i = tid_0)$ or $(tid_i = tid_0 - 1)$ in configuration c, then line 9.10 is $false$ in the $receive_i^m(t)$ event following c.*

(2) *If in phase k there exists a configuration c in which*

(2.1) $(status_i = joined) \wedge (tid_i = tid_0^k)$*, or*

(2.2) $(status_i = joined) \wedge (\ tid_i \in \{\ M_{tid}, 0, \ldots, M_{tidr}\ \}) \wedge$

$(tid_i \notin [tid_0^k - M_{tidr} - 1, tid_0^k - 1])$

then tid_i remains unchanged after c in phase k.

(3) *If in phase k there exists a configuration c before c_g^k in which $(status_i = joined) \wedge$ $(tid_i = tid_0^k - 1)$, then tid_i remains unchanged between c and c_g^k (including c_g^k).*

Proof. Proof of (1): If $t.id \neq 0$, line 9.10 is $false$. Otherwise since $t.id = 0 \in [tid_0 - M_{tidr}, tid_0]$ by definition 6, we have $tid_0 \in [0, M_{tidr}]$, that is, $tid_i \in \{M_{tid}, 0, \ldots, M_{tidr}\}$, so line 9.10 is $false$.

Proof of (2): If (2.1) is true in c, then line 9.2 is $false$, line 9.10 is $false$ by part (1), which was just proved, and line 9.13 is $false$ because no $m\text{-}token$ with ID $tid_0^k + 1$ exists by definition 6. If (2.2) is true in c, then line 9.2 and line 9.10 are $false$, and line 9.13 is

$false$ because all the $m\text{-}token$s have ID in $[tid_0^k - M_{tidr}, tid_0^k]$ by definition 6. So (2) is proved.

Proof of (3): If $status_i = joined$, then line 9.2 is $false$. If $tid_i = tid_0^k - 1$, then line 9.10 is $false$ by part (1), which has already been proved. Since no $m\text{-}token$ with ID tid_0^k exists before e_g^k in phase k by definition 6, line 9.13 is $false$ between c and c_g^k. So (3) is proved. □

Now we consider the generation of current tokens of phase $k \geq 1$ in such an execution.

Lemma 12 *Consider any execution σ in which all the configurations are $token\_safe$. For each phase $k \geq 1$ of σ, we have:*

(1) *All the current tokens of phase k are generated in phase k with the same $ring$ and new_set.*

(2) *The time interval between e_g^k and e_s^k is no more than M_{lt}, and there is no $m\text{-}token$ in the configuration previous to e_g^k if $(discon_0 = true)$ in e_s^k.*

Proof. Since $k \geq 1$, the c_s^k configuration exists, which is $token\_safe$. So there is no $m\text{-}token$ with ID tid_0^k in c_s^k by definition 6 and (1) is proved.

Now we prove (2). First we prove at least one $generate_0^m$ event occurs in phase k. Otherwise no $m\text{-}token$ with ID tid_0^k exists in phase k by part (1) (which was just proved), so $timer_0^{phase}$ is never reset in a $receive_0^m$ event and e_s^{k+1} is activated only when $timer_0^{phase}$ goes off. Thus the length of phase k is TM_{phase}. Since $timer_0^{gen}$ is set to 0 or to M_{lt}(which is less than TM_{phase}) in e_s^k, it goes off and a $generate_0^m$ event is activated in phase k, which is a contradiction.

Since $timer_0^{gen}$ is not reset between e_g^k and e_s^k, the time interval between e_g^k and e_s^k is equal to its reset value in e_s^k, which is M_{lt} if $(discon_0 = true)$ and 0 otherwise. If $(discon_0 = true)$, no $m\text{-}token$ exists in the configuration previous to e_g^k by Lemma 5. □

Now we consider membership maintenance in such an execution.

Lemma 13 *Consider an execution σ and phase $k \geq 1$ of σ, such that all the configurations in σ are $token\_safe$. The following are true for each $p_i \in ring_0^{k+1}$.*

(1) *There is a $receive_i^m(t)$ event, denoted by e_i^v, in phase k, such that $t.id = tid_0^k$ and p_i is added to $t.visited\_set$ in e_i^v. Furthermore, $status_i = joined$ in any configuration after e_i^v in phase k and phase $k+1$.*

(2) *If $p_i \notin ring_0^k$, then $tid_i = tid_0^k$ in c_s^{k+1}.*

Proof. By line 4.3, we have $p_i \in visited\_set_0$ in c_s^{k+1}. Notice $visited\_set_0$ is set to $\{0\}$ in e_s^k, and it is updated in phase k according to $visited\_set$ in the current tokens received by p_0, which are generated in phase k by Lemma 12. So there exists a current token which includes p_i in its $visited\_set$. Since every $m\text{-}token$'s $visited\_set$ is set to $\{0\}$ when generated, there exists such an e_i^v event in phase k. Right after e_i^v, $status_i$ is $joined$. The only code changing $status_i$ to $joining$ is a $send_i^j$ event. Since $timer_i^{join}$ is set to TM_{join} in e_i^v, after which $timer_i^{join}$ keeps going down unless it is reset to TM_{join} in a $receive_i^m$ event, it takes at least TM_{join} for $timer_i^{join}$ to reach 0. Thus a $send_i^j$ event will not occur within TM_{join} from e_i^v. Because $TM_{join} > 2TM_{phase}$, a $send_i^j$ event will not occur after e_i^v in phase k and phase $k+1$ by Lemma 7. So (1) is proved.

If $p_i \notin ring_0^k$, p_i can be added to $t.visited\_set$ only in line 9.5, so we have $tid_i = t.id = tid_i^k$ and $status_i = joined$ in the configuration following e_i^v. After that, tid_i remains unchanged in phase k by (2.1) of Lemma 11. So (2) is proved. □

3. Safe Properties of Non-member

In this section, we discuss the properties of non-members. First we define the appropriate levels of information related to a non-member processor according to how long ago the processor left the system (definition 7). Lemma 14 shows the time required to move from one level to the next. We prove in Lemmas 15 and 16 that eventually the information related to a non-member will be kept in an appropriate level and starting from that point, property $nmem\_safe$ (definition 8) is always guaranteed, that is, a processor rejoin is allowed to the system only when there is no old information relating to it still in the system. Thus we guarantee that when a processor rejoins the system, it cannot enter the critical section before it is initialized by a current token (Lemma 17).

First we give definitions of $conv_{nmem}$ and $nmem\_safe$.

Definition 7 *($conv_{nmem}$). Given a configuration c, c satisfies property $conv_{nmem}$ if and only if for every processor p_i with $(member_i = false)$ in c, we have:*

(1) *if* $(timer_i^{app} \leq 2M_{lt} + 2TM_{phase})$ *then* $(pred_1 = true)$

(2) *if* $(timer_i^{app} \leq M_{lt} + 2TM_{phase})$ *then* $(pred_2 = true)$

(3) *if* $(timer_i^{app} \leq M_{lt} + TM_{phase})$ *then* $(pred_3 = true)$

(4) *if* $(timer_i^{app} \leq M_{lt})$ *then* $(pred_4 = true)$

where

- $pred_1 \equiv (\, \nexists$ *m-token* t, $p_i \in t.visited\_set\,) \wedge (\nexists$ *j-token* t, $t.pid = p_i)$
- $pred_2 \equiv (p_i \notin visited\_set_0) \wedge (p_i \notin join\_set_0)$
- $pred_3 \equiv (p_i \notin ring_0) \wedge (p_i \notin new\_set_0)$
- $pred_4 \equiv \forall$ *m-token* t, $(p_i \notin t.ring) \wedge (p_i \notin t.new\_set)$

Definition 8 *($nmem\_safe$). A configuration is $nmem\_safe$ if and only if for every p_i with $(member_i = false) \wedge (timer_i^{app} = 0)$, $(pred_3 \wedge pred_4)$ is true for p_i.*

The level of information related to p_i represented by each predicate is decreasing. The following lemmas shows the time required to move from one level to the next.

Lemma 14 *Given $i > 0$ and any execution σ in which $(member_i = false)$ at time T, we have the following properties, where $T_a = T + M_{lt}$, $T_b = T_a + TM_{phase}$, $T_c = T_b + TM_{phase}$, and $T_d = T_c + M_{lt}$.*

(0) *$\forall i$, $1 \leq i \leq 4$, if $\wedge_{j=1}^{i} pred_j$ is true at T, then $\wedge_{j=1}^{i} pred_j$ remains true as long as $member_i$ remains $false$.*

(1) *If $member_i$ is $false$ in $[T, T_a]$, then $pred_1$ is true at T_a.*

(2) *If $pred_1$ is true at T_a and $member_i$ is $false$ in $[T_a, T_b]$, then $pred_2$ is true at T_b.*

(3) *If $(pred_1 \wedge pred_2)$ is true at T_b and $member_i$ is $false$ in $[T_b, T_c]$, then $pred_3$ is true at T_c.*

(4) *If $(pred_1 \wedge pred_2 \wedge pred_3)$ is true at T_c and $member_i$ is $false$ in $[T_c, T_d]$, then $pred_4$ is true at T_d.*

Proof. Suppose T_b is in phase k_b and T_c is in phase k_c. We notice that (a) p_i does not send any $j\text{-}token$ and p_i is not added to any $m\text{-}token$'s $visited\_set$ as long as $member_i$ remains $false$; (b) p_i can be added to $join\_set_0$ ($visited\_set_0$ respectively) only if there exists j-$token$ from p_i ($m\text{-}token$ t with $p_i \in t.visited\_set$ respectively); (c) p_i can be added to $ring_0$ (new_set_0 respectively) only if p_i has been added to $visited\_set_0$ ($join\_set_0$ respectively) ; (d) p_i is included in a $ring$ (new_set respectively) of a $m\text{-}token$ t only if $p_i \in ring_0$ ($p_i \in new\_set_0$ respectively) when t is generated.

If $pred_1$ is $true$ at T, it is $true$ as long as $member_i$ is $false$ by (a); if $(pred_1 \wedge pred_2)$ is $true$ at T, it is $true$ as long as $member_i$ is $false$ by (a) and (b); if ($pred_1 \wedge pred_2 \wedge pred_3$) is $true$ at T, it is $true$ as long as $member_i$ is $false$ by (a), (b) and (c); if ($pred_1 \wedge pred_2 \wedge pred_3 \wedge pred_4$) is $true$ at T, it is $true$ as long as $member_i$ is $false$ by (a), (b), (c) and (d). So (0) is proved.

Since all tokens existing at T have been discarded by T_a by Lemma 5, (1) can be proved by (a). Since event $e_s^{k_b}$ exists in $[T_a, T_b]$ by Lemma 7, in which $visited\_set_0$ and

$join\_set_0$ are reset, (2) can be proved by (b) because $pred_1$ is $true$ in $[T_a, T_b]$ by case (0). Since by Lemma 7 configuration $c_s^{k_c}$ exists in $[T_b, T_c]$, (3) can be proved by (c) because $pred_2$ is $true$ in $c_s^{k_c}$ by case (0). Since all the tokens existing at T_c have been discarded by T_d by Lemma 5, (4) can be proved by (d) because $pred_3$ is $true$ in $[T_c, T_d]$ by cases (0).

□

Lemma 15 *Starting from an arbitrary configuration, within time $2M_{lt} + 3TM_{phase}$, there is a configuration satisfying $conv_{nmem}$.*

Proof. We will prove the configuration c at time $T = 2M_{lt} + 3TM_{phase}$ from the initial configuration satisfies $conv_{nmem}$. Consider any processor p_i with $member_i = false$ in c. If there is no $leave_i$ event before c, then $member_i$ has remained $false$ for T time, so c satisfies $conv_{nmem}$ by (1), (2), (3), (4) and (0) of Lemma 14.

Otherwise let e be the last $leave_i$ event before c. In e, $timer_i^{app}$ is set to TM_{app}. Denote the value of $timer_i^{app}$ in c by T', so in c, $member_i$ has been $false$ for $T_f = TM_{app} - T'$ time. In c, if $T' \leq 2M_{lt} + 2TM_{phase}$, that is, $T_f \geq TM_{phase} > M_{lt}$, $pred_1$ is $true$ by (1) and (0) of Lemma 14; if $T' \leq M_{lt} + 2TM_{phase}$, that is, $T_f \geq M_{lt} + TM_{phase}$, $pred_2$ is $true$ by (1), (2) and (0) of Lemma 14; if $T' \leq M_{lt} + TM_{phase}$, that is, $T_f \geq M_{lt} + 2TM_{phase}$, $pred_3$ is $true$ by (1), (2), (3) and (0) of Lemma 14; if $T' \leq M_{lt}$, that is, $T_f \geq 2M_{lt} + 2TM_{phase}$, $pred_4$ is $true$ by (1), (2), (3), (4) and (0) of Lemma 14. □

Lemma 16 *Given any execution σ starting from a configuration c_0 satisfying $conv_{nmem}$, every configuration c in σ is $nmem\_safe$.*

Proof. Consider any processor p_i such that $member_i = false$ and $timer_i^{app} = 0$ in c. If there is a $leave_i$ before c in σ, then $member_i$ has been $false$ for at least $TM_{app} \geq 2TM_{phase} + 2M_{lt}$ time by c, so the claim is true by (1), (2), (3), (4) and (0) of Lemma 14. Otherwise, $member_i$ is $false$ between c and c_0. Let T be the value of $timer_i^{app}$ in c_0. Since

$timer_i^{app} = 0$ in c, $member_i$ has been $false$ for T time by c. If $T > 2TM_{phase} + 2M_{lt}$, the claim is proved by (1), (2), (3), (4) and (0) of Lemma 14. If $T \in [M_{lt} + 2TM_{phase}\ , 2M_{lt} + 2TM_{phase}\ ]$, then $pred_1$ is $true$ in c_0 by definition 7 and the claim can be proved by (2), (3), (4) and (0) of Lemma 14. If $T \in [M_{lt} + TM_{phase}, M_{lt} + 2TM_{phase}]$, then $(pred_1 \wedge pred_2\ )$ is $true$ in c_0 by definition 7 and the claim can be proved by (3), (4) and (0) of Lemma 14. If $T \in [M_{lt}, M_{lt} + TM_{phase}]$, then $(pred_1 \wedge pred_2 \wedge pred_3\ )$ is $true$ in c_0 by definition 7 and the claim can be proved by (4) and (0) of Lemma 14. If $T \in [0, M_{lt}]$, then $(pred_3 \wedge pred_4)$ is $true$ in c_0 by definition 7 and the claim can be proved by (0) of Lemma 14. $\square$

From the following lemma, we can see that eventually a processor not in $ring_0$ will not enter the critical section.

Lemma 17 *Given an arbitrary configuration c_0, consider any execution σ starting from c_0. Within time $T = 3M_{lt} + 5TM_{phase}$, there exists an $e_s^{k_c}$ event, $k_c > 1$, such that*

(1) all configurations after $e_s^{k_c}$ are $token\_safe$ and $nmem\_safe$, and

(2) given any $k \geq k_c$, in any configuration between e_g^k and e_g^{k+1}, for any $p_i \notin ring_0^k$, we have

(2.1) *$(member_i = false)$ or $(\forall t, p_i \notin t.ring)$, and*

(2.2) *p_i does not enter the critical section.*

Proof. By Lemmas 9, 10, after time $2M_{lt}$, all the configurations are $token\_safe$. By Lemmas 15 and 16, after time $2M_{lt} + 3TM_{phase}$, all the configurations are $nmem\_safe$. Thus after time $T' = 2M_{lt}{+}3TM_{phase}$, all the configurations are $token\_safe$ and $nmem\_safe$. By Lemma 7, there exists some $k_0 \geq 1$ such that the $e_s^{k_0}$ event occurs in $[T', T'']$ where $T'' = T' + TM_{phase}$, and k_c such that the $e_s^{k_c}$ event occurs in $[T'' + M_{lt}, T]$. Thus (1) is true for k_0 and k_c.

