DISTRIBUTED SERVICES FOR MOBILE AD HOC NETWORKS

A Dissertation

by

GUANGTONG CAO

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

DISTRIBUTED SERVICES FOR MOBILE AD HOC NETWORKS

A Dissertation

by

GUANGTONG CAO

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
	Narasimha Reddy
	Yoonsuck Choe
Head of Department,	Valerie E. Taylor

August 2005

Major Subject: Computer Science

ABSTRACT

Distributed Services for Mobile Ad Hoc Networks. (August 2005)

Guangtong Cao, B.E., Beijing University of Aeronautics & Astronautics;

M.E., Beijing University of Aeronautics & Astronautics

Chair of Advisory Committee: Dr. Jennifer L. Welch

A mobile ad hoc network consists of certain nodes that communicate only through wireless medium and can move arbitrarily. The key feature of a mobile ad hoc network is the mobility of the nodes. Because of the mobility, communication links form and disappear as nodes come into and go out of each other's communication range. Mobile ad hoc networks are particularly useful in situations like disaster recovery and search, military operations, etc. Research on mobile ad hoc networks has drawn a huge amount of attention recently. The main challenges for mobile ad hoc networks are the sparse resources and frequent mobility. Most of the research work has been focused on the MAC and routing layer. In this work, we focus on distributed services for mobile ad hoc networks. These services will provide some fundamental functions in developing various applications for mobile ad hoc networks. In particular, we focus on the clock synchronization, connected dominating set, and k-mutual exclusion problems in mobile ad hoc networks.

To My parents

ACKNOWLEDGMENTS

First, I would like to thank my advisor Dr. Jennifer L. Welch. Without her inspiration, guidence and encouragement, this dissertation would not be possible. She led me into the exciting field of distributed computing and taught me how to approach problems in a rigorous way. 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. Yoonsuck Choe, and Dr. Narasimha Reddy, for their valuable comments and earnest help. Special thanks to Dr. Jennifer Walter for many helpful discussions.

I also thank the fellow students in my research group, Yu Chen, 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 owe them so much.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A. Mobile Ad Hoc Networks

The last decade has witnessed great advances in distributed computing and communication technologies. The advances in miniaturization and low-cost, low-power design have had a huge impact in distributed computing and networking. A new kind of network called mobile ad hoc network (MANET) emerged. A mobile ad hoc network ([1]) is formed by a set of nodes that communicate only through wireless medium. A node can communicate directly with another node only if they are within each other's transmission range. Otherwise, the message may traverse multiple hops to reach the destination node, and each node in the network functions as a router to relay messages.

Unlike traditional wired networks or cellular networks, a mobile ad hoc network does not have any infrastructure or central control. In a mobile ad hoc network, each node is free to move arbitrarily. Because of the mobility, communication links form and disappear as nodes come into and go out of each other's communication range. The frequent topology changes due to the mobility poses a serious challenge to algorithms for mobile ad hoc networks. A mobile ad hoc network is particularly useful in some "ad hoc" environments such as disaster recovery and search, military operation, game, etc. Another big application area of ad hoc network is the sensor network, which a large amount of tiny sensors are deployed in a certain area, and computing is done by collaboration of the data by each autonomous sensor. Sensor networks usually have less mobility than mobile ad hoc networks.

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

B. Distributed Services for Mobile Ad Hoc Networks

Due to various application areas, research on mobile ad hoc networks has drawn huge attention in recent years. However, these unique features of mobile ad hoc networks bring a lot of new challenges. First, as each node in a mobile ad hoc network is powered by battery, energy consumption is the first and foremost issue in any algorithm design. The message complexity of an algorithm should be as small as possible to save the battery. Second, because of the mobility, many existing algorithms for wired networks which assume a stable network topology become inapplicable in mobile network networks. Algorithms for mobile ad hoc networks must be adaptable to the frequent topology changes and partitions due to the mobility.

So far, most of the research work has been focused on the MAC [1] and routing layer [2], which provides the fundamental functionality for communication. However on the other hand, little research has been done on distributed services for mobile ad hoc networks. Distributed services provide some fundamental functionalities for many applications and algorithms for mobile ad hoc networks. These services are usually sit above the routing layer and beneath the application layer in the layer system design infrastructure. Distributed services and middleware have been extensively studied in traditional networks, including, for example, the Transis group communication system [3]. However, the unique features of mobile ad hoc networks bring new challenges for distributed services.

Most early works on distributed services and applications for mobile ad hoc networks simply apply existing services for wired networks directly to mobile ad hoc networks ([1]), assuming that the network layer provides the fundamental functionality. This approach follows exactly the existing layered system architecture of network protocols, as shown in Figure 1.

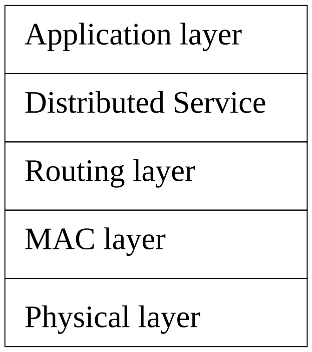


Fig. 1. Distributed Services without Mobility Awareness

However, some researchers found that simply applying the existing services to MANET is not efficient and sometimes even not correct ([4, 1]). Instead, distributed algorithms with "mobility awareness" is much more efficient. Figure 2 illustrates the new system architecture. Distributed services in the new system architecture can directly utilize some MAC layer functionalities such as broadcast to get the updated topology information without going through the complicated routing layer. In this dissertation, we study distributed services for mobile ad hoc networks. In particular, we focus on three distributed services: clock synchronization, connected dominating set and k-mutual exclusion in mobile ad hoc networks.

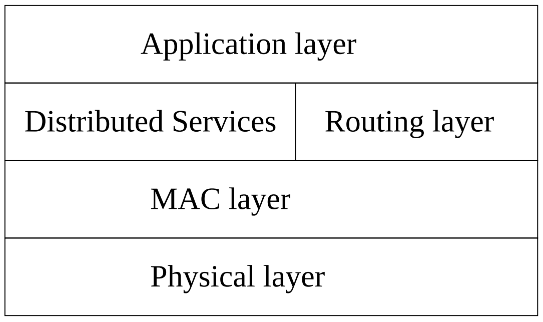


Fig. 2. Distributed Services with Mobility Awareness

Synchronized clocks are important for many distributed applications. Mobility and constraints in wireless communication bring new challenges for clock synchronization in mobile ad hoc network, making existing clock synchronization algorithms inapplicable in mobile ad hoc networks. In the first part of the dissertation, we introduce a new clock synchronization algorithm for mobile ad hoc networks ([5]). Our algorithm uses a synchronization technique called Reference Broadcast Synchronization (RBS) [6] in a single broadcast domain and extends RBS to the entire network using a connected dominating set (CDS). Our algorithm has the advantage of high accuracy in synchronization and adaptability to the mobility and requires fewer messages than existing algorithms.

Our clock synchronization algorithm is not the only place in which a connected dominating set is useful. In fact, a connected dominating set (CDS) can act as a general backbone in a mobile ad hoc network for many purposes like broadcasting, routing, information management, etc. Most of the algorithms to find a CDS require expensive message exchange and do not consider energy efficiency, which is a very important issue in wireless networks. In the second part of this dissertation, we introduce a randomized connected dominating set algorithm which can reduce the message overhead in finding the CDS and balance the energy consumption ([7]). Although the size of the CDS found by our algorithm is larger than the best CDS algorithms developed ([8]), it has more redundancy in routing. We also show that a routing algorithm using a CDS as a backbone can always find a routing path whose length is within a constant of that of the path found by using the same algorithm on the whole network.

In the third part of this dissertation, we study the problem of k-mutual exclusion. Mutual exclusion is a fundamental concept in operating system for resource sharing and task management ([9]). K-mutual exclusion extends mutual exclusion by allowing at most k shared resources to be accessed simultaneously in the system. Mutual exclusion and k-mutual exclusion deal with exclusive access to shared information. One application is data fusion in mobile ad hoc networks. In the third part of our research, we propose an algorithm called K-reversal Link with token forwarding (KRLF). KRLF is based on the KRL algorithm[4]. We show that KRLF provides more concurrency than KRL such that a request to enter the critical section is granted even if at most $k - 1$ processors are in the critical section forever.

C. Organization of the Dissertation

The rest of the dissertation is organized as follows. Chapter II presents our work on clock synchronization in mobile ad hoc networks. Chapter III discusses the randomized connected dominating set for mobile ad hoc networks. In Chapter IV, we present our k-mutual exclusion algorithm, in which we will define a new property for k-mutual exclusion. Chapter V concludes this dissertation and discusses future work.

CHAPTER II

CLOCK SYNCHRONIZATION IN MOBILE AD HOC NETWORKS

A. Motivation

Many distributed systems need synchronized clocks to work correctly For example, bank transaction systems need synchronized clocks to do snapshots and transaction commitment, and network protocols rely on synchronized clocks to measure message delay. Clock synchronization is listed as a core service for middleware systems by the Internet RFC ([10]).

Clock synchronization is also important in mobile wireless networks. As we mentioned in chapter I, energy efficiency is one of the most important design issue in wireless networks. MAC layer protocols for wireless networks are designed to put the wireless interface into a "sleep" mode when there is no communication going on in order to save energy. Wireless nodes wake up periodically to communicate with each other ([11]). But in order to do that, each node should be aware when it should wake up and start the communication. This requires clock synchronization. In sensor networks, a key application is event monitoring in which a large number of sensors are deployed in a designated area, and coordinate to monitor a certain event ([12]). Usually the data observed by each sensor will be collected and correlated for further analysis and processing, and synchronized clocks are critical for getting the correct result.

Though the clock synchronization problem for wireless networks can be solved by equipping each node with a device like a GPS receiver, a GPS receiver's functionality is limited due to its geographical constraints, high energy consumption and cost. Therefore, studying clock synchronization algorithms for mobile ad hoc networks has great significance.

The clock synchronization problem has been well studied in wired networks. Various synchronization bounds have been derived under many message delay assumptions and network topologies. NTP is the most widely deployed clock synchronization protocol in the Internet ([13]). However, mobile ad hoc networks bring new challenges to many network protocols and services. Particularly for clock synchronization, mobility can cause a pre-configured synchronization hierarchy (as in NTP) to become invalid. Moreover, since messages between nodes usually traverse multiple hops and because of the contention for the wireless communication medium among the mobile nodes, the round trip time of a message is very hard to measure. These uncertainties make the existing clock synchronization protocols and algorithms like NTP inapplicable in mobile ad hoc networks. Researchers tried to develop some new clock algorithms that overcome these uncertainties. Elson et al. ([6]) have proposed a clock synchronization algorithm for wireless sensor networks called Reference Based Synchronization (RBS). By taking advantage of the property of the physical layer broadcast communication in the wireless network, RBS can achieve much higher accuracy for synchronization than traditional synchronization schemes. But RBS only works within one broadcast domain, and the "time routing" method proposed in [6] for synchronization in multiple broadcast domains does not work in the case of mobility.