Now we prove (2.1) is true and we can get (2.2) directly from (2.1). Let k_1 be the smallest number such that $p_i \notin ring_0^l$ for any $l \in [k_1, k]$ (informally, phase k_1 to phase k is a maximal execution segment in which $p_i \notin ring_0$.) Let σ_1 be the execution of phase $k_1 - 1$ if $k_1 > k_0$, otherwise σ_1 is empty. Let σ_2 be the execution from $e_s^{max\{k_1,k_0\}}$ to e_g^{k+1}. Thus all the configurations in σ_1 or σ_2 are $token\_safe$ and $nmem\_safe$. We notice that (a) if σ_1 is not empty, that is $k_1 > k_0$, we have $p_i \in ring_0^{k_1-1}$ in σ_1 and there is no $join_i$ event in σ_1 by definition 8; (b) for any m-$token$ t generated in σ_2 we have $p_i \notin t.ring$; and (c) all the m-$token$s in phase k are generated after $e_s^{k_0}$ by Lemma 5. We have the following two cases.

- If there is a $join_i$ event e_{join} in σ_2, let c_{join} be the configuration previous to e_{join}, in which $(member_i = false) \wedge (timer_i^{app} = 0)$. By definition 8, there is no m-$token$ including p_i in $ring$ in c_{join}. So there is no m-$token$ including p_i in $ring$ in all the configurations after e_{join} in σ_2 by (b). If e_{join} occurs before e_g^k, the claim is proved. Otherwise e_{join} occurs after e_g^k. We notice the time interval between two $join_i$ and $leave_i$ events is at least TM_{app} because $timer_i^{app}$ is set to TM_{app} in a $leave_i$ event and a $join_i$ event occurs only if $timer_i^{app} = 0$. So there is no $leave_i$ event between e_g^k and e_{join} by Lemmas 7 and 12. Thus $member_i$ remains $false$ between e_g^k and e_{join} and the claim is proved in this case.
- Otherwise by (a) no $join_i$ event occurs in σ_1 and σ_2. If $member_i = false$ in the configuration previous to e_g^k, then $member_i$ remains $false$ in σ_2 and the claim is proved. Otherwise $member_i$ is always $true$ in σ_1 and σ_2. We consider the following two cases:
  - If σ_1 is empty, that is, $k_1 \leq k_0$, then σ_2 is the execution between $e_s^{k_0}$ and e_g^{k+1}. Thus all the tokens in phase k are generated in σ_2 by (c) and in phase k there is no token which includes p_i in $ring$ by (b). Thus (2.1) is proved in this case.

– Otherwise we have $k_1 \geq k_0 + 1$, so all the configurations in phase $k_1 - 1$ are $token\_safe$. Since $member_i$ is $true$ in phase $k_1 - 1$, we have $p_i \notin t.leave\_set$ for any current token t of phase $k_1 - 1$, which is generated in this phase by Lemma 12. Because $k_1 \geq k_0 + 1 \geq 2$, the $e_s^{k_1-1}$ event exists, in which $leave\_set_0$ is reset. Since $leave\_set_0$ is updated according to the $leave\_set$ on the current tokens received by p_0 in phase $k_1 - 1$, we have $p_i \notin leave\_set_0$ in $e_s^{k_1}$. Because $p_i \notin ring_o^{k_1}$, we have $p_i \notin visit\_set_0$ in $e_s^{k_1}$. So in $e_s^{k_1}$, $discon_0$ is $true$ and no $m\text{-}token$ exists in the configuration previous to $e_g^{k_1}$ by Lemma 12. So after e_g^{k-1}, all the $m\text{-}token$s are generated after $e_g^{k_1}$ and no $m\text{-}token$ includes p_i in $ring$ by (b). Thus the claim is proved.

□

4. Eventual Convergence

In this section, we define a property $conv_{member}$ of configurations. Based on Lemma 18, we show in Lemma 19 that, starting from an arbitrary configuration, there exists a phase k_c, such that $c_s^{k_c}$ satisfies $conv_{member}$ and all the configurations after $c_s^{k_c}$ are $token\_safe$ and $nmem\_safe$.

The intuition for $conv_{member}$ is that, in a configuration satisfying $conv_{member}$, all the members indicate they have joined the system by setting $status_i$ to $joined$, they have a correct token ID, which is equal to tid_0, and they have been visited recently, that is, $timer_i^{join} \geq TM_{phase} + M_{lt}$.

Definition 9 *($conv_{member}$). A configuration is $conv_{member}$ if and only if for any $p_i \in ring_0$, we have $(status_i = joined) \wedge (tid_i = tid_0) \wedge (timer_i^{join} \geq TM_{phase} + M_{lt})$.*

In Lemma 18, we show that, starting from a phase with $tid_0 = 0$, the range of the members' token IDs is decreased by one in each phase.

Lemma 18 *Consider an execution σ, in which all the configurations are $token\_safe$, and a phase b of σ, such that $tid_0^b = 0$ and $b \geq 1$. For any $p_i \in ring_i^{b+k+1}$, $0 \leq k \leq M_{tidr}$, we have $tid_i \in [k, M_{tidr}]$ in c_s^{b+k+1}.*

Proof. We have $tid_0^{b+k} = k$ and $tid_0^{b+k+1} = k + 1$ by Lemma 7. Given $p_i \in ring_i^{b+k+1}$, by Lemma 13 the e_i^v event exists in phase $b + k$ and we have $status_i = joined$ after e_i^v in phase $b + k$ and phase $b + k + 1$. Note that if we have $tid_i \in [k, M_{tidr}]$ in the configuration c_i^v following e_i^v, then tid_i remains unchanged in phase $b + k$ by (2.2) of Lemma 11. So in the rest we only need to prove $tid_i \in [k, M_{tidr}]$ in c_i^v. We prove it by induction on k.

In phase b, if p_i is added to $t.visited\_set$ by line 9.5 in e_i^v, we have $tid_i = t.id = 0 \in [0, M_{tidr}]$ in c_i^v. Otherwise we have $tid_i \in \{M_{tid}, 0, \ldots, M_{tidr}\}$ at line 9.13 because $t.id = 0$, after which tid_i is updated to 0 if $tid_i = M_{tid}$ by lines 9.13-9.15. So we have $tid_i \in [0, M_{tidr}]$ in c_i^v. Thus the claim is true for $k = 0$.

Assume the claim is true for l, $0 \leq l \leq k-1$. We show it holds for k. If $p_i \notin ring_0^{b+k}$, we have $tid_i = tid_0^{b+k} = k \in [k, M_{tidr}]$ in c_s^{b+k+1} by Lemma 13. Otherwise we have $p_i \in [k - 1, M_{tidr}]$ in c_s^{b+k} by the inductive hypothesis, and $status_i = joined$ in phase $b + k$ by Lemma 13. Note if $tid_i \in [k - 1, M_{tidr}]$ is true in the configuration previous to a $receive_i^m(t')$ event in phase $b + k$, then it is true at the end of this event for the following reason. Since line 9.2 and 9.10 are $false$, tid_i can be changed only when line 9.13 is true. Since we have $tid_i \in [k - 1, M_{tidr}]$, $t'.id \in [k - M_{tidr}, k]$ by definition 6, and $[k, M_{tidr} + 1] \cap [k - M_{tidr}, k] = \{k\}$, line 9.13 is true only if $t'.id = tid_i + 1 = k$, and tid_i is updated to $k \in [k - 1, M_{tidr}]$ if it is true.

Since we have $tid_i \in [k - 1, M_{tidr}]$ in c_s^{b+k}, $tid_i \in [k - 1, M_{tidr}]$ is true in the configuration previous to all $receive_i^m$ events in phase $(b + k)$, including e_i^v. Since we have $t.id = k$ in e_i^v, tid_i is updated to k in line 9.14 if $tid_i = k - 1$. So we have $tid_i \in [k, M_{tidr}]$ in c_i^v, and the claim is true for k. □

Theorem 19 *Given an arbitrary configuration c_0, consider any execution σ starting from c_0. Within time $2M_{lt} + 3TM_{phase} + (M_{tid} + M_{tidr}) \cdot TM_{phase}$, there exists an $e_s^{k_c}$ event, for some $k_c > 1$, such that*

(1) all configurations after c_s^t are $token\_safe$ and $nmem\_safe$ and

(2) $c_s^{k_c}$ satisfies $conv_{member}$.

Proof. By Lemmas 9, 10, after time $2M_{lt}$, all the configurations are $token\_safe$. By Lemmas 15 and 16, after time $2M_{lt} + 3TM_{phase}$, all the configurations are $nmem\_safe$. Thus after time $T = 2M_{lt}+3TM_{phase}$, all the configurations are $token\_safe$ and $nmem\_safe$.

By Lemma 7, after T there exists a phase b such that $tid_0^b = 0$ and $b \geq 1$. Let m be $b + M_{tidr}$. We will prove $c_s^{k_c}$ satisfies $conv_{member}$, where $k_c = m + 1$. By Lemmas 7, 9 and 15, we can see the time of $e_s^{k_c}$ is no more than $2M_{lt} + 3TM_{phase} + (M_{tid} + M_{tidr}) \cdot TM_{phase}$ after the initial configuration.

For any $p_i \in ring_0^{m+1}$, by Lemma 13 the e_i^v event exists in phase m and we have $status_i = joined$ after e_i^v in phase m and phase $m + 1$, including c_s^{m+1}. By Lemma 18 we have $tid_i = M_{tidr} = tid_0^m$ in c_s^{m+1}. Since $timer_i^{join}$ is set TM_{join} in e_i^v, we have $timer_i^{join} \geq TM_{join} - TM_{phase} \geq TM_{phase} + M_{lt}$ in c_s^{m+1} by Lemmas 7 and 12. So c_s^{m+1} satisfies $conv_{member}$. □

In the sequel, we say an execution has *converged* if it has reached a configuration $c_g^{k_c}$, where k_c satisfies the properties in Lemma 17 and Theorem 19.

5. Mutual Exclusion after Convergence

In this section we focus on an execution σ which has converged. We denote by σ^k the subsequence between c_g^k and c_g^{k+1} of σ, including c_g^k and c_g^{k+1} (Fig. 19). By Lemma 17, we know that only processors in $ring_0^k$ can enter the critical section in σ^k. In this section,

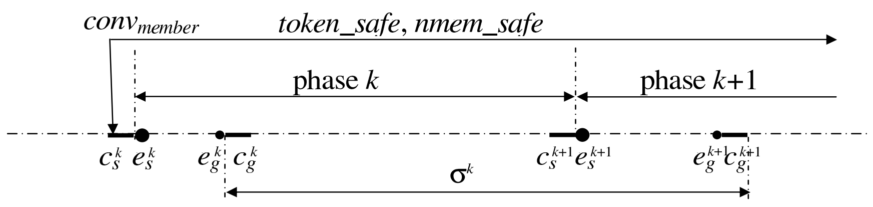


Fig. 19. Proof of $Mutual\ Exclusion$ after convergence

we show that if c_s^k satisfies $conv_{member}$, then no more than one processor is in the critical section at the same time in σ^k (Lemma 20), and c_s^{k+1} satisfies $conv_{member}$ (Lemma 21). Thus $Mutual\ Exclusion$ can be guaranteed from c_g^k (Theorem 22).

Lemma 20 *Given an execution σ and $k > 1$, such that c_s^k satisfies $conv_{member}$, and all the configurations after c_s^k are $token\_safe$ and $nmem\_safe$, we have in σ^k:*

(1) *For any $p_i \in ring_0^k$,*

(1.1) *p_i is in the critical section only when being visited by an m-$token$ with ID tid_0^k the first time in σ^k.*

(1.2) *if p_i never enters the critical section in σ^k, tid_i remains unchanged. Otherwise let c_{cs} be the last configuration in σ^k in which p_i is in the critical section. We have $tid_i = tid_0^k - 1$ in all configurations in σ^k previous to c_{cs} and including c_{cs}, and $tid_i = tid_0^k$ in all configurations in σ^k after c_{cs}.*

(2) *No more than one processor in $ring_0^k$ is in the critical section at the same time in any configuration in σ^k.*

Proof. By Definition 9, we have $status_i = joined$ and $tid_i = tid_0 = tid_0^k - 1$ in c_s^k. First we notice $status_i$ remains $joined$ in σ^k for the following reason: $status_i$ is set to $joining$ only in a $send_i^j$ event, which will not occur in σ^k because $timer_i^{join}$ will not reach 0 in σ^k by Definition 9 and Lemmas 7 and 12. We denote the first event that updates tid_i in σ^k by e_i^u. By (3) of Lemma 11, tid_i remains unchanged before c_g^k. So we have $tid_i = tid_0^k - 1$ in

c_g^k and line 9.10 is $false$ in e_i^u by (1) of Lemma 11. Since line 9.2 is always $false$ because $status_i = joined$, line 9.13 is $true$ in e_i^u, that is, p_i is visited by an $m\text{-}token$ with ID tid_0^k and tid_i is updated to tid_0^k by line 9.14 in e_i^u. Since line 9.13 is always $true$ when p_i is visited by an $m\text{-}token$ with ID tid_0^k if $(tid_i = tid_0^k - 1)$, this is the first time p_i is visited by an $m\text{-}token$ with ID tid_0^k. Once tid_i is updated to tid_0^k, tid_i remains unchanged in phase k by (2.1) of Lemma 11 and in phase $(k+1)$ before e_g^k by (3) of Lemma 11. Thus (1) is proved.

Now we prove (2). Assume in contradiction that p_i and p_j are in the critical section at the same time, say in configuration c. Let $m\text{-}token$ t_i (t_j respectively) be the first token in $queue_i$ ($queue_j$ respectively) in c. By part (1.1), which was just proved, t_i (t_j respectively) is the first token with ID tid_0^k which visits p_i (p_j respectively). Since any $m\text{-}token$ with ID tid_0^k is generated in phase k by Lemma 12, we have $t_i.ring = t_j.ring = ring_0^k$. By the predicate for the critical section, we also have in c: (a) p_i is the first processor in $t_i.ring$ not in $t_i.visited\_set$, and (b) p_j is the first processor in $t_j.ring$ not in $t_j.visited\_set$. Without loss of generality, we assume p_i is before p_j in $ring_0^k$, thus we have $p_i \in t_j.visited\_set$ by (b). So p_i is visited by t_j before p_i is visited by t_i. Contradiction! □

Lemma 21 *Given an execution σ, if there exists a phase $k > 1$, such that c_s^k satisfies $conv_{member}$, and all the configurations after c_s^k are $token\_safe$ and $nonmember_{safe}$, then c_s^{k+1} satisfies $conv_{member}$.*

Proof. Given any $p_i \in ring_0^{k+1}$, by Lemma 13 we have an event e_i^v in phase k such that $status_i = joined$ after e_i^v in phase k and phase $k+1$, including c_s^{k+1}. Since $timer_i^{join}$ is set to $TM_{join} \geq 2TM_{phase} + 2M_{lt}$ in e_i^v, we have $timer_i^{join} \geq TM_{phase} + M_{lt}$ in c_s^{k+1} by Lemmas 7 and 12. Now we consider tid_i in c_s^{k+1}. If $p_i \notin ring_0^k$, we have $tid_i = tid_0^k$ in c_s^{k+1} by Lemma 13. Otherwise we have $p_i \in ring_0^k$. Since p_i is visited by a current token in e_i^v in phase k, we have $tid_i = tid_0^k$ in c_s^k by (1.2) of Lemma 20. So c_s^{k+1} satisfies $conv_{member}$.

□

By Lemmas 17, Lemma 20 and Lemma 21, we have:

Theorem 22 *$Mutual\ Exclusion$ is satisfied when an execution has converged.*

6. Progress

Since one of the conditions for processor p_i to enter the critical section is being visited by a token, the level of progress provided by this algorithm depends on the frequency with which a processor is visited by tokens, that is, LR_L's performance for token circulation. The performance of LR_L is affected by many factors, including message loss, choice of L, and the mobility pattern of the network. In this section, we discuss the different levels of requirements on LR_L's performance to achieve different levels of progress. We define the following levels of LR_L's performance, in which R_0 is required to guarantee that p_0 can receive p_i's $join\_request$ eventually, $R_1(i)$ is used to guarantee that p_i enters the critical section infinitely often, and R_2 is used to guarantee $Bounded\ Waiting$.