In this chapter, we propose a new clock synchronization algorithm for multi-hop mobile ad hoc networks. Our approach works in two tiers. First, we still take advantage of broadcast like RBS does to synchronize the clocks within one broadcast domain. In synchronization of multiple hops, we use an approximate Minimum Connected Dominating Set (MCDS). Only CDS nodes send out reference broadcast messages, and synchronization is relayed between neighboring broadcast domains of CDS nodes and extended to multiple hops. The size of the CDS generated by our algorithm is within a constant factor of the optimal, and message complexity is linear in the number of nodes in the network. Using a CDS greatly reduces the number of broadcast messages for synchronization as compared with other algorithms like [14]. In fact, our clock synchronization algorithm can be applied on any connected dominating set. In the next chapter, we introduce a randomized CDS algorithm that finds a larger CDS, but using fewer messages. Applying our clock synchronization algorithm on the randomized CDS could give a larger synchronization error because of the larger CDS size. We give an upper bound for our algorithm on how closely clocks can be synchronized to the reference clock in the presence of clock drift and mobility. Simulation shows that our approach can achieve very good synchronization accuracy in the entire network.

The rest of the chapter is organized as follows. Section B discusses the related work. In section C, we briefly review NTP and discuss the challenges which make NTP not applicable in mobile ad hoc networks. System model and definitions are given in Section D. Section E briefly introduces the Reference Broadcast Synchronization (RBS). In Section F, we present our algorithm to generate and maintain an approximate Minimum Connected Dominating Set (MCDS) in mobile ad hoc networks. Section G discusses how to use the approximate MCDS and RBS to synchronize clocks in multiple hops and the performance analysis. Simulation results are presented in Section H.

B. Related Work

The clock synchronization problem has been well studied both theoretically and in practice. Synchronization of physical clocks was first introduced by Lamport ([15]). Algorithms have been developed for a variety of system models and failure assumptions. Upper and lower bounds on how closely the clocks can be synchronized have been proved.

Lundelius and Lynch [16] prove an upper bound of $u(1-\frac{1}{n})$ on how closely clocks can be synchronized in a fully connected network, where u is the uncertainty in the message delay and n is the number of nodes in the network. Halpern et al. prove a lower bound for an arbitrary network topologies with arbitrary symmetric message delay uncertainties ([17]). Cristian studies the clock synchronization problem when there is no bound for message delay and proposes a probabilistic clock synchronization mechanism which achieves a better precision of estimating remote processor's clock with a high probability ([18]). [18] also shows that high precision of clock synchronization requires more message exchange. Patt-Shamir and Rajsbaum study the best synchronization that is achievable for a given message pattern ([19]), so it is quite a different model, or approach, than the others.

In practice, NTP ([13]) has been widely used as the Internet clock synchronization protocol. NTP has a self-organized hierarchical synchronization infrastructure and is scalable to networks as large as the Internet. NTP uses unreliable unicast message exchange (UDP) to synchronize clocks. Fine-grained statistical algorithms are designed to filter messages with uncertain and large delays. Because of its robustness and self configurablity, NTP has become the most widely used time synchronization protocol in the Internet. According to a survey in 1999, NTP networks contains at least 175,000 hosts ([20]), that does not include hosts behind firewalls or only run NTP occasionally. Synchronization using global time sources like GPS in a broadcast LAN is studied in [21]. [21] is the first work that uses physical layer broadcast to synchronize computer clocks.

[22] and [6] study the clock synchronization in wireless networks using reference broadcast, but none of the algorithms works in multi-hop mobile ad hoc networks.

Mitra and Rabek ([23]) propose a cluster service for clock synchronization in sensor networks. The algorithm is similar to the cone-based topology control algorithm proposed by [24]. In their work, they assume every node can sense the direction of the incoming message, which is different with the system model in our research. [14] proposes a probabilistic clock synchronization algorithm for sensor networks based on RBS. [25] adopts a on-demanding strategy combining a push and pull mechanism to avoid unnecessary clock synchronization. [26, 27, 28] study the time synchronization in wireless sensor networks using unicast message exchange. [26, 27] use the same message delay estimation technique as NTP. [28] bounds the drift and offset estimation by a linear envelope. All of these algorithms synchronize clocks in multiple hops by a spanning tree in the network. In another direction, Römer proposes a time transformation function for multiple hop synchronization in [29].

[30] studies a new property called the gradient property in clock synchronization, which requires the synchronization error between two nodes in the network to be a function of the distance between the two nodes in terms of message uncertainty. [30] gives a lower bound which indicates that clock synchronization is not a local property, in that the worst case synchronization error between two nodes depends not only on the distance between the two nodes, but also on the diameter of the network.

Our approach to synchronize clocks in multiple hop ad hoc networks is based on the notion of a Minimum Connected Dominating Set (MCDS). A connected dominating set (CDS) can serve as a virtual backbone for mobile ad hoc networks and is widely used in broadcast and routing ([31, 32]). A CDS can reduce the number of broadcast messages dramatically. However, finding the minimum connected dominating set in a general graph and in a unit disk graph[1] are both NP-complete ([33, 34]).

[1]Unit disk graphs, defined in Section D, model the type of mobile ad hoc network we consider.

Many approximation algorithms have been proposed to generate a dominating set in general graphs ([35, 36, 37, 38]). Most of the algorithms reduce the problem of finding a dominating set to a set cover problem. If the set has to be connected, some heuristics are used to connect the set generated in the first stage. [37] gives two distributed implementations of the algorithm of [36], both of which have approximation ratio $O(\ln(\Delta))$, where Δ is the maximal degree of the graph. There are several algorithms developed to approximate the MCDS in unit disk graphs ([39, 8, 40, 41]). Most of these algorithms use the relationship between the size of the maximum independent set and minimum connected dominating set proved in [39]. [39] and [8] both generate an independent set and connect the set using a spanning tree. The approximation ratios are both 8 times the optimal solution, and both of the algorithms end within $O(n)$ time, and have message complexity $O(n \log n)$. However, neither of the two papers discusses how to maintain the approximate MCDS in the mobile situation.

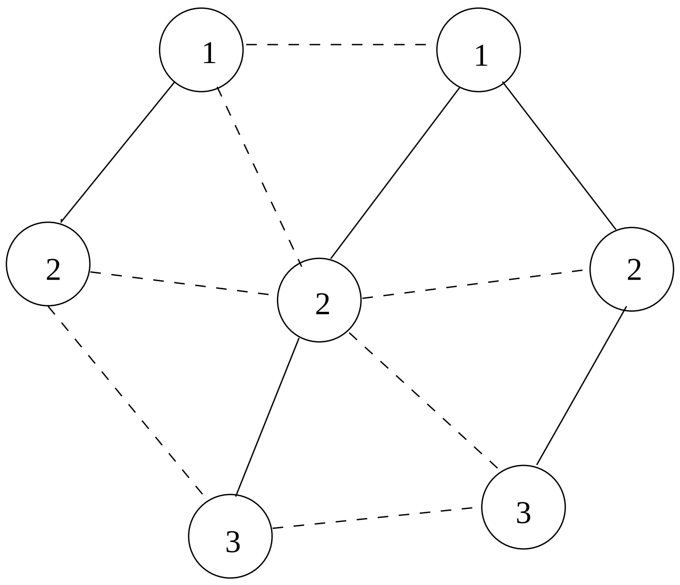


Fig. 3. A Typical NTP Network Topology

C. Challenges for Clock Synchronization in Mobile Ad Hoc Networks

In this section, we briefly review the Network Time Protocol (NTP) and analyze the challenges for synchronization techniques like NTP in mobile ad hoc networks. As shown later in the chapter, our clock synchronization algorithm creates a synchronization hierarchy that is similar to NTP. However, we use a broadcast based synchronization scheme instead of unicast message exchange in NTP to reduce the uncertainty in the synchronization.

Each NTP node has a variable called *stratum* indicating its distance to the top of the hierarchy. At the top of the tree are the stratum 1 clocks, which are nodes that have accurate time device (typically a GPS or WWVC receiver). Nodes that synchronize to the stratum 1 clocks become stratum 2 clocks. Each NTP node select a peer to synchronize with, and each node has several backup peers. All NTP nodes form a hierarchical synchronization network which is similar to the Domain Name Service in the Internet. Figure 3 illustrates a typical NTP network. Each node has a synchronization peer (denoted in the solid line) in the NTP network, and has several backup synchronization peers (in dash lines).

The basic procedure of synchronization in NTP is to use a two-way unicast message exchange to synchronize the clock to its peer. As shown in Figure 4, when a NTP node synchronizes its clock with its peer, it sends out a request message to the peer, with the timestamp of request t_1 in its own clock. When the peer receives the request, it records the reception time of the request in its own clock t_2, and sends a response in t_3, the node receives the response at t_4 in its own clock. Assuming that the clock drift and the propagation delay do not change in this small synchronization time span, NTP calculate the clock offset of the NTP node to its synchronization peer as $\frac{(t_2-t_1)-(t_4)-(t_3)}{2}$. $(t_2 - t_1) - (t_4 - t_3)$ indicates the difference in the message delay.

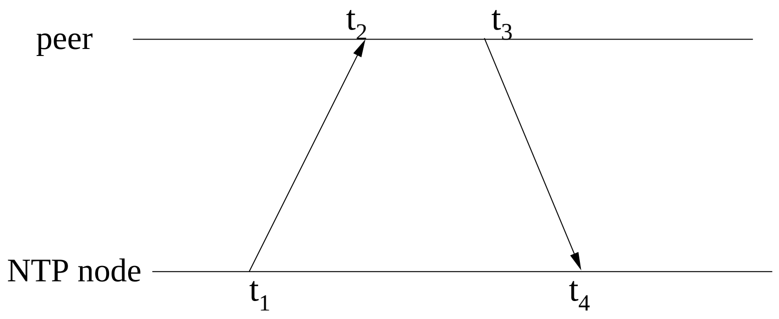


Fig. 4. An Illustration of Message Exchange in NTP Synchronization

The accuracy of NTP is dependent on the uncertainty in the communication. Kopetz et.al. characterizes the message delivery latency into four distinct components in [42], which are defined as *critical path*:

- send time – the time spent at the sender to construct the message. This includes the time for kernel processing, context switches, and system call overhead incurred by the synchronization application and is hence highly variable depending on the current system load.

- access time – delay incurred waiting for access to the transmit medium. This time depends on the MAC protocol in use and its methods to handle congestions. Typical wireless MAC protocols like IEEE 801.11 networks exchange RTS/CTS before the actual exchange of the message. Depending on the congestion in the network, this waiting time is most significant in terms of the total delay latency.

- propagation time – the time needed for the message to transit from sender to receiver. If the sender and receiver are in the same broadcast region, this time is typically very small. This time can be approximately calculated by dividing the distance between the sender and receiver by the speed of light. This time is negligible compared to the other delays.

- receive time – processing required for the receiver's network interface to receive the message from the channel and notify the host of it arrival. If the time-stamp of the reception can be done at a suitable low level, this delay can be made very small, and more importantly, deterministic.

Because of the contention in the MAC layer, the uncertainty in message delay in a mobile ad hoc network is much larger than that in the wired network. In addition, in a mobile ad hoc network, the hierarchical structure of NTP synchronization network is always violated by the frequent mobility of nodes. Therefore, synchronization algorithms for wired networks like NTP is not applicable in mobile ad hoc networks. A broadcast-based synchronization scheme is more applicable in mobile ad hoc networks, as introduced later.

D. System Model and Definitions

In this section, we introduce the notation and system model we use throughout the paper. First we model a mobile ad hoc network as a graph $G = (V, E)$, in which the set of vertices V represents the nodes in the network and the set of edges E represents the links between nodes. We assume that each node in the network has the same transmission power, so the network can be modeled as a *unit disk graph*, which is an undirected graph whose vertices are points in the 2-dimensional plane with an edge between any pair of nodes whose Euclidean distance is at most 1.

A *dominating set* D of G is a subset of V such that each node not in D has at least one neighbor in D. We call the nodes in D *dominators*, and the nodes in $V - D$ *dominatees*. If the induced subgraph of D is a connected graph, then D is called a *Connected Dominating Set* (CDS). A *Minimum Connected Dominating Set* (MCDS) is a CDS that has the minimum cardinality. An *independent set* I of G is a subset of V such that no nodes in I are neighbors. A *maximum independent set* is an independent set that has the maximum cardinality. Notice that a maximum independent set is also a dominating set, but not necessarily a minimum dominating set.

An algorithm that returns near-optimal solutions of a problem is called an *approximation algorithm*. The *relative error bound* of an approximation algorithm is defined as $\frac{|C-C^*|}{C^*}$, where C is the cost of the solution by the approximation algorithm, and C^* is the cost of the optimal solution. For many NP-complete problems, including finding the minimum connected dominating set, approximation algorithms that have a constant relative error bound and run in time polynomial in the size of the input are desirable.

Each node P_i is assumed to have a *hardware clock* $HC_i(t)$, which is a function from the real numbers (real times) to real numbers (the hardware clock times). We assume that the hardware clock for a node P_i is within a linear envelope of the real time. For all times t_1 and t_2, $t_2 > t_1$, $(1-\rho)(t_2-t_1) \leq HC_i(t_2) - HC_i(t_1) \leq (1+\rho)(t_2-t_1)$, ρ is a constant positive constant. The node P_i cannot change its hardware clock time; in order to set its clock, it alters an *offset* variable, offset_i, which is added to the hardware clock to produce the node's *virtual clock* value. Formally, the virtual clock $VC_i(t)$ of P_i is equal to $HC_i(t)+\mathit{offset}_i(t)$, where $\mathit{offset}_i(t)$ is the value of the variable offset_i at real time t. We assume there is a distinguished node P_0 equipped with GPS in the system whose clock serves as the "reference clock". The *synchronization error* of node P_i at time t is $|VC_i(t) - VC_0(t)|$. The *clock synchronization problem* with parameter γ is to update the *offset* variables at each node P_i (other than P_0) in order to bound the largest synchronization error by γ. Obviously we would like γ to be as small as possible.

We make some assumptions about the behavior of the mobile ad hoc network G:

- Every node maintains its one hop neighbor information. We denote the neighbor set as N. Neighbor information can be realized by each node periodically broadcasting and gathering hello messages from its one hop neighbors.
- Every node in G has a unique id.
- A node can receive every broadcast message from its neighbors, which means that broadcast message delivery is reliable. However, [31] shows that the deliverability of broadcast can go as low as 80% because of the hidden terminal problem, the contention of the broadcast channel, and the noise. Reliable broadcast is not the topic we are focused on here. We also assume that a node can receive all broadcast messages from neighbors within a finite time interval. We call this the *broadcast timeout interval*.

E. Reference Broadcast Synchronization

For synchronizing clocks in a single hop, we use the Reference Broadcast Synchronization (RBS) proposed in [6]. The goal of RBS is to eliminate the uncertainty in estimating remote clocks. As we mentioned earlier, due to the mobility and contention-based communication in mobile ad hoc networks, the message delay is even more difficult to measure than in the Internet. Instead of estimating the message delay as in traditional clock synchronization schemes like NTP, RBS takes an alternative approach by taking advantage of the following property of the broadcast-based communication medium: The difference in the reception time for the same broadcast message by two nodes is very small in a wireless network. The reason is the relatively short distance between nodes within a broadcast domain and the extremely fast propagation speed of a wireless communication channel.

In a broadcast domain, there is a specific node sending out reference broadcast

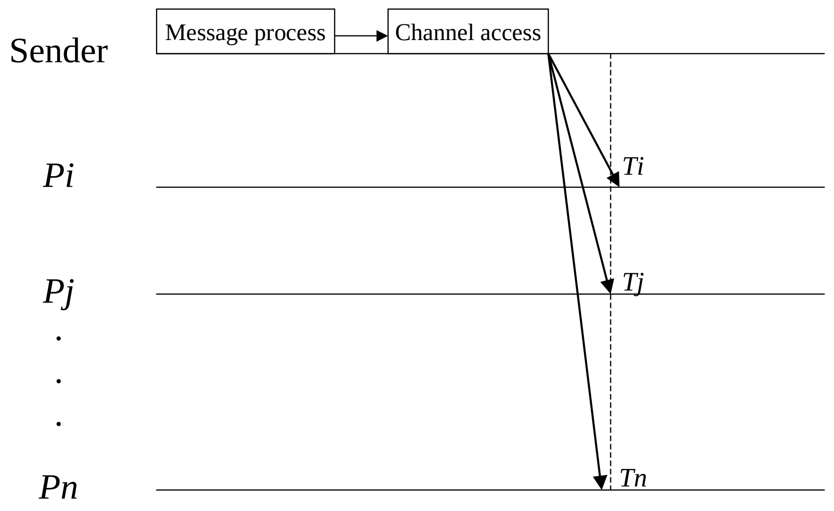


Fig. 5. Reference Broadcast Synchronization

messages. Nodes inside the broadcast domain synchronize their clocks using the reception time of the reference broadcast messages. RBS works as follows.

1. A node in the network periodically broadcasts reference beacons.

2. When a node in the broadcast domain receives a reference beacon, it records its virtual clock value at the time of reception of the reference beacon.

3. Nodes exchange their virtual clock values for the reference beacon arrival time.

4. Each node calculates the offset from the other nodes using the virtual clock values for the same reference beacon. As an example shown in Figure 5, node P_i's offset to P_j is $VC_i - VC_j$, where VC_i and VC_j are the virtual clock values of P_i and P_j for the same reference beacon.

As shown in [6], by using the physical layer broadcast, the most uncertain part for the synchronization, which is packet sending time and channel access time, is eliminated. Now the only uncertainty affecting synchronization is the difference in propagation time and the time spent to receive a reference beacon, which should be very small in a wireless network. As a result, the accuracy of estimating the offset between two nodes is much higher. [6] also shows that the difference in reception time of the broadcast packet follows a Gaussian distribution with high confidence statistically. By receiving multiple reference beacons and taking the average of the difference in the reception time for multiple reference beacons, the variance of the offset is smaller. Thus nodes can increase the precision of synchronization. Now P_i's offset to P_j is $\frac{1}{m}\sum_{k=1}^{m}(T_{i,k} - T_{j,k})$, where $T_{i,k}$ is the time on P_i's virtual clock when P_i receives the kth reference beacon, and $T_{j,k}$ is defined analogously for P_j. where m is the number of reference beacons, $T_{i,k}$ and $T_{j,k}$ are the the reception times on virtual clocks for the kth reference beacon by P_i and P_j respectively.

However, RBS only works within one broadcast domain. For multiple hops, [6] proposes a "time routing" scheme by picking some special nodes to broadcast reference beacons, such that the neighboring broadcast domains are guaranteed to overlap. Since nodes that are overlapped by two broadcast domains can receive reference beacons from both broadcast domains and calculate the offset of their clocks to the nodes within all the nodes within the two broadcast domains, they can act as the "time routing nodes" in the network. *But [6] does not discuss how these special nodes are selected or how to deal with mobility.* In a mobile ad hoc network, since all the nodes can move arbitrarily, the pre-selected special nodes can move so that there is no overlap between their broadcast domains, making this approach infeasible. *Our work proposes a solution to the selection of these special nodes by using the connected dominating set and synchronizes clocks in multi-hop ad hoc networks, as introduced next.*

F. Synchronization Backbone: Approximate MCDS

In this section, we introduce a distributed algorithm to approximate a minimum connected dominating set (MCDS) in a mobile ad hoc network. As shown later in Section G, we use the connected dominating set (CDS) to broadcast reference beacons and synchronize the clocks in multiple hops. Each node in the CDS has a layer variable indicating its hop distance to the root node of the network. The layer variable is used for the clock synchronization as will be explained. Our algorithm can achieve a constant approximation ratio in CDS size. Then we discuss how to maintain the connected dominating set under mobility. We argue that our algorithm is more suitable for clock synchronization because other algorithms with similar performance ([8, 39]) either are not adaptive to the mobility or do not have the layer information needed for synchronization. [43] generates a CDS whose size is within a constant factor of the optimal and is able to adjust to mobility, but the algorithm is fairly complicated and it requires neighbor information up to three hops, which is hard to achieve in highly mobile networks.

1. The Algorithm

Our algorithm is executed in a distributed fashion. Every node in the network runs a copy of the algorithm. The pseudo-code of the algorithm is shown in Figure 6.

Each node P_i has these local variables:

- *state*: The current state of the node, the state of a node can be **active, inactive, dominator or dominatee**. Initially *state* is set to **inactive**.
- *layer*: The shortest hop distance from P_i to the root P_0 in the spanning tree.
- *dominator*: the dominator of the current node.

- *inactive_degree*: the number of **inactive** neighbors of the node.

- N: P_i's neighbor set. A set of tuples (*id*, *state*, *layer*, *inactivedegree*) which stores P_i's neighbors' *id*, *state*, *layer* and *inactive_degree.* We define several operations on N. $N[i]$ returns the tuple in N in which $i = id$. $N.id$, $N.state$, $N.layer$ and $N.inactivedegree$ return the set of *id*, *state*, *layer* and *inactivedegree* of N , and $N[i].j$ is the element j of the tuple $N[i]$. N is initially empty.