- $R_0(i)$: If infinitely many tokens are generated by p_i, then infinitely many of them arrive at p_0 within L hops.
- $R_1(i)$: If infinitely many tokens that include p_i in $ring$ or new_set are generated by p_0, then infinitely many of them visit p_i and come back to p_0 within L hops.
- R_2: At least one of every $(TM_{phase} - M_{lt}) \cdot g$ tokens generated by p_0 visits all the processors in their new_set and $ring$ within L hops.

Lemma 23 *Consider phase k of any execution which has converged. Let T^k be the time of event e_s^k. Let processor p_i and m-token t_m be such that t_m is generated in $[T^k, T^k + TM_{phase} - M_{lt}]$, p_i is in $t_m.new\_set$ or $t_m.ring$ and t_m visits p_i and comes back to p_0*

within L hops. Then we have $p_i \in ring_0^{k+1}$ and p_i enters the critical section in phase k if $p_i \in t_m.ring$.

Proof. By Lemma 12 any current token t of phase k, including t_m, satisfies $(t.ring = ring_0^k) \wedge (t.new\_set = new\_set_0^k)$. By Lemma 5, token t_m comes back to p_0 after visiting p_i within $[T^k, T^k + TM_{phase}]$. Since phase k is ended only if a current token comes back to p_0 after visiting all the processors in $new\_set_0 \cup ring_0$, including p_i, or phase k has lasted for time TM_{phase}, p_i is visited by a current token and added to $visited\_set_0$ in phase k. So we have $p_i \in ring_0^{k+1}$ and p_i enters the critical section if $p_i \in t.ring$ by (1.1) of Lemma 20.

□

Theorem 24 *Consider any execution that has converged,* (1) *If* $R_0(i)$ *and* $R_1(i)$ *are satisfied for some processor* p_i*, then* $No\ Deadlock$ *is guaranteed;* (2) *if* $R_0(i)$ *and* $R_1(i)$ *are satisfied for every processor* p_i*, then* $No\ Lockout$ *is guaranteed;* (3) *if* $R_0(i)$ *is satisfied for every processor* p_i *and* R_2 *is satisfied, then* $Bounded\ Waiting$ *is guaranteed with waiting time bounded by* TM_{phase}.

Proof. First we notice that if $R_0(i)$ and $R_1(i)$ are $true$, then p_i is added to $ring_0$ or new_set_0 in infinitely many phases for the following reason: If it is not true, p_i stops being visited by tokens eventually and $timer_i^{join}$ will not be reset to a non-zero value, so infinitely many $send_i^j$ events are activated, in which infinitely many $j\text{-}token$s are sent to p_0, thus infinitely many $j\text{-}token$s arrive at p_0 by R_0 and p_i is added to new_set_0 in infinitely many phases, which is a contradiction.

Note in any phase such that $p_i \in new\_set_0$, at least one token that includes p_i in new_set is generated during $[T, T + TM_{phase} - M_{lt}]$, where T is the time when this phase starts. From the discussion above, we know there are infinitely many phases such that $p_i \in new\_set_0$, so we have $p_i \in ring_0$ in infinitely many phases by $R_1(i)$ and Lemma 23. Note in each of the infinitely many phases such that $p_i \in ring^0$, at least one token that includes $p_i \in ring$ is generated during $[T, T + TM_{phase} - M_{lt}]$, where T is the time the phase starts. So p_i enters the critical section infinitely often by Lemma 23 and $R_1(i)$. Thus (1) and (2) are proved.

Now we consider $Bounded\ Waiting$. In each phase, p_0 keeps generating tokens. By R_2, at least one token which is generated within time $TM_{phase} - M_{lt}$ from the beginning of this phase visits all the members and comes back to p_0 within L hops, so every processor enters the critical section in this phase by Lemma 23. By Lemma 3, the waiting time is bounded by TM_{phase}. □

H. Remarks

The assumption of a distinguished processor can be relaxed by using a self-stabilizing leader election algorithm for mobile ad hoc networks. ([26] is a first step toward the development of such an algorithm, in which a "weak" self-stabilizing algorithm is presented.)

The assumption that a processor stays in the critical section only for a negligible time can be relaxed by replacing d by $d + C$ in the definition of M_{lt} in Fig I, where C is the maximum time a processor stays in the critical section. No change is needed in the analysis, since the analysis is done in terms of M_{lt}.

Theorem 19 implies that the time for the algorithm to converge is $O(L^2)$ and the waiting time implied by Theorem 24 is $O(L)$.

Simulation results in [5] [6] show that the (unbounded) LR algorithm performs well in mobile networks. In particular, the round length is very close to the optimal value n, where round length is the number of hops for a token to be forwarded to all the nodes and n is the number of nodes in the network. If the parameter L to our bounded version of LR is chosen to be on the order of the "usual" round length, then the conditions for progress (R_0, R_1, and R_2) are likely to be satisfied. With such a value of L, namely $L = O(n)$, the simulation results indicate that $O\left(n^2\right)$ and $O(n)$ are reasonable approximations to the convergence time and waiting time, respectively. Thus we can see that the requirements on LR's performance to guarantee different levels of progress are feasible in practice.

Table III. Fields on tokens and variables at processors

Token	Field	Data Type	Explanation
j-$token$	ts	$timestamp$ array	used by LR_L
	pid	integer in $[0, N]$	id of sender
m-$token$	ts	$timestamps$ array	used by LR_L
	id	$token\_id$	token ID
	$ring$	permutation of subset of $[0, N]$	processors that have joined
	new_set	subset of $[0, N]$	joining processors
	$visit\_set$	subset of $[0, N]$	processors visited
	$route\_list$	permutation of subset of $[0, N]$	the order in which the token was routed
	$leave\_set$	subset of $[0, N]$	leave requests

Processor	Variable	Data Type	Explanation
p_0	tid_0	$token\_id$	p_0's local token ID
	$ring_0$	permutation of subset of $[0, N]$	processors that have joined
	new_set_0	subset of $[0, N]$	joining processors
	$visit\_set_0$	subset of $[0, N]$	processors visited
	$route\_list_0$	permutation of subset of $[0, N]$	the order in which the token was routed
	$join\_set_0$	subset of $[0, N]$	join requests
	$leave\_set_0$	subset of $[0, N]$	leave requests
	$discon_0$	boolean	indicates whether there exist partitions
	$timer_0^{gen}$	$timer$	for token generation
	$timer_0^{phase}$	$timer$	for starting a new phase
p_i, $i \neq 0$	tid_i	$token\_id$	p_i's local token ID
	$member_i$	boolean	indicates whether p_i is a member
	$status_i$	$joined/joining$	indicates whether p_i has joined
	$timer_i^{join}$	timer	for resending j-$token$s
	$timer_i^{app}$	timer	time between application requests

CHAPTER VI

LOCATION BASED BROADCASTING

In this chapter, we focus on the *broadcasting problem* in dense mobile ad hoc networks. Broadcasting is one of the fundamental tasks in network communication. Its goal is to transmit a message from one node to all other nodes [1] Recent advances in wireless communication make it possible to envision large scale dense ad-hoc networks, which are characterized by a large number of energy-constrained, unattended nodes [48]. Examples include the Smartdust project [52], Dataspace [53, 54], and sensitive skin [55].

Most of the protocols designed for mobile networks depend on the communication topology and its unpredictability prevents the protocols from providing reliable services. Usually a stable base is required for a protocol to provide reliable services. In spite of the fact that the topology keeps changing in a mobile ad hoc network, a specific network may have some properties which can provide stable information for protocol design. Our approach to handle mobility is to reduce the impact of the changing topology by basing the protocol on one stable aspect of a network. In a dense network, mobile nodes keep moving but their distribution is fairly stable. Instead of relying on the frequently changing communication topology, our protocols depend on the *distribution* of mobile nodes.

Most communication protocols specify the procedure of message propagation by determining sequences of *nodes* to forward the message. Usually global or local node identification is required in these protocols. However, in a mobile ad hoc network, global identification may not be available. Furthermore, since topology keeps changing, local identification usually requires sending hello beacons periodically to get the latest neighbor information, which wastes bandwidth and introduces collisions. Given the assumption

[1]In this chapter, we use the term "broadcast" to mean dissemination of a message to all nodes in the system and not to refer to a physical radio transmission.

that the distribution of mobile nodes is stable in a dense network, our protocols achieve broadcasting by covering the whole area occupied by mobile nodes; instead of node identifications, the procedure of message propagation is specified by determining sequences of *locations* which are provided by GPS [56] or a location service [57, 58].

Location information has been used in message forwarding protocols for mobile ad hoc networks. The most important differences between our approach and the existing position-based protocols are that, in our approach the location-based paths are decided completely *online* during message forwarding and *no knowledge about the distribution of mobile nodes is required a priori*. So the selection of the next location can reflect the latest system information, including distribution of mobile nodes, obstacles, and the battery states of mobile nodes. Furthermore, we do not require knowledge of the neighborhood topology in the selection of the next location and the expected number of messages transmitted in each selection of the next location is a constant if the distribution of mobile nodes satisfies some properties. These properties are desirable in mobile ad hoc networks, in which global knowledge might not be available and the system keeps changing.

An important characteristic of the wireless medium is that when a node transmits, all the nodes in its transmission range might receive this message if no collision occurs, depending on the reliability of the transmission. This property is usually considered as a source of extraneous energy consumption [59, 60]. The argument is that when a node transmits a packet, all nodes in the radio range of the transmitter must receive each packet to determine whether to retransmit; although most of these packets are immediately discarded, they still consume energy [60]. However, this property also means that one broadcasting covers multiple nodes. Our approaches use this property to speed up the message propagation and reduce the number of transmissions.

Our basic idea to achieve broadcasting is to forward a packet along a set of *location-based* paths. By "location-based path", we mean the path is determined by a sequence of locations, instead of node identifications. Our approach consists of two modules: *Message Forwarding* and *Path Selection*: the *Path Selection* module dynamically selects the next location, specified by a destination area, and the *Message Forwarding* module provides service to forward the message to the next location, One approach is proposed for the Message Forwarding module under the assumption that each node is able to detect collisions; a discussion on how to relax this assumption is also provided. Our approach to forward the message to the next location relies on the *distribution of nodes* instead of network topology, and neither *neighbor information* nor *global identification* is required. Furthermore, the number of steps in forwarding a message to a $u \times u$ area is $O(\log m)$, where m is the maximum number of mobile nodes that can be in a $u \times u$ area, and the expected number of steps is a constant under certain constraints on distribution of nodes. We propose two approaches for the Path Selection module. In both approaches, the selection of the location-based paths is done online. For each approach, we provide specific constraints on distribution and mobility of mobile nodes to guarantee that all the nodes receive the broadcast data. Under these constraints, given a network with area $\mathcal{A}$, our approaches achieve broadcasting in $O\left(\mathcal{A}/R^2\right)$ steps, where R is the transmission range.

No fully reliable broadcasting service can be guaranteed in networks with arbitrary mobility. In this chapter, we discuss constraints on mobility, specified by the maximum speed of mobile nodes, required by each approach to guarantee that all the nodes receive the broadcast data. The reason to choose the maximum speed to define the constraints is that, in practice the speed of mobile nodes is more predictable than the other mobility parameters. For example, a reasonable speed restriction of surface vehicles is 120 miles per hour and the usually speed of a commercial jet airplane is about 250 meters per second. Note we do not put any restriction on any other mobility parameter.

A. System Model — Dense Mobile Ad Hoc Network

We focus on *dense* mobile ad hoc networks. We assume each node has a means to determine its own location, which can be met either by GPS service [56] or some other location service [57, 58]; such assumption is common in position-based algorithms [43, 44, 45, 46, 48, 60, 14].

1. Communication and Computation Model

We assume all the devices have the same transmission range R. A device receives message from the other if and only if it is in the sender's transmission range. We assume the network is always connected. Communication connectivity between mobile devices can be described by a graph, in which each node represents a mobile device and there exists edge between two nodes if and only if the corresponding mobile devices can communicate. We assume a synchronous communication model in which the communication is structured into *time-slots*, achievable in practice by equipping mobile nodes with local GPS receivers [56]. We denote the upper bound on the time interval of each synchronous step by δ. This paradigm is commonly employed in the practical design of protocols for radio communication [61, 62, 63, 32]. Two communication primitives, $receive$ and $transmit$, are provided by the basic communication service, which has a mechanism to *detect collisions* (Fig 20.) Execution is partitioned into steps. Each step consists of three phases: *reception*, *computation* and *transmission*. In *reception phase*, each node receives none or one message, depending on the number of nodes in its neighborhood which transmitted in the last step. In the *computation phase*, each node updates its state based on its state at the end of the last step and what occurred in the reception phase of the current step. In the *transmission phase*, a node can transmit a message msg; the decision whether to transmit and what to transmit is based on its current state. Protocols specify each node's behavior in the compu-

- $receive()$:
  - $\emptyset$: no message is received if no neighbor transmitted in the last step.
  - C: collision is detected if more than one neighbor transmitted in the last step.
  - $\textbf{Message}\ msg$: message msg is received if the sender of msg is the only neighbor which transmitted in the last step.
- $transmit(\textbf{Message}\ msg)$: message msg is transmitted.

Fig. 20. Communication primitives

tation and transmission phase in the forms of *function* and *events*. The action of an event is executed on a node if and only if the associated predicate is true.

In order to simplify the presentation, we use the model with the assumption that each node has the ability to detect collisions. This assumption can be relaxed and it is only used in the Message Forwarding module: in section VI.D, we propose an approach under this assumption and then discuss how to relax it. Approaches designed for the Path Selection module do not rely on this assumption directly; they are based on the services provided by the Message Forwarding module.

We assume that all the nodes in the transmission range of the sender receive the message if no collision occurs. This property might not hold all the time due to the unreliable wireless channels. However, the probability of reliable transmssion can be made quite large using mechanisms such as acknowledgement/retransmission. The discussion of improving transmission reliability is outside scope of our work. As is done in other works in this area (e.g. [32]), we assume that the transmission is reliable if no collision occurs.

2. Mobility and Distribution of Mobile Nodes

We assume the movement of mobile devices is restricted to a square network area with size $\mathcal{A}$. Network area can be divided into two parts: the *covered area* and the *uncovered area*, where the covered area is the union of all the positions which are within some node's transmission range. We assume the algorithm knows the network area *a priori* and has no knowledge about covered area and uncovered area, which may keep changing due to failures and mobility. In the sequel, we denote the total number of nodes by n.

Nodes are placed in a network according to some distribution function. We define D as $\sqrt{\frac{A}{n}}$. Intuitively, $D \times D$ is the average area occupied by one node. Note the larger D is, the less dense the network is. This parameter is used in Theorem 27. In this chapter, we are interested in the probability of having more than one node in a given area, which is decided by the distribution function of nodes placement. We define a function $V(x, y)$ below. Note $V(x, y)$ is non-decreasing in the sense that $V(x, y) \leq V(\max(x, y), \max(x, y))$. We denote $V(x, x)$ by $V(x)$.