Our algorithm finds an approximate MCDS by generating a spanning tree in the network. Every node has a *layer* in the tree which indicates its shortest hop distance to the root node P_0 in the tree, whose *layer* is 0. Every node except P_0 has a *dominator* in the tree which is the parent of the node in the tree. The *dominator*'s *layer* is one less than the *layer* of the node. The algorithm starts by the root node P_0 setting its *state* to **dominator** and broadcasting a **dominator** message to all its neighbors. After each neighbor node receives a **dominator** message, it sets its state to **active**, its *layer* to one plus the layer of the sender of the **dominator** message, and broadcasts an **active** message. After each **active** node receives the **active** messages from all its **active** neighbors, it will calculate the number of neighbors which are not **active**, and broadcast an **inactive_degree** message. We assume that a node can receive all broadcast messages from neighbors within a finite time interval. We call this the *broadcast timeout.* After a broadcast timeout, the node which has the maximum inactive neighbors among all its **active** neighbors will be set to a **dominator**. If two neighbor nodes have the same *inactive_degree*, the one with smaller id will become a **dominator**. If a node is not selected as a **dominator** at first, it will continue to compare the *inactive_degree* with its newly found **active** neighbors and make the decision again. If it doesn't have any **inactive** neighbors, it will become a

dominatee. A node may receive **dominator** messages from all its neighbors, it sets its own *dominator* to the neighbor which has the minimum (*layer*, *id*) tuple, and its *layer* to be one greater than the *layer* of its *dominator*.

```
Code for $P_0$:
  $layer \leftarrow 0$;
  $state \leftarrow$ dominator;
  broadcast(dominator, $P_0, layer$);

Code for $P_i$:

recv(dominator, $q, layer$)
  $N[q].state \leftarrow$ dominator;
  $N[q].layer \leftarrow layer$;
  if($state$ == inactive) {
     $state \leftarrow active$;
     broadcast(active, $P_i$);
     wait a broadcast timeout;
  }

recv(active,$q$)
  if($P_i.state$ == active and $N[q].state$ == inactive)
  {
     $inactive\_degree \leftarrow inactive\_degree - 1$;
  }
  $N[q].state \leftarrow$ active;

recv(inactive_degree, $q$, $inactdgr$)
  $N[q].inactivedegree \leftarrow inactdgr$;

at broadcast timeout
  $dominator \leftarrow$ dominator neighbor with the minimum $(layer, id)$;
  $layer \leftarrow N[dominator].layer + 1$;
  $inactive\_degree \leftarrow$ number of inactive neighbors;
  if ($inactive\_degree == 0$) $state \leftarrow$ dominatee;
  if(the first broadcast timeout after $P_i$'s state change to active)
  {
     broadcast(inactive_degree, $P_i$, $inactive\_degree$);
     wait a broadcast timeout;
  }
  else if ($P_i$ has the maximum $(inacitve\_degree, id)$ among its active neighbors)
  {
     $state \leftarrow$ dominator;
     broadcast(dominator, $P_i, layer$);
  }
  else wait a broadcast timeout;
```

Fig. 6. Pseudo-code of the Approximate MCDS Algorithm

2. An Example of the Algorithm

Figure 7 illustrates an execution of the algorithm. In the graph, the numbers are the IDs of the nodes, node 0 is the root node. The solid lines are the edges of the graph induced by the resulting approximate MCDS, such that if there are two dominator nodes P_i and P_j, and P_j sets its *dominator* to P_i according to the algorithm, there is a solid line linking P_i and P_j. The dashed lines are the other edges of the graph.

- Initially, node 0 sets its state to **dominator** and broadcasts a **dominator** message to its neighbors. We mark the dominator nodes black. Nodes 8, 1 and 3 receive the **dominator** message and set their states to **active**, which we mark with gray. The above procedure is shown in Figure 7(a).

- At the first broadcast timeout after the sending of the **active** message, nodes 8, 1 and 3 receive **active** messages from all their neighbors and calculate their own *inactive_degree*. Nodes 8 and 3 become **dominator** after another broadcast timeout because they have the maximum inactive degree among their **active** neighbors. Then 8 and 3 broadcast **dominator** messages, which will make node 4, 10, 5 and 2 set their states to **active**. After this phase, the **dominator** set is {0, 3, 8} and **active** nodes are {1, 4, 10, 5, 2}. The result is shown in Figure 7(b).

- When receiving **active** messages within a broadcast timeout, the **active** nodes 1, 4, 10, 5, 2 broadcast **inactive degree** messages and compare their own *inactive_degree* with the *inactive_degree*s of their **active** neighbors. As a result, nodes 4, 5, 2 become the **dominator**s and broadcast **dominator** messages, making nodes 6, 9, 7, 11 set their states to **active**. Node 1 finds it has no **inactive** neighbors, so 1 sets its state to **dominatee**. Node 10 remains in the

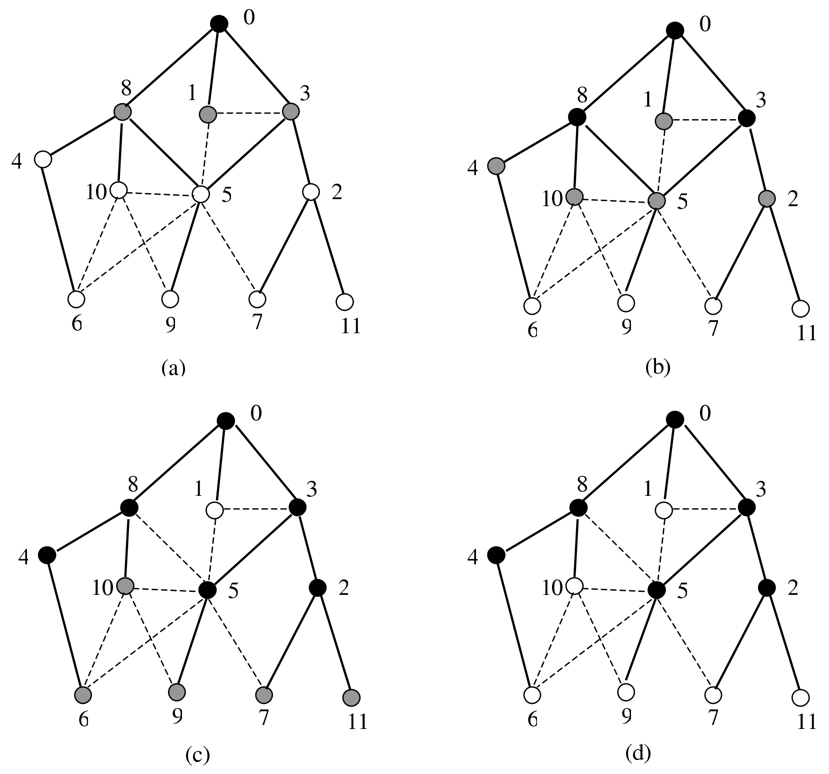


Fig. 7. An Example of the Execution

active state. After this phase, the **dominator** set is {0, 3, 8, 4, 5 ,2}, **active** nodes are {10, 6, 9, 7, 11}, and the **dominatee** set is {10}. This is shown in Figure 7(c).

- After broadcast timeout, every **active** node (10, 6, 9, 7, 11) finds it has no **inactive** neighbors, so they all set their states to **dominatee**, the **dominator** nodes are {0, 3, 8, 4, 5, 2}, and the **dominatee** nodes are {1, 10, 6, 9, 7, 11}. The finding of the approximate MCDS ends as Figure 7(d).

3. Correctness and Performance Analysis

In this section, we analyze our algorithm. We first prove that the algorithm finds a connected dominating set, and then we prove that the size of the connected dominating set generated by our algorithm is within a constant factor of the MCDS. Also, we show that the time and message complexity of the algorithm are both linear in the number of nodes in the network.

Theorem 1 *Our algorithm generates a connected dominating set.*

Proof. To prove this, we first show that eventually, all the nodes in the network with the state dominator form a dominating set, which means every node in the network will have at least one neighbor whose state is dominator. From the pseudo-code, if a node has the maximum *inactive_degree* among all its active neighbors, it will be added into the dominating set. Otherwise, the node is going to wait for another timeout and then decides if it's going to be a dominator by comparing its *inactive_degree* with its neighbors. Eventually, the node will either become a *dominator* or a *dominatee*. Every dominatee will have at least one dominator neighbor.

Next we show that the set is connected. Since every node in the resulting dominating set is set by changing its state from active to dominator (except P_0), and every active node has a *dominator*, this indicates that every dominator will have at least one neighbor which is also a dominator, so the resulting dominating set is connected.

■

Lemma 1 *In every broadcast timeout, the newly generated dominators are independent.*

Proof. At each timeout interval, a node will compare the number of its inactive neighbors to all its neighbors, and sets itself to **dominator** if it has the maximal inactive neighbors. Since we break ties using node id, this eliminates the possibility that two neighbor nodes become dominator at the same time. ■

To prove the constant factor of approximation, we rely on the result of [39], which shows the relation between the size of a minimum connected dominating set and the size of a maximum independent set.

Lemma 2 *In a unit disk graph, the size of a maximum independent set is at most* $4 \times opt + 1$*, where opt is the size of a minimum connected dominating set.*

Proof. The detailed proof is given by Alzoubi et.al. in [39], which basically finds the largest possible independent set given any MCDS and uses the fact that a single node in a unit disk graph can have at most 5 independent neighbors. Here we assume that the network topology does not change during the generation of the approximate MCDS. Note this assumption is only for the convenience of the approximation ratio analysis. As we show later in this section, the heuristic we propose is able to maintain the connectivity of the dominating set in the presence of mobility, but the size of the CDS in a mobile situation may not be within the bound we prove. ■

Theorem 2 *The size of the connected dominating set generated is at most* $8 \times opt + 2$.

Proof. To prove this, we use the result of Lemma 2 , which quantifies the relationship between the size of a maximum independent set and the size of a minimum connected dominating set.

Let S be the dominator set that our algorithm finds. We divide S into two subsets S_1 and S_2. S_1 is the set of nodes that are directly generated at the first broadcast timeout after changing their states to active, S_2 is the set of nodes that are generated by taking extra timeout (see line 28 of the pseudo code in Figure 6), which means they are not selected as the member of the connected dominating set at first because there are neighbors which have larger *inactive_degree*. We further divide the nodes in S_1 according to their value of the layer into odd layers and even layers, denoting as S_1^{odd} and S_1^{even}.

Now we claim that S_1^{odd} and S_1^{even} are two independent sets. This is easy to verify. Each node in S_1^{odd} is selected as a dominator at the first broadcast timeout after its state changed to active, which means it doesn't have any neighbor belong in S_1^{odd}, so S_1^{odd} is an independent set. The same argument holds for S_1^{even}.

Then we look at a node P_j in the set S_2. Without loss of generality, we assume P_j is in an odd layer. P_j is not selected for the dominating set at first because it has at least one neighbor, say P_i, that has more inactive neighbors. P_j is selected for the dominating set later because there are still neighbors which are not activated by the neighbors of the same layer or the next layer. We pick a node from these, P_k. Note P_k and the nodes in S_1^{odd} are independent. Let $R = S_1^{odd} \cup \{P_k\}$. According to Lemma 2, we have $|R| \leq 4\times\text{opt}+1$. We then replace P_k by its *dominator* P_j, and we get $|S_1^{odd} \cup \{P_j\}| = |R|$. If P_j is in S_1^{even}, we also have the same argument. So we have at most $2 \times (4\times\text{opt}+1) = 8\times\text{opt}+2$ nodes in our connected dominating set. ∎

Theorem 3 *The message complexity of the algorithm is $O(n \log n)$, and the time complexity is $O(n)$.*

Proof. In the algorithm, every dominator node broadcasts three messages, active, inactive_degree and dominator message, except the root node, which only broadcasts dominator messages. The dominatees only broadcast a constant number of active and inactive_degree messages. Since each message only includes a node's state, id and degree, each message is $O(\log n)$ bits, so the message complexity is $O(n \log n)$.