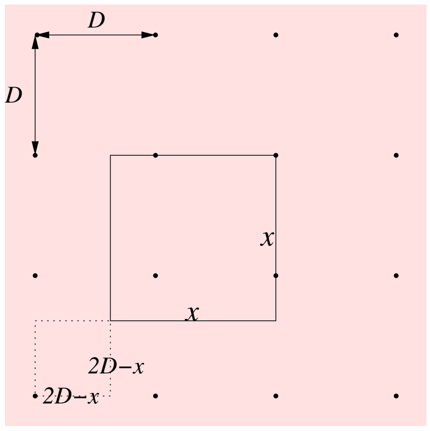
$$V(x, y) \quad = \quad \text{the probability of having more than one node in any } x \times y \text{ area.}$$

An example is a grid distribution of mobile nodes (Fig 21): nodes are distributed on a grid graph in which each grid is a D by D square. In Fig. 21, we see that any x by x area (the solid square), where $D \leq x \leq 2D$, covers exactly one node if and only if its southwest corner is not in the southwest $(2D - x)$ by $(2D - x)$ square of a grid (the dotted square). So we have $V(x)$ for this distribution as follows:

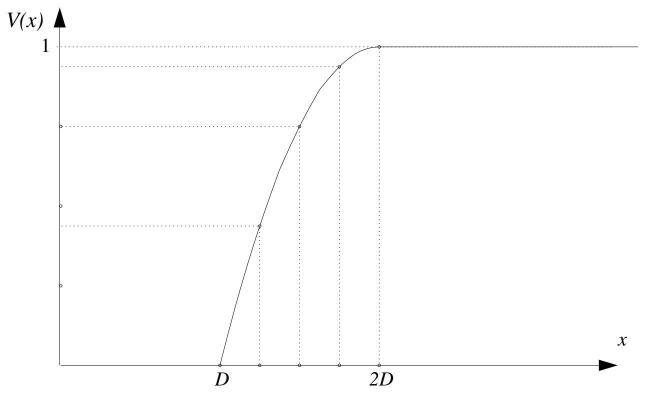
$$V(x) = \begin{cases} 0 & \text{if } x < D; \\ 1 - \frac{(2D-x)^2}{D^2} & \text{if } D \leq x \leq 2D; \\ 1 & \text{if } 2D < x. \end{cases}$$

3. Problem Definition and Protocol Stack

We focus on *broadcasting* protocols that propagate a message from a node, called *source*, to all of the nodes of a network. We define *time complexity* of a broadcasting protocol as the interval between the time the source node starts transmitting a broadcast message and the



(a) Grid distribution of nodes



(b) $V(x)$ for grid distribution

Fig. 21. An example of $V(x)$: Grid distribution of nodes

last time that a node takes an action in the broadcasting protocol. We define the *coverage* as the fraction of nodes which receive the broadcast message when the execution of the protocol ends.

The architecture of the protocol stack of each mobile node is presented in Fig. 22. An application accesses the broadcasting service by calling the function $broadcast(\mathbf{Data}\ data)$ and receives the broadcast data in event $broadcast\_recv(\mathbf{Data}\ data)$. Function $broadcast$ and the precondition of event $broadcast\_recv$ are specified by the broadcasting service. Our approaches rely on the basic communication service through primitives $transmit($ $\mathbf{Message}\ msg)$ and $receive()$. Each node gets its position at time t through the function $location(\mathbf{int}\ t)$ provided by the location service.

B. Overview of Our Approaches

In our approaches, the area of the network is divided into cells, based on which a virtual graph is constructed — a virtual vertex represents an existing cell and a virtual edge between two vertices exists if and only if the corresponding cells are adjacent. Broadcasting

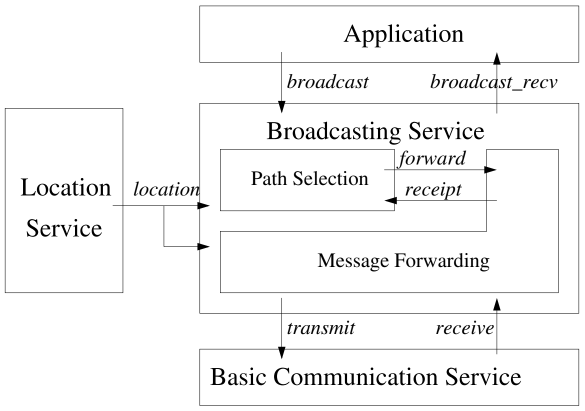


Fig. 22. Protocol stack

is achieved by propagating messages throughout the virtual graph, which is started by the vertex corresponding to the cell in which the source node resides. The proposed broadcasting approaches are organized in two modules: *Path Selection* and *Message Forwarding* (see Figure 22.)

- The *Message Forwarding* module provides the services of sending ($forward$) and receiving messages ($receipt$) among cells. It also notifies the application that broadcast data has arrived.
  - When function $forward(\textbf{PMessage}\ m,\ \textbf{Cell}\ s\_cell,\ \textbf{Cell}\ d\_cell)$ is called by the Path Selection module at node p, a PMessage m is sent by node p, on behalf of cell s_cell, to cell d_cell; the format of PMessage is defined by the *Path Selection* module, which includes the broadcasting data.
  - The *Message Forwarding* module notifies the Path Selection module that a PMessage message has arrived by triggering event $receipt(\ \textbf{PMessage}\ m,\ \textbf{Cell}\ from\_cell)$. When $forward(\textbf{PMessage}\ m,\ \textbf{Cell}\ s\_cell,\ \textbf{Cell}\ d\_cell)$ is called, if there exist nodes in d_cell, exactly one node in d_cell will be selected on which event $receipt(\textbf{PMessage}\ m,\ \textbf{Cell}\ from\_cell)$ occurs, and

the selected node will be responsible for further communication. When this event occurs on node p, we say the cell in which p resides receives m from cell $from\_cell$ and p is the *representative* node of d_cell.

  – The Message Forwarding module also notifies the application that broadcast data has arrived by triggering event $broadcast\_recv(\mathbf{Data}\ data)$.

- The purpose of the *Path Selection* module is to dynamically select a set of paths which cover the whole virtual graph. This is achieved by specifying a subset of neighboring cells to forward broadcast data when a cell starts broadcasting (function $broadcast$ is called) or receives broadcast data from others (event $receipt$ occurs). When a subset of neighboring cells is selected, function $forward$ is called to forward messages to the selected cells.

In our approaches, several types of messages are defined. Each message has a field indicating its type, followed by the data fields. We denote a message in the form of $msg($ $Type, data_1, data_2, \dots)$. Given a message msg, we use the denotation $msg.\mathbf{FieldName}$, where $\mathbf{FieldName}$ is the name of a field defined in the message definition, to retrieve the value of data field $\mathbf{FieldName}$ in msg.

In the sequel, first we introduce the construction of the virtual graph. Then we present a protocol for the Message Forwarding module. An analysis of this protocol is given. After that we propose two approaches, denoted by $\mathbf{DFS}$ and $\mathbf{SF}$, for the Path Selection module. We present specific constraints on distribution and mobility of mobile nodes for each approach to guarantee coverage. Sample values of the constraints are provided. Complexity analysis are presented under these constraints. We present a comparison of $\mathbf{DFS}$ and $\mathbf{SF}$ in Table B, where R is the radio range, $\mathcal{A}$ is the network area and δ is the time interval of each step in a synchronous system:

Table IV. Comparion of two approaches

Approach	Time	Mobility Constraint (Maximum Speed)
DFS	$O\left(\frac{\mathcal{A}}{R^2}\delta\right)$	$O\left(\frac{R^3}{\delta\mathcal{A}}\right)$
SF	$O\left(\frac{\mathcal{A}}{R^2}\delta\right)$	$O\left(\frac{R}{\delta}\right)$

Both approaches achieve $O(\mathbf{A}\delta/R^2)$ time complexity. The idea of $\mathbf{DFS}$ is simple but the allowed maximum speed when mobile nodes run $\mathbf{DFS}$ decreases as the network area increases, while the constraint on mobility for $\mathbf{SF}$ does not depend on the network area.

C. Construction of Virtual Graph

In our approaches, the network area is divided into cells; each cell is a square with size $u \times u$. For simplicity, we assume the number of cells is $\mathcal{A}/u^2$. The value of u depends on the transmission range R, the upper bound v on a node's speed and value of a parameter Γ defined below; for each of our approaches, we present how to select the value of u based on the values of v and Γ in section VI.E.1 and section VI.F.1 respectively, and we discuss the constraints on Γ and v to guarantee the coverage in section VI.E.3 and section VI.F.4 respectively. Note v and Γ are defined for the worst-case behavior.

We enumerate eight directions of each cell by $Dirs =$ [North, Northeast, East, Southeast, South, Southwest, West, Northwest]. Each cell has at most one neighbor in each direction. We then construct a virtual graph by assigning a virtual vertex for each cell in which there exist nodes, and adding a virtual edge between two virtual vertices if and only if the corresponding cells are neighbors (Fig 23). We denote by cell $[x, y]$ the cell whose center is $(x + 1/2) \cdot u$ from the north boundary and $(y + 1/2) \cdot u$ from the west boundary of the network area. Given the network area, u and location service, each node p knows the cell in which it resides at a given time t, denoted by $cell(p,t)$. We denote the neighbor of a given cell C in direction $dir \in Dirs$ by $neighbor(C, dir)$ and the neighbors in all directions of C by $neighbors(C)$.

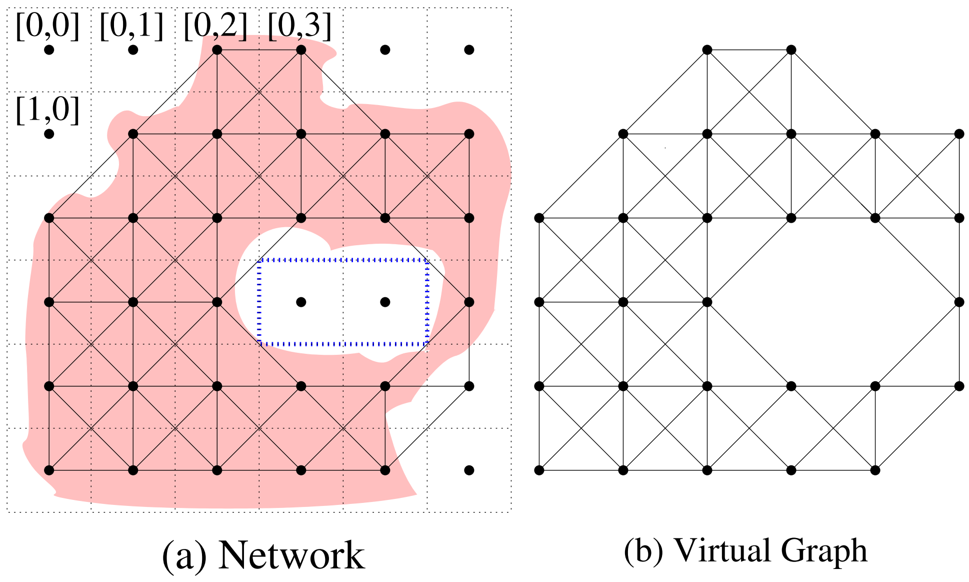


Fig. 23. Virtual graph of a network

Each time a cell communicates with its neighbors, a node residing in this cell is selected as *the representative node*. The Path Selection module will specify a subset of neighbors the cell needs to contact and the representative node communicates with each selected neighbor through function $forward$ provided by the Message Forwarding module. Let Γ be the *maximum* time it takes for a representative node to communicate with all the selected neighbors. We require Γ to be the *maximum* time instead of the *expected* time because this value is required in calculating the maximum speed of mobile nodes to guarantee coverage. Note Γ is a finite value if mobile devices are not infinitely small, since the number of steps taken by each forwarding is finite, as shown in section VI.D.2. We call the area in which the representative node of cell A can be during this Γ time interval as the *representative area*, denoted by $rep(A)$. Since the maximum distance this node can move during this time is $v\Gamma$, where v is the maximum speed of a node, $rep(A)$ is the area consisting of the cell, a u by $v\Gamma$ rectangle at each side of the cell, and a quarter circle with radius $v\Gamma$ centered at each corner of the cell (Fig 24.) Since the distance between any point in $rep(A)$ to the center of A is at most $u/\sqrt{2} + v\Gamma$, any point within the circle centered at the center of cell A with radius $r_{cover} = R - \left(u/\sqrt{2} + v\Gamma\right)$ is in the transmission range of the representative node of cell A. We call this area the *covered area* of cell A and denote it by $cover(A)$.

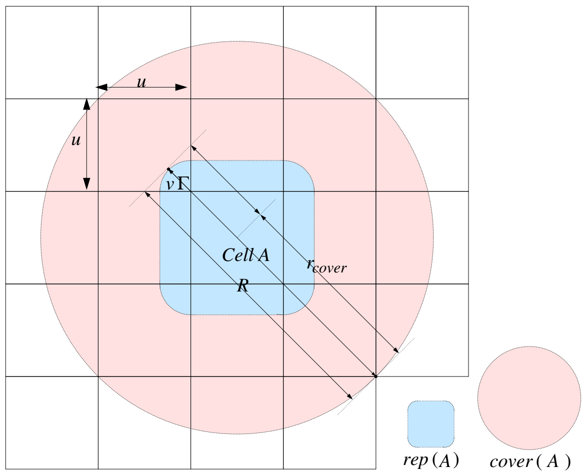


Fig. 24. $Rep(A)$ and $cover(A)$

In each of the approaches for Path Selection module, the selection of u will guarantee cell $A \subseteq cover(A)$. Our approaches guarantee the following two conditions:

B1 When a cell calls function $forward$, all the nodes in the covered area of the sending cell receive the message.

B2 Each cell calls $forward$ at least once.

Coverage can be achieved by these two conditions under the following assumption. We will provide specific constraints for each approach to meet this assumption.

- **Assumption**: if the whole area is covered by the message, then all the nodes receive the message.

D. Message Forwarding

In the Message Forwarding module, we consider the problem of forwarding a $\mathbf{PMessage}$ m by a node p, on behalf of cell s_cell, to a neighboring cell d_cell. The format of $\mathbf{PMessage}$ is defined by the Path Selection module, which includes a field $\mathbf{Data}$ to store the broadcast data. The Message Forwarding module provides the following services: if d_cell has no nodes in it, then the forwarding call returns NULL. Otherwise, a node in d_cell is selected and receives the message. In either cases, the message to be forwarded is broadcast to all nodes in $cover(s\_cell)$.

- Function $forward($ $\mathbf{PMessage}$ m, **Cell** s_cell, **Cell** $d\_cell)$ on node p: if no node exists in d_cell, returns NULL; otherwise returns SUCC.

- If $forward(m, s\_cell, d\_cell)$ returns SUCC, a node in d_cell is selected. By “a node being selected”, we mean event $receipt(\mathbf{PMessage}\ m, \mathbf{Cell}\ from\_cell)$ occurs on this node, where parameters m and $from\_cell$ are the parameters m and s_cell in $forward$ respectively.

- When $forward(m, s\_cell, d\_cell)$ is called, event $broadcast\_recv(\mathbf{Data}\ data)$ occurs on all the nodes in $cover(s\_cell)$, where parameter $data$ is the value of field $\mathbf{Data}$ in the parameter m of $forward$.

We propose a solution to this problem which does not depend on neighbor information or global identification. The expected message and time complexity is a constant if the distribution of nodes satisfies a certain condition. The procedure involves the sender that calls $forward$ and the nodes in its 2-hop neighborhood — the sender communicates with its neighbors and messages cover its neighbors' neighborhoods when its neighbors respond. It is possible that multiple forwardings are active in the network simultaneously. Here we give a complexity analysis under the assumption that the sets of nodes involved in the different forwardings are disjoint. An extended version will be presented later in section VI.F for the general case and the complexity analysis is easily extended.

1. Forwarding

The Message Forwarding module calls the basic communication service to send and receive these types of messages. If there exist nodes in the destination cell, we select exactly one node as the representative node by finding a subarea in the destination cell which contains exactly one node; the single node in this subarea is selected as the representative node. Our solution is based on the following simple idea for a node p to check whether the number of nodes in a given area A is zero, one, or more than one: p transmits an $\mathbf{N}$-type message and each neighbor in area A responds in the next step; p will receive no (one respectively) response if there is zero (one respectively) node in A, otherwise a collision will be detected.

Let t be the time slot in which function $forward$ is called on node p. In our solution, node p divides the area d_cell and tries to find out an subarea which contains exactly one node. Node p starts with area d_cell. At step t, p transmits a message and nodes in d_cell respond at step $t + 1$. At step $t + 2$, p gets the information about number of nodes in d_cell: if no node or exactly one node exists in d_cell, the forwarding is done. Otherwise, there exists multiple nodes in d_cell.

Then node p applies on d_cell a division and check procedure to find a subarea that contains exactly one node. Given an area A which contains multiple nodes at step $t + 4i$, node p finds a subarea of A as follows. At step $t + 4i$, p divides A into two same size areas, denoted by A_1 and A_2, and transmits to check the number of nodes in A_1; nodes in A_1 respond at step $t + 4i + 1$. Node p knows the information about the number of nodes in A_1 at step $t + 4i + 2$: if there is no node in A_1, p transmits at step $t + 4i + 2$ to check the number of nodes in A_2 and the nodes in A_2 respond at step $t + 4i + 3$. Thus at step $t + 4i + 4$, node p either gets an area which contains exactly one node or it gets an area which contains multiple nodes. If an area that contains multiple nodes is found, node p applies the above procedure on this area, starting at step $t+4i+4$. This procedure is repeated until an area containing exactly one node is found.

Once such an area is found, p transmits a $\mathbf{D}$ message carrying the broadcast data and the information about the found area.When a node receives this message, event $broadcast\_recv$ occurs, in which the broadcast data is delivered to the application; the node checks the information about the found area carried by this message — if it is in this area, then this node is selected and event $receipt$ occurs. Note that node p transmits only in time slots $t+2i$, while the nodes in the destination area respond only in time slots $t+2i+1$, $i=0,1,\ldots$.

We define the types of messages as below. The codes are provided in Figs. 25-28.

- Fields in $\mathbf{N}$-type $\mathbf{Message}$: $(Type$, Msg, S_cell, $Area)$,

  In order to check the number of nodes in $Area$, a node sends an $\mathbf{N}$-type message on behalf of cell S_cell; nodes in $Area$ are required to response. This type of message also carries broadcast data packed in a $\mathbf{PMessage}$ message Msg.

- Fields in $\mathbf{N_{ack}}$-type $\mathbf{Message}$: $(Type)$

  A node responds a $\mathbf{N}$-type message by an $\mathbf{N_{ack}}$-type message.

- Fields in $\mathbf{D}$-type $\mathbf{Message}$: $(Type, Msg, Area, TimeSlot)$,

  A node sends a $\mathbf{D}$-type message to designate a node that resides in $Area$ at step $TimeSlot$. This type of message also carries broadcast data packed in a $\mathbf{PMessage}$ message Msg.

2. Complexity

We give an analysis for the complexity in terms of D and $V(x)$. Note function $forward$ relies only on the *number* of nodes in a specific area, instead of the *specific nodes* in this

```
Local state of node $p$: $int\ l$, $int\ x$, $area\ A$, $area\ A_1$, $area\ A_2$, $int\ ack\_step$.
Function $forward(\mathbf{PMessage}\ m_p, \mathbf{Cell}\ s\_cell, \mathbf{Cell}\ d\_cell)$ called by node $p$:
step $t$, code $forward_0$:
    Create an $\mathbf{N}$-type Message $m(\mathbf{N}, m_p, s\_cell, d\_cell)$; $transmit(m)$;

step $t+2$, code $forward_2$:
    switch $receive()$
    case $\emptyset$:
        return NULL;
    case Message:
        Create a $\mathbf{D}$-type Message $m(\mathbf{D}, m_p, d\_cell, t+1)$;
        $transmit(m)$;
        return SUCC;
    case C:
        $l = 1$;
        Divide $A$ into two $(\frac{u}{2} \times u)$ areas $A_1$ and $A_2$;
        $A = A_1$;
        Create an $\mathbf{N}$-type Message $m(\mathbf{N}, m_p, s\_cell, A)$; $transmit(m)$;
    endswitch

step $t+4i$, code $forward_{4i}$, $i = 1, 2, \ldots$
    switch $receive()$
    case $\emptyset$:
        $x = 0$; $A = A_2$;
        Create an $\mathbf{N}$-type Message $m(\mathbf{N}, m_p, s\_cell, A))$; $transmit(m)$;
    case Message: $ack\_step = t + 4i - 1, x = 1$;
    case C: $x = 2$;
    endswitch

step $t+4i+2$, code $forward_{4i+2}$, $i = 1, 2, \ldots$
    if $(x == 0)$ then
        switch $receive()$
            case $\emptyset$: $x = 0$;
            case Message: $ack\_step = t + 4i + 1$; $x = 1$;
            case C: $x = 2$;
        endswitch
    endif
    if $(x == 1)$ then
        Create a $\mathbf{D}$-type Message $m(\mathbf{D}, m_p, A, ack\_step)$;
        $transmit(m)$;
        return SUCC;
    else if $(x == 0)$ then
        return NULL;
    else
        $l{+}{+}$;
        Divide $A$ into two areas with same size: $A_1$ and $A_2$, each has size $\left(\frac{u}{2^{\lceil l/2 \rceil}} \times \frac{u}{2^{\lfloor l/2 \rfloor}}\right)$;
        $A = A_1$;
        Create a $\mathbf{N}$-type Message $m(\mathbf{N}, m_p, s\_cell, A)$;
        $transmit(m)$;
    endif
```

Fig. 25. Function *forward*

Event $receipt_N$ on node p in step t:
Pre: $receive()$ returns a $\mathbf{N}$-type Message m_{rev}
Action:
step t:

```
1 **if** $(p.location() \in m_{rev}.Area)$ **then**
2     Create a $\mathbf{N_{ack}}$ Message $m(\mathbf{N_{ack}})$;
3     $transmit(m)$;
```

Fig. 26. $receipt_N$ on node p in step t

/* Node is selected as the representative node in Event */

Event $receipt(\mathbf{Message}\ m, \mathbf{Cell}\ from\_cell)$ on node p at step t:

Pre: $(receive()$ returns a $\mathbf{D}$-type Message $m_{rev}) \wedge (p.location(m_{rev}.TimeSlot) \in m_{rev}.Area)$

Parameters: $m = m_{rev}.Msg$ and $from\_cell = m_{rev}.S\_cell.$

Fig. 27. Precondition of Event $receipt(\mathbf{Message}\ m, \mathbf{Cell}\ from\_cell)$

/* Node receives broadcast data in this event */

Event $broadcast\_recv(\mathbf{Data}\ data)$ on node p at step t:

Pre: $receive()$ returns a $\mathbf{D}$-type message m_{rev}

Parameters: $data = m_{rev}.Msg.Data$

Fig. 28. Precondition of $broadcast\_recv(\mathbf{Data}\ data)$

area, and the analysis results shown below only rely on the distribution of mobile nodes.

First we show in the worst case, the number of steps in forwarding a message to a $u \times u$ area is $O(\log m)$, where m is the maximum number of mobile nodes that can be in a $u \times u$ area. The reason is that function $forward$ returns immediately if no node exists in the destination area; otherwise it divides the area into halves and eventually the area being checked is small enough that it contains only one node. Note if u is a constant, m is a constant since nodes cannot be arbitrarily small. However, if nodes are small relative to u^2, m can be large. In the sequel, we consider the expected time complexity and present a constraint on the distribution that guarantees the expected time complexity is a constant independent of the size of mobile nodes.

Note that the function returns in step $t+4k+2$, that is, it takes $4k+3$ steps for some value of $k \geq 0$. We have the following lemma:

Lemma 25 *Denoting the probability that function $forward$ returns at or after step $t+4k+2$, that is, it takes at least $4k+3$ steps, by $g(k)$, we have*

$$g(0) = 1 \textit{ and } g(k) \leq V\left(\frac{u}{2^{\lfloor (k-1)/2 \rfloor}}\right), k \geq 1$$

Proof. Since the function takes at least three steps, we have $g(0) = 1$. Suppose the function returns at or after step $t+4k+2$, $k \geq 1$. If there exists zero or exactly one node in A at the beginning of step $t+4(k-1)+2$, the function will return in this step (line 11 and 13). So at the beginning of step $t+4(k-1)+2$, there exist at least two nodes in A. Note the size of A at the beginning of step $t+4(k-1)+2$ is the same as that of A_1 or A_2 at the end of step $t+4(k-2)+2$, which is $(u/2^{\lceil l/2 \rceil}) \times (u/2^{\lfloor l/2 \rfloor}) = (u/2^{\lceil (k-1)/2 \rceil}) \times (u/2^{\lfloor (k-1)/2 \rfloor})$ because of $(l = i+1)$ at the end of step $t+4i+2$. Since the probability of having more than one node in such an area is $V((u/2^{\lceil (k-1)/2 \rceil}) \times (u/2^{\lfloor (k-1)/2 \rfloor}))$, we have $g(k) \leq V((u/2^{\lceil (k-1)/2 \rceil}) \times (u/2^{\lfloor (k-1)/2 \rfloor})) \leq V(u/2^{\lfloor (k-1)/2 \rfloor})$. $\square$

Directly from this lemma, we have:

Lemma 26 *Denoting the probability that function $forward$ takes no more than $4k+3$ steps by $f(k)$ and the probability that it takes exactly $4k+3$ steps by $P(k)$, we have*

- $f(k) = 1 - g(k+1) \geq 1 - V\left(\frac{u}{2^{\lfloor k/2 \rfloor}}\right), k \geq 0$, *and*
- $P(0) = 1 - V(u)$ *and* $P(k) \leq g(k) \leq V\left(\frac{u}{2^{\lfloor (k-1)/2 \rfloor}}\right), k \geq 1.$

Given $f(k)$ presented in Lemma 26 , we have

Theorem 27 *The probability that function $forward$ takes no more than $8\left\lceil \log\left(\frac{u}{D}\right)\right\rceil + 3$ steps is at least $1 - V(D)$.*

We have the following upper bound on the expected value of the number of steps taken in an execution of $forward$:

Lemma 28 *Denoting the expected value of $P(k)$ by E_P, we have*

$$E_P \leq 3 + 16\sum_{i=1}^{\infty} i \cdot V\left(\frac{u}{2^{i-1}}\right)$$

Proof. Note that the function only returns in step $t + 4k + 2$, for some $k \geq 0$, that is, the number of steps is $4k + 3$ for some $k \geq 0$.

$$\begin{aligned}
E_P &= \sum_{k=0}^{\infty}(4k+3) \cdot P(k) \\
&= \sum_{k=0}^{\infty} 4k \cdot P(k) + \sum_{k=0}^{\infty} 3 \cdot P(k) \\
&= 4\sum_{k=1}^{\infty} k \cdot P(k) + 3\sum_{k=0}^{\infty} P(k) \\
&\leq 3 + 4\sum_{k=1}^{\infty} k \cdot V\left(\frac{u}{2^{\lfloor (k-1)/2 \rfloor}}\right) \\
&= 3 + 4\sum_{i=1}^{\infty}\left((2i-1) \cdot V\left(\frac{u}{2^{\lfloor (2i-2)/2 \rfloor}}\right) + 2i \cdot V\left(\frac{u}{2^{\lfloor (2i-1)/2 \rfloor}}\right)\right) \\
&= 3 + 4\sum_{i=1}^{\infty}(4i-1) \cdot V\left(\frac{u}{2^{i-1}}\right) \\
&\leq 3 + 16\sum_{i=1}^{\infty} i \cdot V\left(\frac{u}{2^{i-1}}\right)
\end{aligned}$$

□

A sufficient condition for E_P to converge is that $\sum i \cdot V\left(\frac{u}{2^{i-1}}\right)$ converges. Based on the fact that $\sum \frac{1}{i^2}$ converges, a sufficient condition is $i \cdot V\left(\frac{u}{2^{i-1}}\right) = O\left(\frac{1}{i^2}\right)$, $i \to +\infty$. If u is a constant, it is equivalent to $V\left(\frac{1}{2^i}\right) = O\left(\frac{1}{i^3}\right)$, $i \to +\infty$. Letting x be $\frac{1}{2^i}$, that is, $i = \log \frac{1}{x}$, this condition is equivalent to

$$\exists A > 0, \lim_{x \to 0+} \frac{V(x)}{\frac{1}{\left(\log \frac{1}{x}\right)^3}} \le A \tag{6.1}$$

Note that for any $c > 0$, the function x^c satisfies condition 6.1.

Theorem 29 *If $V(x)$ satisfies condition 6.1 and u is a constant, then the function $forward$ has a constant expected time complexity.*

Note this analysis only depends on the distribution of mobile nodes: nodes can move into and out of an area, but function $forward$ has a constant expected time complexity as long as the distribution of mobile nodes satisfies condition 6.1.

This condition is just a sufficient condition. In a network in which the node distribution does not satisfy this condition, it still is possible that the function $forward$ has a constant expected number of steps. In order to have an intuitive idea of this condition, we provide the plot of function $1/\log^3(1/x)$ in Figure 29, compared to the plots of functions x and x^2.

The example presented in section VI.A (a network in which nodes are distributed on a grid graph) satisfies condition 6.1 since $V(x) = 0$ when $x < D$ for some constant D. Thus the expected time complexity of forwarding is a constant when applied on such a network. Note the distribution of such a network is deterministic. Here we give an example in which the distribution is probabilistic. In many applications, the purpose of sensor placement is to cover the whole area of interest. A simple deployment is to divide the area into grids and distribute a node in each grid. Assume the size of each grid is $C \times C$ and the area of the network is $kC \times kC$. Thus the number of grids and the number of sensors is k^2.

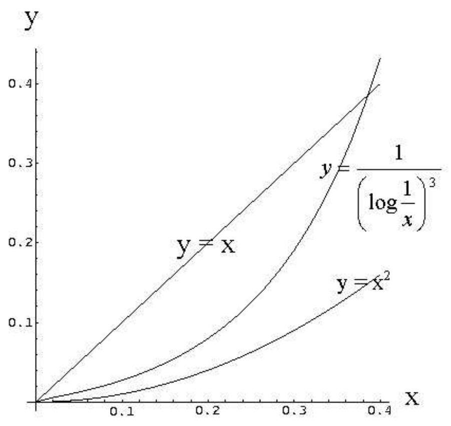


Fig. 29. A sufficient condition for E_P to converge

Given the sensor assigned to the grid centered at location (a, b), we denote the probability of that sensor being placed at location (x, y) by $P_{a,b}(x, y)$. Assume the sensor is *uniformly* distributed in the designated grid, that is

$$P_{a,b}(x,y) = \begin{cases} \frac{1}{C^2} & \text{if } \left(x \in [a - \frac{C}{2}, a + \frac{C}{2}]\right) \wedge \\ & \left(y \in [b - \frac{C}{2}, b + \frac{C}{2}]\right) \\ 0 & \text{otherwise} \end{cases}$$

Now we calculate $V(x)$ based on this distribution function. Consider any $x \times x$ area A, where $x < C$. Since $x < C$, there exist at most four grids that overlap with A (Fig. 30.) Denote the probability that there is no node (exactly one node respectively) in A by P_0 (P_1 respectively) and the probability that there is no node (exactly one node respectively) in area A_i by P_0^i (P_1^i respectively), $i = 1, 2, 3, 4$. Note P_0^i is $(1 - (a_i \cdot b_i)/C^2) \geq (1 - x^2/C^2)$ for $i = 1, 2, 3, 4$. Thus we have $P_0 = \prod_{i=1}^4 P_0^i \geq (1 - x^2/C^2)^4$. Note $P_1^i = (a_i \times b_i)/C^2$, $i = 1, 2, 3, 4$ and $\sum_{i=1}^4 P_1^i = x^2/C^2$. Since the probability that exactly one node exists in A and this only node is in A_i is $\left(P_1^i \cdot \prod_{j \neq i, j=1}^4 P_0^j\right)$, we have $P_1 \geq \sum_{i=1}^4 \left(P_1^i \cdot \prod_{j\neq i,j=1}^4 P_0^j\right)$ $\geq \sum_{i=1}^4 \left(P_1^i \cdot (1 - x^2/C^2)^3\right) = (x^2/C^2) \cdot (1 - x^2/C^2)^3$. Thus we have $V(x) = 1 - (P_0 + P_1) \leq 1 - \left((1 - x^2/C^2)^4 + (x^2/C^2) \cdot (1 - (x^2/C^2))^3\right) = 1 - (1 - x^2/C^2)^3 = x^6/C^6 -$

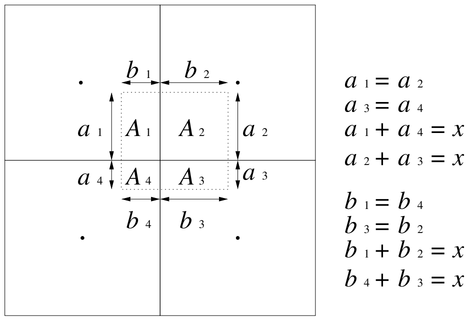


Fig. 30. An example of network satisfying condition 6.1

$3x^4/C^4 + 3x^2/C^2$. Straightforward calculations verify that function $x^6/C^6 - 3x^4/C^4 + 3x^2/C^2$ satisfies condition 6.1, and thus so does V(x).