The approximate MCDS found by our algorithm induces a spanning tree in the network. Since at least one new dominator node is found in every two broadcast timeouts and there are n nodes in the network, the time complexity of finding an approximate MCDS is $O(n)$. ■

4. Maintaining the CDS in the Presence of Mobility

In a mobile ad hoc network, each node can move around with an arbitrary speed and direction. Due to the mobility, the existing connected dominating set may become disconnected or redundant. In this section, we discuss how to maintain a connected dominating set dynamically. We assume that the mobility does not cause a partition of the network, otherwise a connected dominating set is not possible. Our goal here is to keep the size of the dominating set small and keep it connected.

The update procedure works as follows.

- When a dominatee node moves and is not the neighbor of its original dominator, as long as it has dominator neighbors, it resets its dominator to the one with the minimal $(layer, id)$ value. In this case, the dominating set is not changed.

- When a dominator node loses connection to its dominator, as long as it has a dominator neighbor whose layer is less than its own, it still maintains its dominator state and sets it *dominator* to the dominator neighbor with the minimal $(layer, id)$ and its layer to 1 plus the new *dominator*'s layer.

- When a dominator or a dominatee node moves and has no dominator neighbor, it will choose a neighbor with the minimal $(layer, id)$ as its new *dominator* and send out a notification to that neighbor. After receiving the notification message, the neighbor will become a dominator node.

- At each broadcast timeout, if a node finds it has no neighbors setting itself as the dominator, which means all of its neighbors have other dominators, it changes to a dominatee.

As shown in the above procedure, if a dominator is not the neighbor of its original dominator, as long as it has a dominator neighbor, it only takes one broadcast timeout interval to get the CDS reconnected. Dominatee moves do not impact the connectivity of the CDS. The worst situation is when a dominatee moves and it has no dominator neighbors. In this case, the node will request one of its neighbors to be a dominator, and it will take two broadcast timeout interval to reconstruct the CDS. As shown in section H, our heuristic can keep down the size of the CDS by eliminating redundant dominators from the CDS.

G. Multi-hop Synchronization

1. The Algorithm

Our clock synchronization algorithm works in two tiers. For synchronization in multiple hops, we use the approximate MCDS generated in section F as a hierarchical synchronization infrastructure. The layer of the GPS node is 0. Inside a broadcast domain of a CDS node, we still use RBS. Synchronization is started by the root node broadcasting reference beacons and is propagated along the spanning tree induced by the approximate MCDS. Each node in the first layer synchronizes its clock with the GPS node after receiving the reference beacon from first layer CDS nodes.

Figure 8 illustrates a synchronization graph for a mobile ad hoc network. Every dominator node has a broadcast domain represented by a disk, and a layer is assigned to each broadcast domain by the approximate MCDS algorithm. The solid lines are the edges between dominator nodes. The dotted lines represent the synchronization

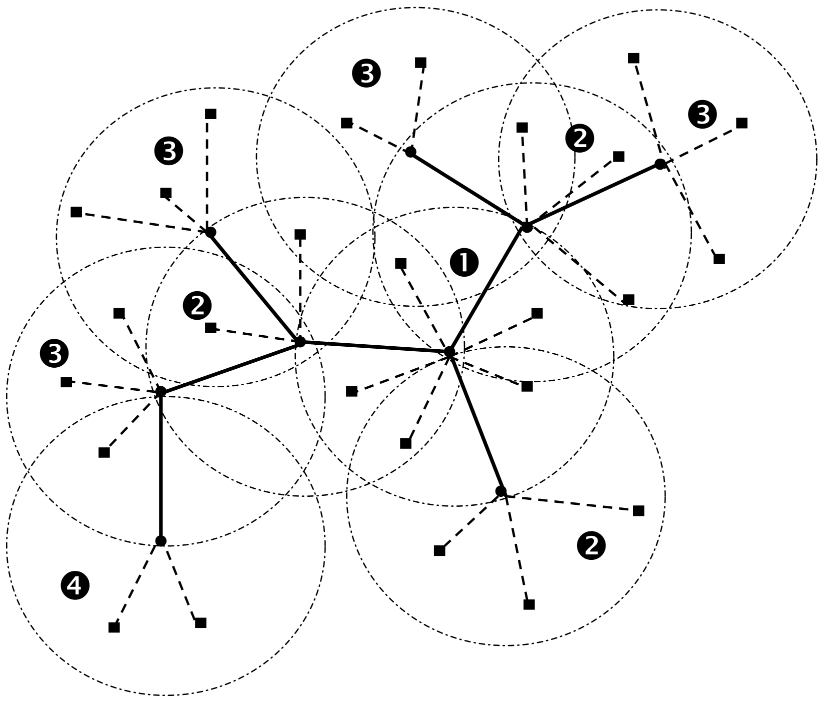


Fig. 8. A Synchronization Graph for a Mobile Ad Hoc Network

inside the broadcast domain of the dominator nodes. The pseudo-code of the clock synchronization algorithm is shown in Figure 9. Recall that we denote by VC_i the virtual clock of a node P_i, and by $offset_i$ the offset variable of P_i. The procedure of clock synchronization is as follows.

1. The root node P_0 broadcasts m reference beacons in sequence to all its neighbors. The layer of the dominator node is also included in the reference beacons. Note that m is a parameter that can be adjusted.

2. All the nodes within the broadcast domain receive the reference beacons from a dominator. Each records the reception times on its virtual clock for each of the m reference beacons.

3. If a node's layer is less than the layer of the dominator, it sends its reception time (on its virtual clock) for each of the m reference beacons back to the dominator node. For the first layer, the GPS node sends its reception time of the GPS clock for each of the reference beacons back to the dominator node.

4. For each k, $1 \leq k \leq m$, after the dominator node receives all the reception times from its neighbors with lower layers or from the GPS node for the kth reference beacon, it broadcasts the average of the virtual clocks to all its neighbors as a reference clock for the kth reference beacon.

5. The node dominated by the dominator node receives all the m reference clocks from the dominator and calculates its m offsets to each reference clock. Then it sets its offset to be the average of the offsets. If it is a dominator node, it broadcasts m reference beacons and synchronizes the clocks of the nodes in the next layer using the procedure above.

As we see from the above procedure, the nodes with higher layers use the reception time of lower layer nodes to calculate their clock offsets and achieve synchronization. The nodes in lower layers can achieve more accuracy in synchronization than the nodes in higher layers. This approach creates a layered synchronization network similar to NTP. Meanwhile, the high accuracy in synchronization using reference broadcast is preserved and extended to the whole network through the CDS. But since only CDS nodes send reference beacons, the number of broadcast messages is lowered, and the synchronization infrastructure is adaptive to mobile situations.

Also notice that our synchronization algorithm can be easily integrated with the other CDS algorithms used by many backbone based routing algorithms. If the reference beacons can be piggybacked in other packets for routing purposes, the overhead of the clock synchronization is reduced even further.

```
Code for $P_0$:

start clock synchronization
    for ($seqno$ from 1 to $m$)
        broadcast(**Reference_Beacon**, 0, $seqno$);

Code for a dominator node $P_i$:

recv(**Reference_Ack**, $VC_j$, $seqno$) from $P_j$
    $Reference\_Clock[j, seqno] \leftarrow VC_j$;
    if(received **Reference_Acks** from lower layer neighbors or GPS node
              for the $seqno$th reference_beacon )
    {
        $avg \leftarrow$ Average($\sum_k$Reference_Clock[k,seqno]),
              such that $P_k$ is a lower layer neighbor or the GPS node
        broadcast(**Reference_Clock**, $avg$, $seqno$);
    }

Code for a node $P_j$:
recv(**Reference_Beacon**, $l$, $seqno$) from $P_i$
    if($layer < l$)
        send(**Reference_Ack**, $VC_j$, $seqno$) to $P_i$;
    else if($P_i$ is my $dominator$)
        $reception\_time[seqno] \leftarrow VC_j$;

recv(**Reference_Clock**, $avg$, $seqno$) from $P_i$
    if($P_i$ is my $dominator$)
    {
        $offset[seqno] \leftarrow reception\_time[seqno] - avg$;
        if(received reference clocks for all $m$ reference_beacons)
            $offset_j \leftarrow \frac{1}{m}\sum_{k=1}^{m} offset[k]$
        if(I am a dominator node)
            for($seqno$ from 1 to $m$)
                broadcast(**Reference_Beacon**, $layer$, $seqno$);
    }
```

Fig. 9. Pseudo-code for Clock Synchronization Algorithm

Our algorithm also can be modified easily for synchronizing a subset of the nodes only, in which CDS nodes act as the "time routing" nodes in [6] for the time transformation. In this case, only CDS nodes in the time routing path broadcast synchronization beacons. This will also reduce the number of broadcast messages when network-wide synchronization is not necessary.

2. Performance Analysis

In this section, we analyze the relation between the synchronization error and layer. As we mentioned earlier, a node's layer is the number of hops to P_0, which is maintained in our approximate MCDS algorithm. We first assume the synchronization error within one broadcast domain is bounded by σ using RBS. This assumption is justified by [6], in which the authors observe that the variance of the reception time of a reference beacon by nodes within the same broadcast domain follows a normal distribution with average of 0 and variance of 11.1 μsec with confidence level of 99.8%. For each $k \leq 1$, We define $\delta_k(l)$ to be the maximum synchronization error immediately after the kth resynchronization is finished for any node P_i such that P_i is in layer l. First, without considering clock drift, we show that $\delta_1(l)$ increases with l as follows.

Theorem 4 *If each d_i=1, $\delta_1(l)$ is at most $\sigma \times \lceil \frac{l}{2} \rceil$.*

Proof. According to our algorithm, suppose a dominator node of layer l broadcasts a reference beacon. The nodes at layer $l-1$ will send their reception times back to the dominator and these reception times will be used to synchronize the clocks of the dominatees at layer $l+1$. Given that the nodes in layer $l-1$ have synchronization error bounded by $\delta_1(l-1)$, the synchronization error of the $l+1$ layer nodes is bounded by $\delta_1(l+1) \leq \delta_1(l-1) + \sigma$. Nodes in the first layer synchronize their clocks with the GPS node, so $\delta_1(1) \leq \sigma$ and $\delta_1(0) = 0$. Putting the two claims above together, we get the synchronization error of a layer l node, $\delta_1(l)$, is at most $\sigma \times \lceil \frac{l}{2} \rceil$. ■

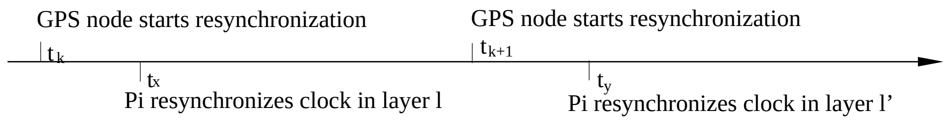
Now we analyze $\delta(l)$ with clock drift. Due to the clock drift, the clock value in each node will drift apart after synchronizing the upper layer nodes. To analyze the synchronization error with clock drift, we bound the time for the synchronization in one broadcast domain by β. β is reasonably small (within a few seconds) considering that only a few messages are sent by each node in a single broadcast domain.