3. Discussion

Function $forward$ also provides a way to gather the distribution information: at the end of each call, node p finds an area in which exactly one node exists. This area can be used to estimate the average area occupied by a single node. If the mobile nodes are roughly evenly distributed, the forwarding can be sped up by using this estimated value — given a destination cell, the function first checks a subarea with the estimated size; if there is no node in this area, the area is doubled, otherwise the area is divided into two smaller areas and this procedure is repeated. Since the area in which only one node exists is close to the estimated area with high probability, the number of steps can be kept small.

In this approach, the division and check procedure is repeated until an area that contains a single node is found. In some situations, we do not require high coverage of message delivery but focus on quick propagation. We can restrict the number of steps to be less than a certain value. There exists a tradeoff between coverage and message delay in this approach: the time complexity of each forwarding is bounded while it is possible an area with a single node is not found. But the probability of failure is bounded according to

Lemma 26.

We also consider message forwarding in a system *without collision detection*. The idea is based on [32]. First we consider the problem defined as follows: upon receipt of a message from a node p', node p forwards the message to a destination area A — if there exist nodes in A, one should be selected; otherwise p should be notified that no node exists in A. We provide a solution to this problem. In step 1, node p transmits a message with information A included; all the nodes in p's neighborhood receive this message in this step. In step 2, only nodes in area A respond. If p receives one response in step 2, node p knows there is only one node in A and this node is selected. Otherwise, nodes in A and p' respond in the next step: if p receives a response, node p knows this response is from p' and there are no nodes in A; otherwise there are multiple nodes in A and A is divided into smaller areas. This procedure is repeated until an area with a single node is found. We assume there exists an auxiliary node in the source's neighborhood, which performs as p' when initially the source starts forwarding the message; such a node p' exists when a node forwards upon receipt of a message from some neighbor. The Message Forwarding module can use this procedure to provide services defined in this section if a collision detection mechanism is not available.

The assumption of the existence of an auxiliary node can be removed if a node can be elected in the source's neighborhood. In a synchronous network with global identification, this election can be done by scheduling nodes' transmission based on their ids [32] — the source transmits a message and each neighbor with id k responds in time slot $2k$; once the source receives a response, say in the $2i$th time slot, it broadcasts in the $2i+1$ time slot to tell its neighbors to stop responding; the node which responds first is elected. In our approach, we do not assume a global identification. But given the fact that mobile devices are not infinitely small and two mobile devices cannot be at the same location, we can divide a given bounded area into a finite number of cells such that at most one mobile node resides in each cell, and define a one-to-one map from this set of cells to a set of integers. A node can be elected in the source's neighborhood by scheduling the transmissions of neighbors based on this map: a node transmits at time slot $2i$, where i is the integer which corresponds to its location, and the source tells its neighbors to stop responding in the time slot after it receives the first response. In this approach, the node whose location corresponds to the smallest integer is elected and it takes at most $O(K)$ time slots, where K is the size of the corresponding set of integers.

E. Approach $\mathbf{DFS}$

In this section, we present an approach, denoted by $\mathbf{DFS}$, for the Path Selection module. In $\mathbf{DFS}$, message propagation is implemented by circulating the message in depth first search order on the virtual graph — when a cell receives the message (through a representative node), it decides which neighboring cell to forward the message according to the depth first search order; it calls the Message Forwarding module to forward the message to the selected cell. Once the message has been forwarded, one node in the destination cell is selected as the representative node and it is responsible for forwarding the message to the next cell. Note that during message propagation, the virtual graph is not constructed explicitly. Instead, given the value of u, which will be presented later in this section, each node knows the location of the cell it resides in and the locations of neighboring cells based on the knowledge of the network area and the location service. Thus a representative can call the Message Forwarding module to forward the message to the selected neighboring cell; if the destination cell does not exist, the representative node will be notified and the selection will be repeated until an exsiting selected neighbor is found.

In this section, first we consider the requirement on the value of u and the code of $\mathbf{DFS}$. Then we present constraints on the distribution and mobility of nodes. Given these constraints, the value of u can be decided. After that, complexity analysis is provided.

1. Selection of u

Approach $\mathbf{DFS}$ requires the representative node to communicate with the nodes in the destination cell. Recall Γ is defined as the maximum time it takes for a representative node to communicate with all the selected neighboring cells. Since the representative node moves at most $v\Gamma$ away from the cell, if the maximum distance between two points in two neighboring cells is at most $R - v\Gamma$, the distance between the representative node and any node in a neighboring cell is at most R. So we guarantee the representative node is able to forward messages to the neighboring cells by requiring $(2u)^2 + (2u)^2 \leq (R - v\Gamma)^2$, that is, $u \leq (R - v\Gamma)/(2\sqrt{2})$ (Fig. 31). We select $u = (R - v\Gamma)/(2\sqrt{2})$, since a smaller u will introduce a larger number of virtual vertices and thus a larger delay in broadcasting. Constraints on v and Γ to guarantee the coverage are discussed in section VI.E.3.

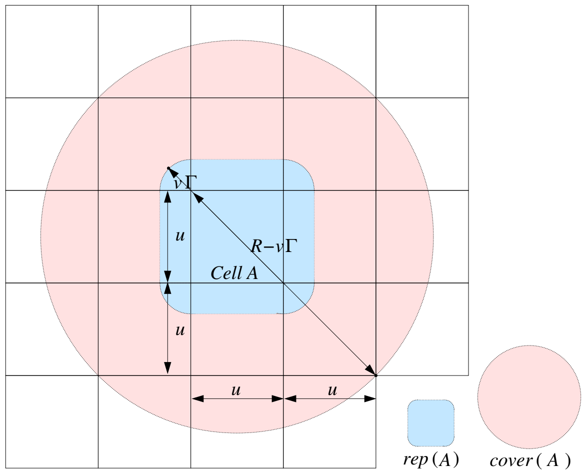


Fig. 31. Selection of u for $\mathbf{DFS}$

2. Code

The code of approach $\mathbf{DFS}$ is presented in Fig. 32.

3. Constraints and Complexity

It is easy to see our algorithm guarantees conditions **B1** and **B2**. Here we give the constraints on distribution and mobility of mobile nodes to meet **Assumption**. One way is to restrict the movement of nodes in a local area. Formally, we require $\forall$ node p, $\exists$ cell A, such that at any time during the broadcasting, $p \in cover(A)$.

We make the following assumption on the distribution of mobile nodes:

> $\mathbf{Con}_{\mathbf{dis}}^{\mathbf{DFS}}$: The virtual graph is static and each forwarding finishes within F steps for some constant F.

When a node is selected as the representative node of the cell it resides in, that is, event *receipt* occurs, this node forwards the message to its neighboring cells in the order of directions in $Dirs$ until the first existing neighboring cell is found. Thus we can set $\Gamma = (21 + F)\delta$ since function $forward$ takes three steps if the destination cell does not exist. The total time Φ for the message to cover the whole network area is $\Phi \leq \Gamma\left(2\mathcal{A}/(u^2) - 1\right) \leq \Gamma\left(2\mathcal{A}/(u^2)\right)$. Note $r_{cover} = R - \left(u/\sqrt{2} + v\Gamma\right) = 3\sqrt{2}u/2$ and the minimum distance between a node in a cell and the boundary of the covered area of this cell is $\sqrt{2}u$ (Fig. 31). Then the assumption can be met if $v \leq \sqrt{2}u/\Phi$, which can be guaranteed by $v \leq (\sqrt{2}u)/(\Gamma\left(2\mathbf{A}/(u^2)\right)) = u^3/(\sqrt{2}\Gamma\mathcal{A})$ where $u = (R - v\Gamma)/(2\sqrt{2})$, that is

> $\mathbf{Con}_{\mathbf{mob}}^{\mathbf{DFS}}$: $v \leq (R - v\Gamma)^3/(32\Gamma\mathcal{A})$, where $\Gamma = \delta(21 + F)$.

One solution to this inequality is $v \leq v_{max} = \min\{R/20\Gamma, (19R)^3/(20^3 \cdot 32\Gamma\mathcal{A})\}$. The reason follows: we have $(R - v\Gamma)^3/(32\Gamma\mathcal{A}) \geq (R - R\Gamma/(20\Gamma))^3/(32\Gamma\mathcal{A}) = (19R)^3/(20^3 \cdot 32\Gamma\mathcal{A}) \geq v$, since $v \leq R/(20\Gamma)$. Sample values of v_{max} are provided in

Format of PMessage: Each message carries the following fields:

- Array P, $\forall$ cell c, $P[c] \in Dirs$: parent cell of c. Initially $P[c]$ =NULL;
- Array N_v, $\forall$ cell c, $N_v[c] \subseteq Dirs$: the set of neighbors of c which have been visited in the current round. Initially $N_v[c] = \emptyset$.
- Data: broadcasting data.

Function $broadcast(\textbf{Data}\ data)$ (called by source s at step t)

```
1  $s\_cell = cell(s, t)$;
2  $i = 0$;
3  $bSucc = false$;
4  **while** $(i \le 7) \wedge (\neg bSucc)$ **do**
5      $d\_cell = neighbor(s\_cell, Dirs[i])$;
6      Create a PMessage $m(P, N_v, data)$;
7      $m.N_v(s\_cell) = \{d\_cell\}$;
8      $bSucc = (forward(m, s\_cell, d\_cell) \neq$ NULL);
9      $i++$;
10 **endwhile**
```

Event $receipt(\textbf{PMessage}\ m, \textbf{Cell}\ from\_cell)$ on node p at step t:
Action:

```
1  $s\_cell = cell(p, t)$;
2  **if** $(m.N_v[s\_cell] == \emptyset)$ **then**
3     $m.P[s\_cell] = from\_cell$;
4     $m.N_v[s\_cell] = from\_cell$;
5  **endif**
6  $i = 0$;
7  $bSucc = false$;
8  **while** $((i \le 7) \wedge (\neg bSucc))$ **do**
9     $d\_cell = neighbor(s\_cell, Dirs[i])$;
10     **if** $(d\_cell \notin m.N_v[s\_cell])$ **then**
11        $m.N_v[s\_cell] = m.N_v[s\_cell] \cup \{d\_cell\}$;
12        bSucc=$(forward(m, s\_cell, d\_cell) \neq$ NULL );
13     **endif**
14     $i++$;
15 **endwhile**
16 **if** $(\neg bSucc)$ **then** /* Backtrack if all the neighbors have been visited */
17    **if** $(m.P(s\_cell) \neq$ NULL) **then**
18       $d\_cell = m.P(s\_cell)$;
19       $m.N_v[s\_cell] = \emptyset$;
20       $m.P[s\_cell]$ =NULL;
21       $forward(m, s\_cell, d\_cell)$;
22    **endif**
23 **endif**
```

Fig. 32. DFS token circulation on the virtual graph

Table V. In section VI.D.3, we discussed that it is very likely that the number of steps in each forwarding will be small if the mobile nodes are roughly evenly distributed. Here we choose small sample values for F.

Now we consider time complexity. Note by constraint $\mathbf{Con}^{\mathbf{DFS}}_{\mathbf{mob}}$, we have $v\Gamma \leq v\Phi \leq \sqrt{2}u = (R - v\Gamma)/2$, that is, $v\Gamma \leq R/3$. So we have $u = (R - v\Gamma)/(2\sqrt{2}) \geq R/(3\sqrt{2})$. Thus we have $u \in \left[R/(3\sqrt{2}), R/(2\sqrt{2})\right]$, and it is easy to show the following lemma. The intuition is that under the above constraints, the number of calls of $forward$ is $O\left(\mathcal{A}/(u^2)\right) = O\left(\mathcal{A}/(R^2)\right)$ and the time complexity of each call of $forward$ is $O(\delta)$.

Lemma 30 *If* $\mathbf{Con}^{\mathbf{DFS}}_{\mathbf{dis}}$ *and* $\mathbf{Con}^{\mathbf{DFS}}_{\mathbf{mob}}$ *are true, the time complexity of* $\mathbf{DFS}$ *is* $O\left(\mathcal{A}\delta/(R^2)\right)$.

F. Approach $\mathbf{SF}$

In this section, we present another approach, denoted by $\mathbf{SF}$, for the Path Selection module. In this approach, messages are propapated in parallel on the virtual graph. Since multiple forwardings are active simultaneously, interferences may occur. A variant of $forward$, called $scheduled\text{-}forward$, is used to guarantee that all the nodes in the covered area of the sending cell receive the message. In $\mathbf{SF}$, simple flooding is applied on the virtual graph — each message carries the information about the cells it has been forwarded to; upon reception of a message, a cell forwards the message to neighbors that have not received this message. Communication between adjacent cells is done by *scheduled-forwarding*.

In the sequel, we first introduce the requirement on the value of u. Then scheduled-forwarding is proposed. After that we present the code and constraints on distribution and mobility, as well as a complexity analysis.

Table V. Sample values of v_{max} for $\mathbf{DFS}$

Parameters				v_{max}
R (m)	δ (s)	F	A (m^2)	
100	5×10^{-5}	11	100×100	1674.6 m/s =6028.4 km/h
			200×200	418.6 m/s = 1507.1 km/h
			500×500	66.9 m/s = 241.1 km/h
			1000×1000	16.7 m/s = 60.3 km/h
100	10^{-4}	11	100×100	837.3 m/s =3014.2 km/h
			200×200	209.3 m/s = 753.6 km/h
			500×500	33.5 m/s = 120.6 km/h
			1000×1000	8.37 m/s = 30.14 km/h
100	5×10^{-4}	11	100×100	167.5 m/s =602.8 km/h
			200×200	41.9 m/s = 150.7 km/h
			500×500	6.7 m/s = 24.1 km/h
			1000×1000	1.7 m/s = 6.0 km/h
100	5×10^{-5}	15	100×100	1488.5 m/s =5358.6 km/h
			200×200	372.1 m/s = 1339.6 km/h
			500×500	59.5 m/s = 214.3 km/h
			1000×1000	14.9 m/s = 53.6 km/h
100	10^{-4}	15	100×100	744.2 m/s = 2679.3 km/h
			200×200	186.1 m/s = 669.8 km/h
			500×500	29.8 m/s = 107.2 km/h
			1000×1000	7.4 m/s = 26.8 km/h
100	5×10^{-4}	15	100×100	148.8 m/s = 535.9 km/h
			200×200	37.2 m/s = 134.0 km/h
			500×500	6.0 m/s = 21.4 km/h
			1000×1000	1.5 m/s = 5.4 km/h
100	5×10^{-5}	19	100×100	1339.6 m/s = 4822.7 km/h
			200×200	334.9 m/s = 1205.7 km/h
			500×500	53.6 m/s = 192.9 km/h
			1000×1000	13.4 m/s = 48.2 km/h
100	10^{-4}	19	100×100	669.8 m/s = 2411.4 km/h
			200×200	167.5 m/s = 602.8 km/h
			500×500	26.8 m/s = 96.5 km/h
			1000×1000	6.7 m/s = 24.1 km/h
100	5×10^{-4}	19	100×100	134.0 m/s = 482.3 km/h
			200×200	33.5 m/s = 120.6 km/h
			500×500	5.4 m/s = 19.3 km/h
			1000×1000	1.3 m/s = 4.8 km/h

1. Selection of u

In this approach, we require that two representative nodes of neighboring cells can communicate with each other. Since the representative node moves at most $v\Gamma$ away from the cell, if the maximum distance between two points in two neighboring cells is at most $R - 2v\Gamma$, the distance between the representative nodes from two cells is at most R. So we require $(2u)^2 + (2u)^2 \leq (R - 2v\Gamma)^2$, that is, $u \leq (R - 2v\Gamma)/(2\sqrt{2})$ (Fig. 33.) We select $u = (R - 2v\Gamma)/(2\sqrt{2})$ and we have $r_{cover} = R - \left(u/\sqrt{2} + v\Gamma\right) = (3R - 2v\Gamma)/4$. Constraints on v and Γ to guarantee the coverage are discussed in section VI.F.4.

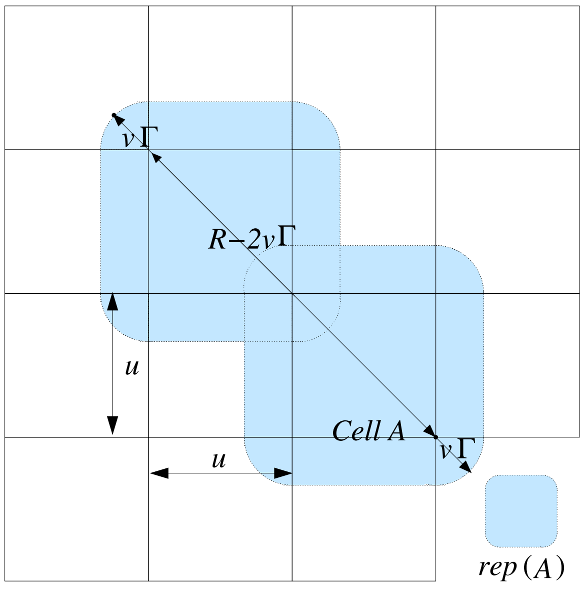


Fig. 33. Selection of u

2. Scheduled Forwarding

Scheduled forwarding is based on a coloring scheme. The coloring scheme, which will be described later in this section, guarantees that given two cells A and B with the same color, the distance between any point in $rep(A)$ and any point in $cover(B)$ is at least R. We denote the number of colors by K and the color of cell C by $color(C)$. Each cell is assigned a set of steps: letting round i, $i \geq 0$, be the subsequence of $2K$ steps: $r_i, r_i+1, \ldots, r_i+2K-1$, where $r_i = i \cdot 2K$, cell C is assigned two steps $\{r_i + 2c, r_i + 2c + 1\}$ in round i, where $c = color(C)$. Transmissions involved in a forwarding called by a cell is allowed only in the steps assigned to this cell. Suppose $scheduled\_forward$ is called at step t by node p, on behalf of cell s_cell, to forward message to cell d_cell. Denoting $color(s\_cell)$ by c, node p starts transmitting in round i_0, where i_0 is the smallest integer such that $r_{i_0}+2c \geq t$. During the execution, the action taken by node p in step $t + 2k$ of function $forward$ (presented in section VI.D) is rescheduled to step $r_{i_0+k} + 2c$, and nodes in d_cell response in step $r_{i_0+k} + 2c+1$. From this assignment, we see the time complexity of scheduled forwarding is $2K$ times the complexity of $forward$.

At the beginning of broadcasting, only the cell in which the source resides is forwarding messages. As messages are propagated in the network, multiple cells might be forwarding simultaneously. We guarantee that for each cell C, at most one node is designated as the representative node of C at any time. This is achieved as following: when a node is designated as a representative node, it responds with a $\mathbf{D_{ack}}$ message; during its forwarding to cell C, if a representative node p_s of cell S hears a $\mathbf{N}$ message on behalf of C, which means there is a representative node of C, or a $\mathbf{D_{ack}}$ message from nodes in C, which means a node is designated as the representative node of cell C, then p_s aborts the forwarding without designating any node. Note the selection of u guarantees p_s is in the transmission range of the representative node of C and of each node in cell C. We can guaranteed for each cell C, at most one node is designated as the representative node of C at any time for the following reason. Consider any round i. Note that the set of steps assigned to C's neighbors are disjoint by the coloring scheme.

- If no node is designated as the representative node of C at the beginning of round

i, at most one node is designated in this round since this node will transmit a $\mathbf{D_{ack}}$ message when it is designated, which causes the cells which are forwarding to C to abort forwarding.

- Otherwise let p_c be the representative node of C and c be $color(C)$. Note p_c transmits in both round $i-1$ and round i: p_c transmits an $\mathbf{N}$ message in round i as a representative node; in round $i-1$, p_c transmits at step $r_{i-1}+2c$ if it has been designated as a representative node, otherwise it responses a $\mathbf{D_{ack}}$ message when it is designated after step $r_{i-1}+2c$. Since it takes at least two rounds for a cell to finish forwarding, all the cells that are forwarding to C in round i receive at least one of these messages from p_c and they abort forwarding without designating any node.

Note that the representative nodes transmit in even steps and nodes in the destination cell respond in odd steps. The coloring scheme guarantees that given two cells C_1 and C_2 with the same set of assigned steps, the distance between $rep(C_1)$ and $cover(C_2)$ is at least R. Thus when the representative node of any cell A transmits, nodes in $cover(A)$ are at least R away from the representative node of any other cell which transmits at the same time; and when the nodes in the destination cell of A responds, the represenative node of A is at least R away from the nodes in the destination cell of any other cell C with the same color (Note given a cell C, the selection of u and r_{cover} guarantees that $D \subset cover(C)$ for any neighbor D of C.) So interference between different executions of forwarding is avoided.

Now we consider coloring scheme. Given any two cells, $A = [x_a, y_a]$ and $B = [x_b, y_b]$, with the same color, we want to guarantee the distance between any node in $rep(A)$ and any node in $cover(B)$ is at least R. This can be achieved by requiring $|x_a - x_b| + |y_a - y_b| \geq I$, where $I = 2\sqrt{2} \cdot R/u$ for the following reason. The distance between the center of A and the center of B is $\sqrt{|x_a - x_b|^2 \cdot u^2 + |y_a - y_b|^2 \cdot u^2} \geq \sqrt{((|x_a - x_b| \cdot u + |y_a - y_b| \cdot u)^2)/2}$

$= uI/\sqrt{2}$. Note the maximum distance between a node in $rep(A)$ to the center of A is $\sqrt{2}u/2 + v\Gamma$ and the maximum distance between a node in $cover(B)$ to the center of B is r_{cover}. So the minimum distance between a node in $rep(A)$ and a node in $cover(B)$ is at least $uI/\sqrt{2} - (\sqrt{2}u/2 + v\Gamma + r_{cover}) = uI/\sqrt{2} - R \geq R$.

Now we consider the assignment of steps. We construct a grid graph by deleting edges between cells $[x_1, y_1]$ and $[x_2, y_2]$, where $(x_1 = x_2 \pm 1) \wedge (y_1 = y_2 \pm 1)$ from the virtual graph. Note the distance between two nodes on the grid graph that correspond to cell $[x_1, y_1]$ and $[x_2, y_2]$ is $(|x_1 - x_2| + |y_1 - y_2|)$. We apply the coloring scheme proposed in [64] on the grid graph to guarantee the distance between any two vertices with the same color is at least I by using $K = I^2 + 1$ colors.

We present the code for $scheduled\_forward$ in Fig. 34.

3. Code

A simple way to broadcast on the virtual graph is simple flooding, as presented in Fig. 35.

4. Constraints and Complexity

It is easy to see that this approach guarantees conditions **B1** and **B2**. Now we consider constraints on distribution and mobility of mobile nodes to meet **Assumption**. In this section, we dicussion the constraints on mobility under the following constraint on the distribution of mobile nodes:

- $\mathbf{Con^{SF}_{dis}}$: The virtual graph is static and each execution of scheduled forwarding is guaranteed to be done within F steps.

Assume the source of broadcasting resides at the most northwest cell. We will give a constraint on the speed v of mobile nodes. Note in this approach, the maximum time for a

Definition of messages

- $\mathbf{D_{ack}}$-type **Message**: $(Type, Rep_{cell})$

$scheduled\_forward(\textbf{PMessage } m, \textbf{Cell } s\_cell, \textbf{Cell } d\_cell)$ called by node n in step t:
$i_0 = \lfloor \frac{t}{2K} \rfloor + \left\lfloor \frac{t \bmod 2K}{2c} \right\rfloor$, where $c = color(s\_cell)$

```
step $r_{i_0} + 2c$:
1 **if** $bAbort()$ **then return**;
2 $forward_0()$;

step $r_{i_0+1} + 2c$:
1 **if** $bAbort()$ **then return**;
2 $forward_2()$;

step $r_{i_0+2i} + 2c$, $i = 1, 2, \ldots$
1 **if** $bAbort()$ **then return**;
2 $forward_{4i}()$;

step $r_{i_0+2i+1} + 2c$, $i = 1, 2, \ldots$
1 **if** $bAbort()$ **then return**;
2 $forward_{4i+2}()$;

other steps:
1 **if** $bAbort()$ **then return**;
```

Subroutine boolean $bAbort()$

```
1 $bAbort = false$;
2 **if** $(receive() \notin \{\emptyset, C\})$ **then**
3     $m_{rev} = receive()$;
4     $bAbort = (m_{rev}.Type == \mathbf{N} \wedge m_{rev}.S\_cell == d\_cell) || (m_{rev}.Type == \mathbf{D_{ack}} \wedge m_{rev}.Rep\_cell == d\_cell)$;
5 **endif**
6 **return** $bAbort$;
```

/* Node p receives a message m from cell $from\_cell$, and it is selected as the representative node of cell rep_cell. */
Event $receipt(\textbf{Message } m, \textbf{Cell } from\_cell)$ on node p at step t:
Pre: $(receive()$ returns a $\mathbf{D}$-type Message $m_{rev}) \wedge (n.location(m_{rev}.TimeSlot) \in m_{rev}.Area)$
Parameters: $m = m_{rev}.Msg$, $from\_cell = m_{rev}.S\_cell$
Action: /* Notes: this action is taken before the action defined in Path Selection module. */

```
1 Create a $\mathbf{D_{ack}}$-type Message $m'(\mathbf{D_{ack}}, rep\_cell)$;
2 $tranmit(m')$;
```

Fig. 34. Scheduled forward

Format of PMessage: Each message carries the following fields:

- list V: the set of cells which has been visited by this message. Initially V =NULL.
- Data: broadcasting data.

Function $broadcast(\mathbf{Data}\ data)$ (called by source s at step t)

```
1  $s\_cell = cell(s, t)$;
2  $i = 0$;
3  **while** $(i \leq 7)$ **do**
4      $d\_cell = neighbor(s\_cell, Dirs[i])$;
5      Create a PMessage $m(V, data)$;
6      $m.V = neighbors(s\_cell)$;
7      $schedule\_forward(m, s\_cell, d\_cell)$;
8      $i++$;
9  **endwhile**
```

Event $receipt(\mathbf{PMessage}\ m, \mathbf{Cell}\ from\_cell)$ on node p at step t:
Action:

```
1  $s\_cell = cell(p, t)$;
2  $i = 0$;
3  **while** $(i \leq 7)$ **do**
4      $d\_cell = neighbor(s\_cell, Dirs[i])$;
5      **if** $(d\_cell \notin m.V)$ **then**
6          Create a PMessage $m'(V, data)$;
7          $m'.V = m.V \cup neighbors(s\_cell)$;
8          $schedule\_forward(m', s\_cell, d\_cell)$;
9      **endif**
10     $i++$;
11 **endwhile**
```

Fig. 35. Simple flooding on the virtual graph

representative node to communicate with all the selected neighbors is $\Gamma = 14KF\delta$ since each cell has at most eight neighbors and a representative node will not send the packet back to the cell from which it receives the packet. We say a cell is *covered* (*uncovered* respectively) if this cell has (has not respectively) received the message. We say a covered cell is *inactive* if it has finished all the transmissions in its first execution of forwarding, otherwise the cell is *active*. Note all the neighbors of an inactive cell are covered. By the scheduled transmission, if a cell is active at step t, this cell transmits at least once during steps t to $t + 2K$. We denote the set of uncovered cells (inactive covered cells and active covered cells respectively) at step t by $Uncovered(t)$ ($Inactive(t)$ and $Active(t)$ respectively) When a cell becomes an inactive covered cell at the end of step t, (that is, the last time the cell transmits in its first execution of forwarding is step t), all the nodes residing in this cell at step t receive the message. Since eventually all the cells become inactive, coverage can be achieved if all the nodes that move into an inactive cell later are guaranteed to receive the message. This property can be achieved by guaranteeing that any node p receives the broadcast message if the following conditions are true:

- (a) $\exists t_1$, such that p is not in the area of $Inactive(t_1)$ in step t_1, and
- (b) $\exists t_2 > t_1$, such that $C \in Inactive(t_1)$ where C is the cell in which p resides at step t_2.

We will show this condition can be guaranteed if the speed v of mobile nodes satisfies $v \leq u/\delta$ and $v \leq R/(4K\delta)$. Note the minimum distance between a point in a cell and the boundary of its covered area is $r_{cover} - u/\sqrt{2} = R/2$. By this constraint, if p moves into a cell A at some step, p will stay in *cover*(A) in the following $2K$ steps. Thus if p moves into an active cell at some step in $[t_1, t_2]$, which will transmit at least once in the following $2K$ steps, p will receive the message.

Now we show by contradiction that it is impossible that p never moves into an active cell during step t_1 and step t_2. Since C is inactive at time t_1, we have $C \in Inactive(t_2)$ by the definition of inactive cell. Thus there exists step $t \in [t_1, t_2]$, such that, denoting the cell in which p resides at step t (step $t+1$ respectively) by A (B respectively), we have $A \notin Inactive(t)$ and $B \in Inactive(t+1)$. By the constraint $v \leq u/\delta$, a node can only move to a neighboring cell within a step, thus A and B are neighboring cells. Since p never moves into an active cell before t_2, A is not active at time t, that is, A is uncovered at t because of $A \notin Inactive(t)$. Cell B cannot be an inactive cell at step t because otherwise all B's neighbors, including A, are covered at step t. So B is uncovered at step t and is inactive at step $t+1$, which is impossible since it takes at least one step for a cell to finish forwarding. Thus coverage can be guaranteed by the following constraint:

- $\mathbf{Con}^{\mathbf{SF}}_{\mathbf{mob}}$: $\left(v \leq u/\delta = (R - 28KF\delta v)/(2\sqrt{2}\delta)\right) \wedge (v \leq R/(4K\delta))$, where $u = (R - 28KF\delta v)/(2\sqrt{2})$, $K = I^2 + 1$ and $I = 2\sqrt{2}R/u$.

We can show that a solution for this constraint is is $v \leq v_{max} = R/(3 \times 28 \times 145 \times \delta F)$. Note the smaller is v, the smaller is K, thus if this constraint is satisfied by a value of v, it is satisfied by all the value smaller than v. So we only need to consider $v_{max} = R/(3 \times 28 \times 145 \times \delta F)$. Denoting $x = 28KF\delta v/R$, we have $u = (R - 28KF\delta v)/(2\sqrt{2})$ $= (1-x)R/(2\sqrt{2})$ and $K = I^2 + 1 = \left(2\sqrt{2}R/u\right)^2 + 1 = 8 \cdot R^2/(u^2) + 1 = 8 \cdot R^2/\left((1-x)R/(2\sqrt{2})\right)^2 + 1 = 64/((1-x)^2) + 1$. By replacing v by v_{max}, we have $x = 28KF\delta v/R = K/(3 \times 145)$. So we have an equation $K = 64/((1 - K/(3 \times 145))^2) + 1$ and the solution is $K = 145$. Thus we have $x = K/(3 \times 145) = 1/3$ and $v_{max} = R/(3 \times 28 \times 145 \times \delta F) = R/(3 \times 28\delta KF)$. Since we have $(R - 28KF\delta v)/(2\sqrt{2}\delta) = (1-x)R/(2\sqrt{2}\delta) = R/(3\sqrt{2}\delta) \geq R/(3 \times 28\delta KF) = v_{max}$ and $R/(4K\delta) \geq R/(84\delta KF)$ $= v_{max}$, the constraint is satisfied. Sample values are presented in Table VI, in which we choose small values for F for the same reason discussed in section VI.E.3.