Theorem 5 *With clock drift, $\delta_1(l)$ is bounded by $(\sigma + \rho\beta) \times \lceil \frac{l}{2} \rceil$.*

Proof. In our algorithm, after a dominator node in layer l broadcasts a reference beacon, the nodes at layer $l-1$ will send their reception times back to the dominator, but due to the clock drift, the synchronization error of the layer $l-1$ nodes will increase after the synchronization with upper layer nodes. Since the drift rate is bounded by ρ, the maximum synchronization error of the layer $l-1$ nodes during the synchronization with the nodes in layer $l+1$ is at most $\delta_1(l-1) + \rho\beta$. Therefore, the synchronization error of the $l+1$ layer nodes is bounded by $\delta_1(l+1) \le \delta_1(l-1) + \rho\beta + \sigma$. Since $\delta_1(0) = 0$ and $\delta_1(1) \le \sigma + \rho\beta$, we have $\delta_1(l) \le (\sigma + \rho\beta) \times \lceil \frac{l}{2} \rceil$. ■

As clock drifts are always subject to environmental changes, it is necessary to resynchronize clocks periodically. A resynchronization begins when P_0 broadcasts a reference beacon. We assume that P_0 starts resynchronizations periodically, with a period of τ as measured on its clock. Since P_0 is assumed to have a perfect clock, τ is also the amount of real time that elapses between resynchronizations. A sequence number for each synchronization round is included in the reference beacons. Now we analyze the synchronization error for a node P_i between the two consecutive resynchronizations. Suppose P_i is in layer l at the kth resynchronization, and P_i is in layer l' at the $k+1$th resynchronization. Due to the mobility, l' is not necessarily the same as l.

Theorem 6 *Suppose node P_i is in layer l during the kth resynchronization round and is in layer l' during the $k+1$th resynchronization round. Then the maximum synchronization error experienced by P_i between the kth and the $k+1$th resynchronizations is at most $\delta_k(l) + \rho(\tau - l\beta + l'\beta)$.*



GPS node starts resynchronization
tk
tx
Pi resynchronizes clock in layer l
GPS node starts resynchronization
tk+1
ty
Pi resynchronizes clock in layer l'

Fig. 10. A Resynchronization Round

Proof. As shown in Figure 10, after the GPS node starts the kth resynchronization round at real time t_k, P_i finishes synchronization at real time t_x. As P_i is in layer l and the time for synchronization in each layer is bounded by β, t_x is bounded by $t_k + l\beta$. At the $k+1$th resynchronization round, which starts at real time t_{k+1}, P_i is in layer l' and P_i finishes its resynchronization at real time t_y. Note that $t_y \leq t_{k+1} + l'\beta$. Thus $t_y - t_x$, the amount of real time that elapses between subsequent resynchronization by P_i is at most $(\tau + l'\beta - l\beta)$. As clock drift is bounded by ρ, the error caused by clock drift before the next resynchronization is at most $\rho(\tau - l\beta + l'\beta)$. P_i's synchronization error at the kth resynchronization is by $\delta_k(l)$. Therefore, the maximum synchronization error before the next resynchronization is $\delta_k(l) + \rho(\tau - l\beta + l'\beta)$. ■

In the worst case, l' is $n-1$, which is the highest layer, and P_i's synchronization error after the $k+1$the resynchronization is $\delta_k(l) + \rho(\tau + (n-1-l)\beta)$. By induction from Theorem 5 and Theorem 6, the upper bound of $\delta_k(l)$ is a monotonic increasing function. However, this upper bound may not be tight. Our intuition is that after each resynchronization, a nodes adjusts its virtual clock value according to the offset computed at the resynchronization, which prevents the continuous increase of the clock offset. As shown in the simulation, the synchronization error after each resynchronization is within a certain range, which is consistent with our intuition. Deriving a tight upper bound for our algorithm is left for future work. Also notice that our result is related to the synchronization error after each resynchronization, which is different than the the result of [30], which gives a lower bound on the worst case synchronization error for any two node in the network at any time, stating that the worst case the synchronization error is a global property depending on the diameter of the network.

H. Simulation

In this section, we describe the results of simulating our clock synchronization algorithm in a mobile ad hoc network. First, we compare the performance of our approximate MCDS algorithm with the MinID and MaxDegree algorithms presented in [8]. The reason we pick the algorithms in [8] is that both of the algorithms in [8] also have an approximation of 8 to the optimal solution in the unit disk graph model, which are the best algorithms developed in the literature so far. Both MinID and MaxDegree assume a spanning tree with a root node in the network, and compute the CDS based on each node's id or degree. The two algorithms proposed in [39] also have constant approximation ratio, but [8] also shows that the algorithms in [8] outperform the algorithms in [39] in simulation. So in this work, we only compare our algorithm with [8].

We then simulate our clock synchronization algorithm using the approximate MCDS generated by our algorithm. As a comparison, we simulate a synchronization scheme similar to NTP in a mobile ad hoc networks in [44]. The synchronization error with that scheme is within the scale of 10^{-3} seconds, whereas our synchronization scheme is much better in terms of synchronization error.

1. Simulation Model

The simulation is implemented using the ns-2 network simulator from USC/ISI/LBNL [45], with wireless extensions from the CMU Monarch project [46]. IEEE 802.11 is the MAC layer protocol. For every wireless node, the bandwidth is 2Mbps and radio transmission radius is 250 meters. We set the initial position of every wireless node randomly in a 1000×1000 meter simulation box. The number of nodes varies from 25 to 200, which covers the typical simulated scenarios for a mobile ad hoc network.

All of the algorithms are implemented in the agent layer. The agents use DSR as the routing protocol. A neighbor discovery agent is implemented to maintain the latest neighbor information. The neighbor discovery agent sends and receives HELLO messages to neighbors using MAC layer broadcast at a hello timeout. The hello timeout is set to 2 seconds in this simulation. The mobility pattern follows the Random Way Point (RWP) model of the ns-2. We simulate nodes moving with average speed of 5, 10 and 15 meters per second. For each speed, we create 5 random mobility files, each of which is 1000 seconds long.

For synchronization, we set the parameter m to 3, which is similar to [6]. Every node is initialized with an offset value randomly chosen between -0.001 and 0.001 seconds. First, we simulate our algorithm with zero drift. The node's clock value is the sum of the clock of the ns-2 simulator and the offset. The results are relevant to the behavior of the algorithm after a single resynchronization. Then we simulate with non-zero drift, the results are relevant to the behavior of the algorithm in the long run. The GPS node's clock value is set equal to the clock value of the ns-2 simulator (thus has offset 0).

2. Simulation Results

As mentioned earlier, the size of the connected dominating set is the most important factor of the approximate MCDS algorithm. A smaller connected dominating set means that fewer broadcast messages are sent. Furthermore, reducing the number of dominators reduces the number of layers, and thus the bound in Theorem 5 is smaller. We also measure the size of the connected dominating set maintained under different average moving speeds of the nodes in the network. In measuring the CDS size, we take the snapshot of the network by having each node report its status (dominator or dominatee) periodically after the CDS is generated. The CDS size is calculated by counting the number of dominator nodes. Each plotted point is the average of 30 snapshots, which is shown in Figures 11 - 12.

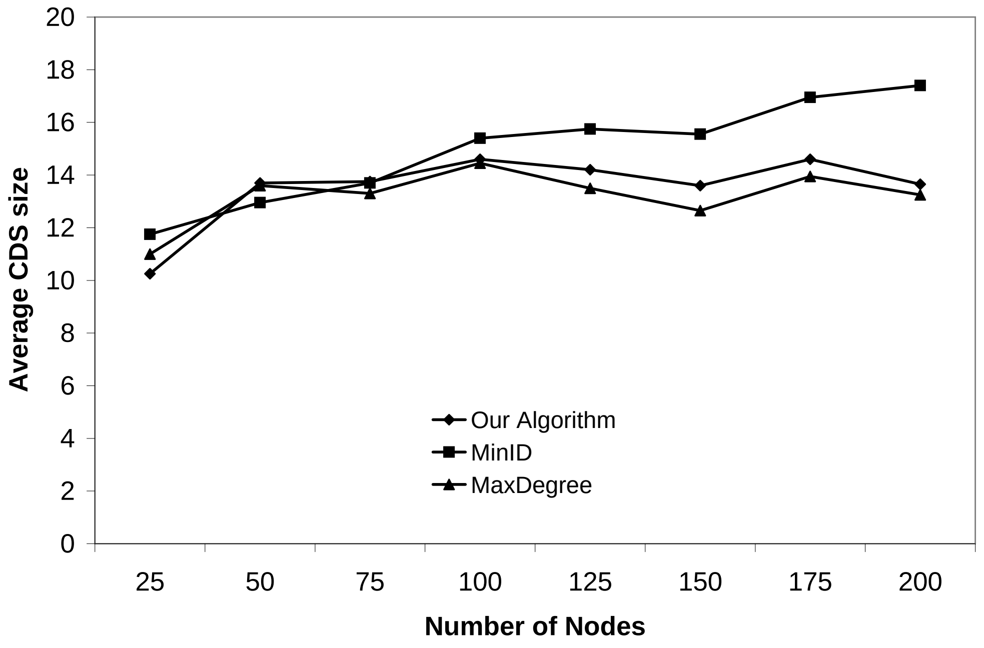


Fig. 11. Average CDS Size vs. Number of Nodes with No Mobility

Figure 11 shows the size of the approximate MCDS versus the number of nodes with no mobility. As shown in Figure 11, the size of the connected dominating set generated by our algorithm is smaller than that of the first algorithm in [8] and a little bigger than that of the second algorithm in [8]. The curve of our algorithm is similar to the second algorithm in [8]. The reason for the similarity in the size of the approximate MCDS between our algorithm and the second algorithm of [8] is that both of the algorithms use the degree of neighbors to determine the dominating set, while the first algorithm of [8] only uses node id and layer information. The average size of the dominating set in a 1000 $\times$ 1000 square meters simulation box is around 13 in most of the simulation scenarios, which reflects the number of grid squares (250 $\times$ 250 square meters) covering the simulation box. And as shown in the plot, the size of the connected dominating set does not increase as the number of the nodes increases in the network. This indicates that the size of the dominating set only depends on the area which the mobile nodes are within.

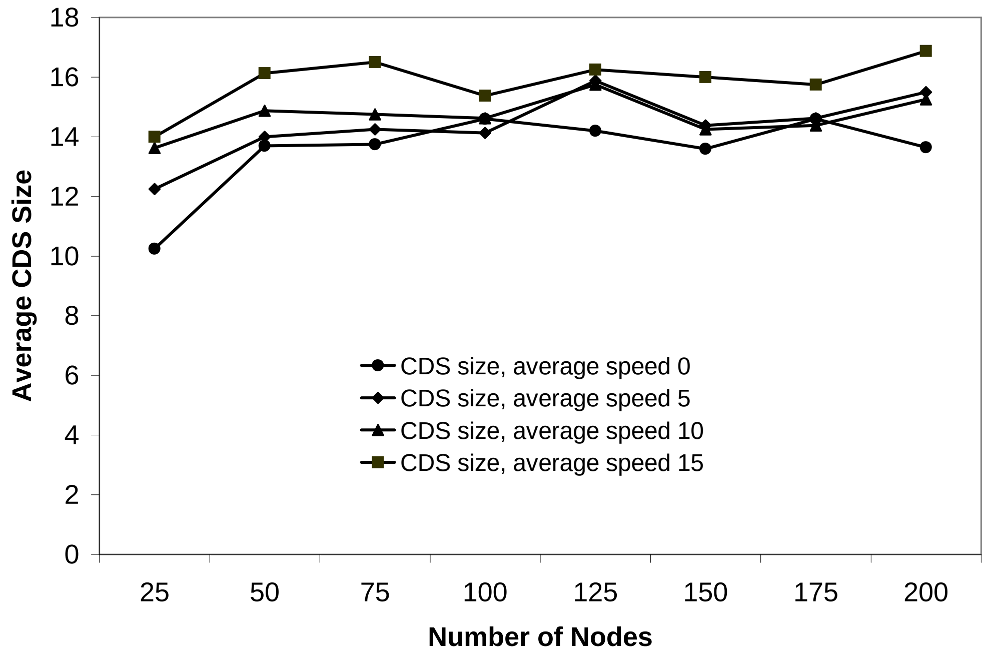


Fig. 12. Average CDS Size vs. Number of Nodes with Mobility

Figure 12 shows the approximate MCDS sizes under different average speeds. As expected, the size of the CDS with high speeds is bigger than the size with low speeds. However, the maintenance heuristic eliminates the redundant dominator nodes which do not have any dominatees while choosing new dominators, which keeps the size of the CDS from increasing unreasonably in the mobile case.

For synchronization, we measure for each layer, the *average* synchronization error after synchronization is complete, taken over all the nodes in that layer. We also measure, for each layer, the difference between the largest and smallest virtual clock in that layer, which we call the *synchronization variance*.

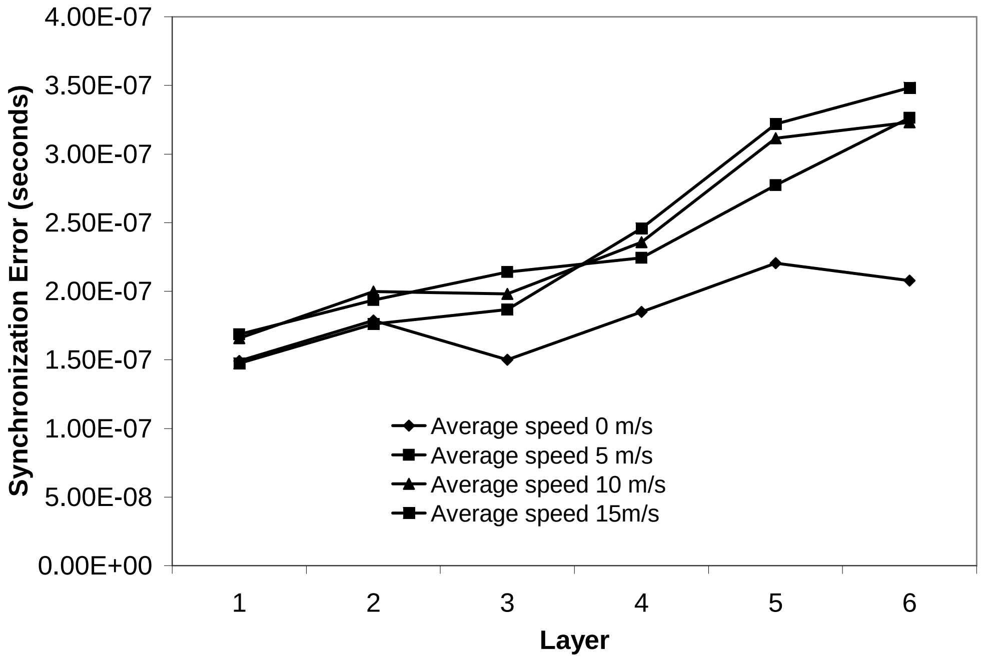


Fig. 13. Synchronization Error vs. Layer with No Drift

Figure 13 and Figure 14 are the simulation results with no clock drift. Each plotted point shown in Figures 13 - 14 is the average of 50 executions. Figure 13 shows the synchronization error under different average speeds. The curve of the figure does not show the relation between synchronization error and layer as described in Theorem 5. This is because Theorem 5 provides a worst case bound, and the

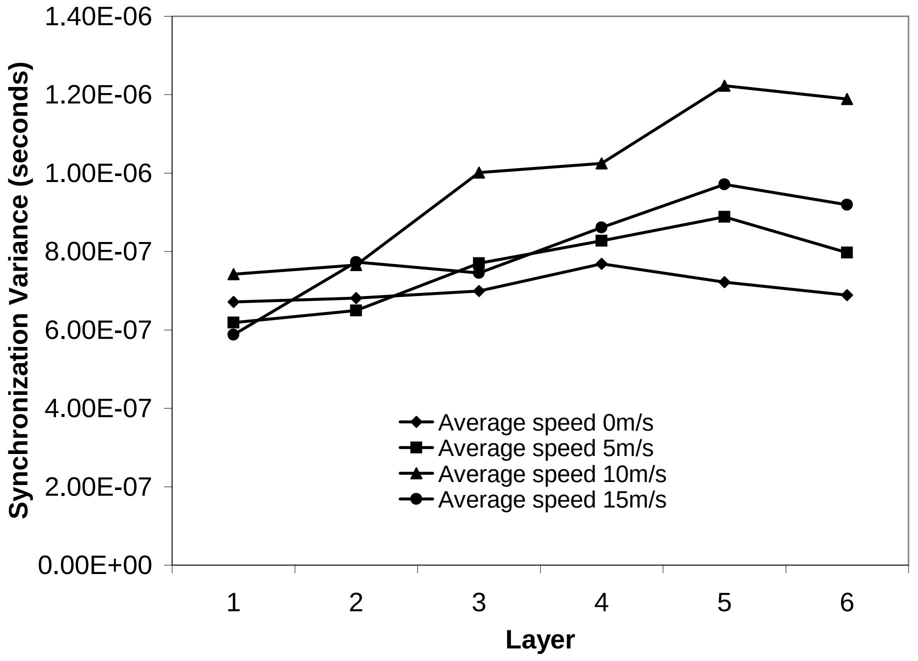


Fig. 14. Synchronization Variance vs. Layer with No Drift

error in most situations does not increase as much as in the worst case. However, the synchronization error does increase as the layer number increases, and the synchronization error in higher average speeds is bigger than that in lower average speeds. The synchronization error is within the scale 10^{-7} seconds, which is smaller than the result in [6]. The reason is that the difference in the reception times of a reference beacon in the ns-2 simulator is smaller than that in the real experimental platform used in [6]. The synchronization error and variance in a real mobile ad hoc network could be larger than our simulation results.

Figure 14 shows the synchronization variance versus the layer under different average speeds. As shown in Figure 14, the synchronization variance increases as the layer increases, and synchronization variance for higher layer nodes is bigger than for lower layer nodes. This is consistent with the statistical property of the RBS mentioned in [6]. However, the curve for the case with average speed of 15m/s does not show the increase in synchronization variance as expected. This is because the synchronization is done at the same time as the update of the approximate MCDS. Since nodes move fairly fast in this case, some nodes may have outdated layer information during the synchronization.

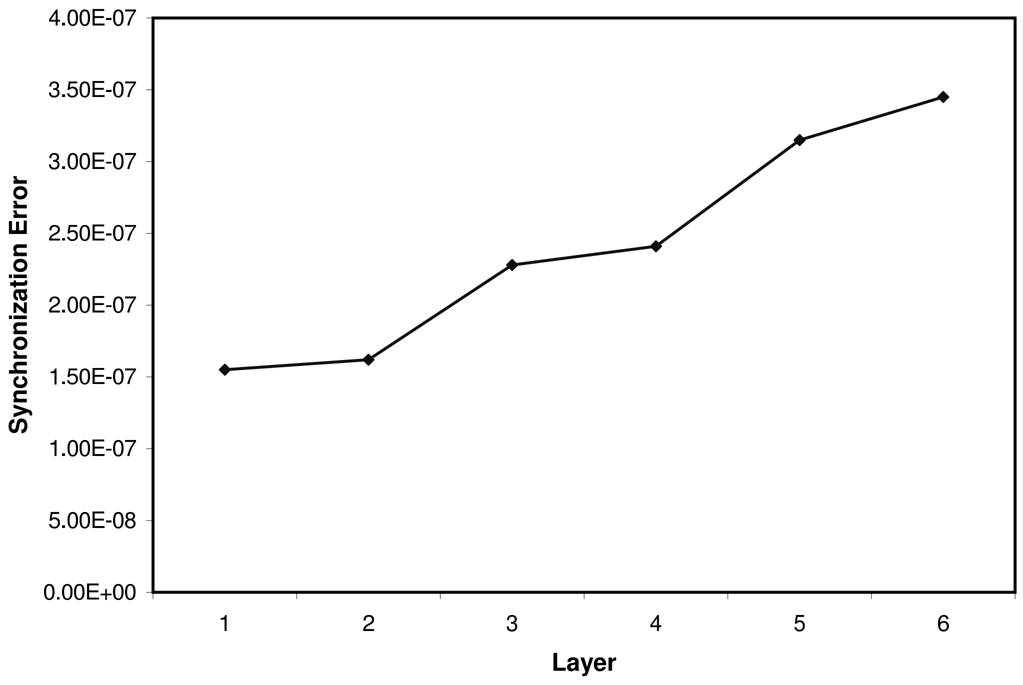


Fig. 15. Synchronization Error vs. Layer with Drift

We also simulated our algorithm with clock drift. We model each node's clock drift rate as a uniform random function between $(1-10^{-6}, 1+10^{-6})$. At the beginning of each resynchronization, each node calculates its new clock drift according to the uniform random function. The average speed in the simulation is $5m/s$. Resynchronization interval is set to $40s$. Simulation runs in 1000 seconds. The plotted data is an average of 20 runs. In our simulation, we calculate the synchronization error after each resynchronization. Figure 15 shows the synchronization error in different layers. Note that the synchronization error increases with the layer number, which is the same trend observed in Figure 13. Our algorithm still keeps the synchronization error within 10^{-6} seconds after the resynchronization. The synchronization error after each resynchronization does not increase as compared to Figure 13 because the main factor that impacts the synchronization error is the difference in the reception of the broadcast messages, which are the same in both cases.

CHAPTER III

A RANDOMIZED CONNECTED DOMINATING SET ALGORITHM IN WIRELESS AD HOC NETWORKS

A. Introduction

In Chapter II, we use a connected dominating set as the synchronization backbone to broadcast reference beacons and relay the synchronization hierarchically In fact, the usage of a connected dominating set can go far beyond clock synchronization. For various applications such as routing ([47]), broadcasting ([41]) and information gathering ([48]), a connected dominating set (CDS) often serves as a virtual backbone. Having a virtual backbone provides a relatively stable topology in the network and reduces a great number of routing request messages. However, since all network nodes are free to move, the connected dominating set has to be recomputed every so often to keep its size small and maintain connectivity. Developing ways to find and maintain a connected dominating set will have a great impact in mobile ad hoc networks.