Table VI. Sample values of v_{max} for $\mathbf{SF}$

Parameters			Constraint on $v \leq \frac{R}{12180\delta F}$
R (meter)	δ (second)	F	$\frac{R}{12180\delta F}$
100	$5 \cdot 10^{-5}$	11	14.92 m/s = 53.74 km/h
		15	10.95 m/s = 39.41 km/h
		19	8.64 m/s = 31.11 km/h
100	10^{-4}	11	7.46 m/s = 26.87 km/h
		15	5.47 m/s = 19.70 km/h
		19	4.32 m/s = 15.56 km/h
100	$2 \cdot 10^{-4}$	11	3.73 m/s = 13.43 km/h
		15	2.74 m/s = 9.85 km/h
		19	2.16 m/s = 7.78 km/h

It is easy to show the following time complexity. The intuition is that under the above constraints, the number of calls of $forward$ is $O\left(\mathcal{A}/(u^2)\right) = O\left(\mathcal{A}/(R^2)\right)$ and the time complexity of each call of $forward$ is $O(\delta)$.

Lemma 31 *If* $\mathbf{Con}^{\mathbf{SF}}_{\mathbf{dis}}$ *and* $\mathbf{Con}^{\mathbf{SF}}_{\mathbf{mob}}$ *are satisfied, the time complexity of* $\mathbf{SF}$ *is* $O\left(\mathcal{A}\delta/R^2\right)$.

CHAPTER VII

CONCLUSION

This dissertation focuses on the design and analysis of distributed primitives for mobile ad hoc networks. Three topics were proposed. In the first part of the dissertation, theoretical analyses were presented for a distributed token circulation algorithm, LR, that causes a token to continually circulate through all the nodes of a network. In particular, a loose upper bound and a rigorous worst-case analysis on the round length were proved for the static case (part of the results appeared in [6]). In the future work of this part, we are interested in identifying characteristics of graphs on which LR has linear round length; the counter-example graphs found so far have a complex recursive construction. We are also interested in defining realistic mobility model that would allow analysis of LR in the mobile case.

In the second part of the dissertation, a self-stabilizing mutual exclusion algorithm was proposed for mobile ad hoc networks (a preliminary version appeared in [7] and the journal version has been accepted by [9]). It was shown that mutual exclusion always holds and different levels of progress hold under different levels of constraints. Interesting topics in the future work of this part is to characterize the specific mobility patterns in which the progress property can be guaranteed, and to evaluate the usefulness of the heuristic by which the predefined ring is updated and compare it to others.

The third part of the dissertation presented two broadcasting protocols which propagate a message from a source node to all of the nodes in the network. Instead of relying on the frequently changing topology, the protocols depend on a less frequently changing and more stable characteristic — the distribution of mobile hosts. Constraints on distribution and mobility of mobile nodes were given which guarantee that all the nodes receive the broadcast data. In our future work, we are interested in mobility model which would allow analysis of designed protocols. This work is our first step in modeling the mobility of mobile nodes using a set of parameters (the distribution and velocity of mobile nodes) and analyzing our protocols' performance based on the values of the designed parameters.

REFERENCES

[1] D. Cavin, Y. Sasson, and A. Schiper, "On the accuracy of MANET simulators," in *Proc. 2nd ACM International Workshop on Principles of Mobile Computing*, Toulouse, France, 2002, pp. 38–43.

[2] *OPNET Modeler*, Available at http://www.opnet.com/products/modeler/home.html, 2004.

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

[4] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla, "Glomosim: A scalable network simulation environment," Technical Report 990027, Department of Computer Science, University of California at Los Angeles, May 1999.

[5] N. Malpani, N. H. Vaidya, and J. L. Welch, "Distributed token circulation on mobile ad hoc networks.," in *Proc. 9th International Conference on Network Protocols(ICNP)*, Riverside, CA, Nov. 2001, pp. 4–13.

[6] N. Malpani, Y. Chen, N. Vaidya, and J. Welch, "Distributed token circulation in mobile ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 4, no. 2, pp. 154–165, March/April 2005.

[7] Y. Chen and J. L. Welch, "Self-stabilizing mutual exclusion using tokens in mobile ad hoc networks," in *Proc. 6th Annual International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM'2002)*, Atlanta, Georgia, 2002, pp. 34–42.

[8] C. Basile, M. Killijian, and D. Powell, "A survey of dependability issues in mobile wireless networks," Available at http://citeseer.nj.nec.com/basile03survey.html, February 2003.

[9] Y. Chen and J. L. Welch, "Self-stabilizing dynamic mutual exclusion for mobile ad hoc networks," *Journal of Parallel and Distributed Computing*, accepted, to appear.

[10] M. M. Carvalho and J. J. Garcia-Luna-Aceves, "A scalable model for channel access protocols in multihop ad hoc networks," in *Proc. 10th Annual International Conference on Mobile Computing and Networking*, Philadelphia, PA, 2004, pp. 330–344.

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

[12] E. M. Royer, S. R. Das, and C. E. Perkins, "Ad hoc on-demand distance vector (AODV) routing (IETF internet-draft)," Available at http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-08.txt, March 2001.

[13] G. Holland and N. H. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *Proc. 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM).* Seattle, WA, 1999, pp. 219-230.

[14] M. Mauve, J. Widmer, and H. Hartenstein, "A survey on position-based routing in mobile ad hoc networks," *IEEE Network Magazine*, vol. 15, no. 6, pp. 30–39, November 2001.

[15] K. P. Hatzis, G. P. Pentaris, P. G. Spirakis, V. T. Tampakas, and R. B. Tan, "Fundamental control algorithms in mobile networks," in *Proc. 16th ACM Symposium on Parallel Algorithms and Architectures*, Saint Malo, France, 1999, pp. 251–260.

[16] N. Malpani, J. L. Welch, and N. Vaidya, "Leader election algorithms for mobile ad hoc networks," in *Proc. 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Boston, MA, 2000, pp. 96–103.

[17] J. Walter, J. Welch, and N. Vaidya, "Mutual exclusion algorithm for ad hoc mobile networks," *Wireless Networks*, vol. 9, no. 6, pp. 585–600, Nov. 2001.

[18] B. Rajagopalan and P. McKinley, "A token-based protocol for reliable, ordered multicast communication," in *Proc. 8th IEEE Symposium on Reliable Distributed Systems.* Seatle, WA, October 1989, pp. 84-93.

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

[20] 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.

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

[22] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Commun. ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[23] A. K. Datta, C. Johnen, F. Petit, and V. Villain, "Self-stabilizing depth-first token circulation in arbitrary rooted networks.," in *5th International Colloquium on Structural*

Information and Communication Complexity (SIROCCO'98), Amalfi, Italy, June 1998, pp. 229–243.

[24] F. Petit and V. Villain, "Color optimal self-stabilizing depth-first token circulation for asynchronous message-passing systems," in *Proc. 10th Conference on Parallel and Distributed Computing Systems*, New Orleans, Louisiana, Oct, 1997, pp. 227–233.

[25] S. Dolev, E. Schiller, and J. Welch, "Random walk for self-stabilizing group communication in ad hoc networks," in *Proc. 21st Annual Symposium on Principles of Distributed Computing*, Monterey, CA, 2002, pp. 259–259.

[26] S. Vasudevan, N. Immerman, J. Kurose, and D. Towsley, "A leader election algorithm for mobile ad hoc networks," Technical Report 0301, Department of Computer Science, University of Massachusetts at Amherst, January 13, 2002.

[27] M. Schneider, "Self-stabilization," *ACM Computing Surveys*, vol. 25, pp. 45–67, 1993.

[28] S. Dolev and T. Herman, "Superstabilizing protocols for dynamic distributed systems.," *Chicago J. Theor. Comput. Sci.*, vol. 1997, Available at http://cjtcs.cs.uchicago.edu/articles/1997/4/contents.html, 1997.

[29] S. Dolev, "Optimal time self-stabilization in uniform dynamic systems," *Parallel Processing Letters*, vol. 8, no. 1, pp. 7–18, 1993.

[30] T. Herman, "Self-stabilization bibliography: Access guide," *Chicago Journal of Theoretical Computer Science*, vol. 1996, Available at http://citeseer.ist.psu.edu/herman98selfstabilization.html, 1996.

[31] R. Baldoni, A. Virgillito, and R. Petrassi, "A distributed mutual exclusion algorithm

for mobile ad-hoc networks," in *Proc. 7th IEEE Symposium on Computers and Communications (ISCC 2002)*, Taormina, Italy, 2002, pp. 539–544.

[32] D. R. Kowalski and A. Pelc, "Broadcasting in undirected ad hoc radio networks," in *Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing*, Boston, MA, 2003, pp. 73–82.

[33] S. Ni, Y. Tseng, Y. Chen, and J. Sheu, "The broadcast storm problem in a mobile ad hoc network," in *Proc. 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Seattle, WA, 1999, pp. 151–162.

[34] H. Lim and C. Kim, "Multicast tree construction and flooding in wireless ad hoc networks," in *Proc. 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, Boston, MA, 2000, pp. 61–68.

[35] W. Peng and X. Lu, "On the reduction of broadcast redundancy in mobile ad hoc networks," in *Proc. 1st ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Boston, MA, 2000, pp. 129–130.

[36] A. Laouiti, A. Qayyum, and L. Viennot, "Multipoint relaying: An efficient technique for flooding in mobile wireless networks," in *Proc. 35th Annual Hawaii International Conference on System Sciences*, Big Island, HI, 2002, pp. 298–307.

[37] J. Sucec and I. Marsic, "An efficient distributed network-wide broadcast algorithm for mobile ad hoc networks," Tech. Rep. CAIP 248, Rutgers University, Piscataway, NJ, September 2000.

[38] E. Pagani, "Providing reliable and fault tolerant broadcast delivery in mobile ad-hoc networks," *Mobile Networks and Applications*, vol. 4, no. 3, pp. 175–192, 1999.

[39] B. Williams and T. Camp, "Comparison of broadcasting techniques for mobile ad hoc networks," in *Proc. 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Lausanne, Switzerland, 2002, pp. 194–205.

[40] Vamsi Paruchuri, Arjan Durresi, and Raj Jain, "Optimized flooding protocol for ad hoc networks," *CoRR*, vol. cs.NI/0311013, Available at http://arxiv.org/abs/cs.NI/0311013, 2003.

[41] H. Takagi and L. Kleinrock, "Optimal transmission ranges for randomly distributed packet radio terminals," *IEEE Trans. Commun.*, vol. 32, no. 3, pp. 246–257, Mar. 1986.

[42] B. Karp and H. T. Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in *Proc. 6th Annual International Conference on Mobile Computing and Networking*, Boston, MA, 2000, pp. 243–254.

[43] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward, "A distance routing effect algorithm for mobility (dream)," in *Proc. 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Dallas, TX, 1998, pp. 76–84.

[44] Y. Ko and N. H. Vaidya, "Location-aided routing (LAR) in mobile ad hoc networks," in *Proc. 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Dallas, TX, 1998, pp. 76–84.

[45] L. Blazević, L. Buttyán, S. Capkun, S. Giordano, J. Hubaux, and J. Le Boudec, "Self-organization in mobile ad-hoc networks: The approach of terminodes," *IEEE Communications Magazine*, vol. 39, no. 6, pp. 166–174, June 2001.

[46] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," in *Proc. 6th Annual International Conference*

on Mobile Computing and Networking (MOBICOM '00), Boston, MA, 2000, pp. 120–130.

[47] K. Seada, A. Helmy, and R. Govindan, "On the effect of localization errors on geographic face routing in sensor networks," in *Proc. 3rd International Symposium on Information Processing in Sensor Networks*, Berkeley, CA, 2004, pp. 71–80.

[48] D. Niculescu and B. Nath, "Trajectory based forwarding and its applications," in *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking*, San Diego, CA, 2003, pp. 260–272.

[49] D. Karger, Personal communication, Massachusetts Institute of Technology, Cambridge, MA, 2001.

[50] M. Lauer and M. Matthes, "MOBICOM poster: Elan—an e-learning infrastructure for ad-hoc networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, no. 1, pp. 53–55, 2003.

[51] G. Varghese, "Self-stabilization by counter flushing," in *Proc. 13th Annual ACM Symposium on Principles of Distributed Computing*, New York, NY, 1994, pp. 244–253.

[52] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister, "Smart dust: Communicating with a cubic-millimeter computer," *IEEE Computer*, vol. 34, pp. 44 – 51, Jan. 2001.

[53] T. Imielinski and S. Goel, "Dataspace: Querying and monitoring deeply networked collections in physical space," *IEEE Personal Communications Magazine*, vol. 7, no. 5, pp. 4–9, October 2000.

[54] T. Imielinski and B. R. Badrinath, "Wireless graffiti - data, data everywhere matters.," in *Proc. 28th International Conference on Very Large Data Bases (VLDB)*, Hong

Kong, China, August 2002, pp. 9–19.

[55] M. Shur and S. Wagner, “Sensitive skin,” *IEEE Sensors Journal*, vol. 1, no. 1, pp. 41–51, June 2001.

[56] E. Kaplan, *Understanding GPS*, Artech House, Boston, MA, 1996.

[57] J. Hightower and G. Borriella, “Location systems for ubiquitous computing,” *IEEE Computer*, vol. 34, no. 8, pp. 57–66, 2001.

[58] S. Capkun, M. Hamdi, and J. Hubaux, “GPS-free positioning in mobile ad-hoc networks,” in *HICSS ’01: Proc. 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 9*, Washington, DC, Jan 2001, pp. 9008–9017.

[59] S. Singh and C. S. Raghavendra, “PAMAS: Power aware multi-access protocol with signalling for ad hoc networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 3, pp. 5–26, 1998.

[60] Y. Xu, J. Heidemann, and D. Estrin, “Geography-informed energy conservation for ad hoc routing,” in *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking*, Rome, Italy, July 2001, pp. 70–84.

[61] R. Gallager, “A perspective on multiaccess channels,” *IEEE Trans. Inform. Theory*, vol. 31, pp. 124–142, 1985.

[62] R. Bar-Yehuda, O. Goldreich, and A. Itai, “On the time-complexity of broadcast operations in multi-hop radio networks: an exponential gap between determinism and randomization,” *Journal of Computer and Systems Science*, vol. 45, pp. 104–126, 1992.

[63] A. E. F. Clementi, A. Monti, and R. Silvestri, "Selective families, superimposed codes, and broadcasting on unknown radio networks," in *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, Washington, DC, 2001, pp. 709–718.

[64] S. S. Kulkarni and U. Arumugam, "Collision-free communication in sensor networks," in *Proc. 6th Symposium on Self-Stabilizing Systems (SSS)*, San Francisco, CA, June 2003, pp. 17–31.

VITA

Yu Chen was born in Fu Zhou, Fu Jian Province, China. She received her B.Eng. and M.S. degree in Computer Science from Zhejiang University, P.R. China, in 1997 and 2000. She began pursuing a Ph.D. degree in Computer Science at Texas A&M University in 2000 and received her degree in August 2005. She worked as a graduate teaching assistant and research assistant for Dr. Jennifer L. Welch in the Department of Computer Science, Texas A&M University. Her research interests include distributed computing, self-stabilization and mobile computing. Her permenant address is: CangXia apartments, JiaXing 1-202, Taijian district, Fuzhou, Fujiang, 350009, P. R. China.