Although many existing algorithms can find a connected dominating set whose size is within a constant ratio of the optimal, it is often expensive in terms of messages and time to maintain the connected dominating set in the presence of mobility. In addition, since backbone nodes usually do more transmission and computing than other nodes in the network, they consume more power. Having a fixed set of nodes act as the backbone will make the battery of these nodes be consumed much faster than the other nodes, and as a result, disconnect the network. To extend the network lifetime, it is desirable to let all nodes in the network share the responsibility of being in the connected dominating set by switching nodes in and out of the connected dominating set, while at the same time keep the size of the CDS small.

In this chapter, we present an algorithm to randomly construct and maintain a connected dominating set in mobile ad hoc networks. Each node runs the algorithm periodically to determine whether it is a CDS node based only on its local information. Our algorithm periodically switches the connected dominating set nodes by selecting new nodes for the CDS and removing nodes from the old CDS. A random delay function is used for the selection of such new nodes to keep the size of the newly selected set small. The random delay function ensures that newly selected CDS nodes have the most remaining battery power with high probability. By periodically switching CDS nodes, our algorithm makes the energy consumption of the CDS evenly distributed among all nodes in the network, thereby extending the network lifetime. Meanwhile, the size of the CDS found by our algorithm is small enough for efficient routing and the number of messages sent by our algorithm is fewer than that in the existing algorithms. Simulation shows that our algorithm has good performance in terms of the CDS size and number of messages.

The remainder of the chapter is organized as follows. In Section B, we review the related work on connected dominating set and energy-efficiency design in wireless ad hoc networks. The system model and assumptions are discussed in Section C. Section D presents our randomized CDS algorithm. In section G, we discuss a further optimization on the result of our algorithm. In particular, we discuss the efficiency of a routing algorithm based on the randomized connected dominating set. Section E discusses how to recompute the CDS. Simulation results are shown in Section F.

B. Related Work

A connected dominating set is widely used as a virtual backbone in wireless ad hoc networks. [47, 49, 32] show through simulation that routing on a connected dominating set can reduce the number of routing request messages significantly. [31] and [50] both study the problem of broadcasting in wireless networks, and both point out that using a dominating set based broadcast can reduce the number of redundant broadcast messages. [48] uses a backbone which is a connected dominating set in essence to manage distributed data in a wireless network.

The connected dominating set problem has been extensively studied both in general graphs and in unit disk graphs, which are commonly used to model wireless networks. Some related work on connected dominating set has been discussed in Chapter II ([37, 36, 51, 39, 8, 41, 40]). As finding the *Minimum Connected Dominating Set (MCDS)* is an NP-complete problem ([33, 34]), all algorithms above find an approximate MCDS.

[52] first studied the issue of balancing clusters in a wireless ad hoc network. The heuristic they proposed is based on a virtual id and timer for each node. No analysis is given in their paper. [53] proposes a randomized algorithm called Span to find and maintain the backbone nodes. Span uses a random delay function in each node to decide whether it is a backbone node. The random delay function takes the node's remaining battery level and the number of pairs of nodes it can connect into account. Our algorithm uses the random delay similarly, but in a much simpler form. In [54], the network is divided into fixed grids. Nodes inside each grid switch between sleeping and listening mode to forward messages. The grids are divided in a way such that each pair of nodes inside a grid can communicate with each other directly. Our algorithm is different in that we do not use the geographic information.

In another direction, many topology control protocols have been proposed to adjust the transmission power in order to minimize the energy consumption and achieve high performance ([55, 24]). The main idea of topology control is to adjust the transmission power for each wireless node in order to generate a network topology with certain properties. In particular, many geometric graphs are used for topology control purposes, including relative neighborhood graph (RNG), Gabriel Graph (GG), and Delauny triangulation, etc. These graphs have certain properties such as planarity and low neighborhood cardinality that are attractive for communications in wireless networks. However, there are some drawbacks in the power control algorithms. First, most of the algorithms generate unidirectional links in the network, which makes most of the existing routing algorithms inapplicable. In order to generate bidirectional links, extra steps have to be taken which brings more complexity to the algorithms. Second, the dependencies on volatile information, such as location information, signal strength or angular positions, contribute to the instability of the topology control algorithms based on power control ([56]). Third, it is hard to tell whether these algorithms really benefit in terms of performance. Some researchers found that using power control algorithms does not help the network throughput ([57]).

C. System Assumptions and Definitions

The system model is the same as Chapter II. A mobile ad hoc network is modeled as a unit disk graph, and each node learns its neighbors by broadcasting HELLO messages. We assume that each message is reliable delivered in the network, and there is an upper bound in the message delay between two neighboring nodes, denoted as T. Each node estimates the density λ locally by counting the number of different nodes from which it receives HELLO messages. Also, each node is able to measure its initial and remaining energy, denoted by EI_i and ER_i respectively.

D. The Algorithm

Our algorithm has four goals.

- First, it ensures that the algorithm quickly finds a connected dominating set with relatively small size. Many existing algorithms ([51, 39, 8]) find a smaller CDS, but the time and messages required to find the CDS are much larger. Quickly finding a connected dominating set with relatively small size may be more applicable in a highly mobile ad hoc network.
- Second, it uses few messages to find and maintain the CDS.
- Third, the algorithm adapts to the mobility efficiently both in terms of message and time. A single movement of a CDS node should not impact the whole network.
- Fourth, it periodically rotates the CDS nodes to allow each node to be a CDS node every so often. This will distribute the energy consumption of the CDS to all nodes in the network, achieving load balancing and energy efficiency.

The construction of the CDS has two steps. First, we find a maximal independent set, then we select some nodes in the network to connect the nodes in the independent set.

1. Finding a Maximal Independent Set

To find a maximal independent set, we can directly apply the algorithms in [51, 39, 8, 41, 40]. The differences between these algorithms are mainly in the criteria for a node to be in the independent set according to their id, degree, etc.

Our algorithm uses the remaining energy level and id to determine the maximal independent set. A node that has the maximum value of (E_r, id) among all its neighbors broadcasts a *Dominator* message. All nodes receiving the *Dominator* message broadcast a *Dominatee* message. A node selects one of the neighboring nodes to be its dominator, and broadcasts a *Dominatee* message. The *Dominatee* message includes the id of the node and id of its dominator. After this step, all *dominator* nodes form an independent set, and all other nodes in the network have at least one neighboring node as a *dominator*. Figure 16 shows the formation of a maximal independent set in a network. The tuple of each node i is its remaining energy level $\frac{ER_i}{EI_i}$ and its id i. In the network, nodes 10, 9 and 7 are selected for the maximal independent set because they have the largest ER among their neighbors.

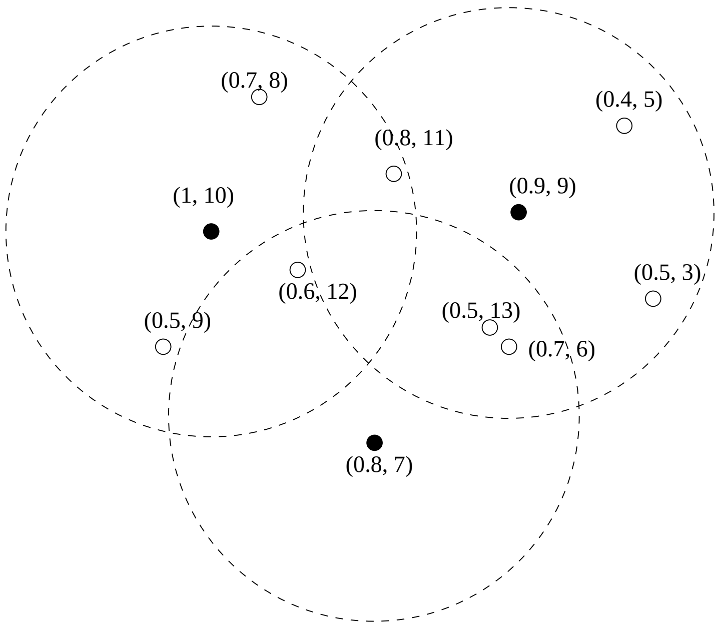


Fig. 16. Step 1: Finding a Maximal Independent Set

Lemma 3 *After step 1, for each dominator u, there exists a dominator node v such that the number of hops between u and v is 2 or 3.*

Proof. We prove by contradiction. Pick a dominator node u in the network after step 1. Suppose there is no dominator node within three hops of u. Find a path from u to a dominatee node v which is three hops to u, denoted as (u, x, y, v), where x and

y are intermediate nodes in the path. Node y is a dominatee and has no dominator neighbor. According to our algorithm, after step 1, each node should have at least one neighboring node in the maximal independent set. Therefore, this is a contradiction.

■

2. Finding Connectors

After generating the independent set, the next step is to find a set of nodes called “connectors”. These “connectors” will connect the independent set nodes, and the two sets will join together as the connected dominating set.

Ideally, the number of connectors should be kept as small as possible, and the messages sent by each node should be as few as possible. In [51], in order to select such “connector” nodes, each dominatee node sends a request message with its id and the ids of the dominator nodes it connects. If the node has the smallest id among all dominatee nodes which connect the same pair of dominators, it will be a connector node. We argue that if the nodes are dense enough in the network, a randomly picked node can act as a connector. Such observation is also used in the design of cluster algorithms as [58] and [54].

In [53], each node waits a randomized delay before announcing itself as a backbone node. If a node learns that one of its neighbors has already announced itself as a connector node and it does not connect any additional pair of nodes, the node will not change to a connector. The purpose of the random delay is to avoid the contention in the announcement of connector and keep the number of connectors small. The randomization is based on the remaining energy level and the number of pairs of nodes that it can connect. However in order to calculate the random delay in [53], each node should know whether all pairs of its neighbors can communicate directly or through another connector.

The random delay used in our algorithm is similar to [53]. However, we do not require that the node know all pairwise connections of the neighbors. Instead, we use the geometric property of the unit disk graph to determine the random delay. The only information that a node needs is the ids of the dominator neighbors and the dominators of its neighbors, which can be acquired by listening to the HELLO messages. The procedure for finding connectors for a dominatee node works as follows.

1. If a dominatee node i has more than one dominator neighbors or has a neighboring dominatee whose dominator is different than itself, it waits a random delay equal to the function $Delay(i)$ (discussed later in this section).

2. After waiting the random delay without receiving any *Connector* message during the delay, it broadcasts a *Connector* message, which includes its id, its dominator neighbor list, and its neighbor's dominators.

3. If a dominatee node is waiting for its random delay to end and receives a *Connector* message from its neighboring node, if it learns that that node will connect the same set of dominators as itself, it cancels the random delay and does not announce itself as a connector.

4. If two *Connector* messages collide, the nodes recalculates the delay and repeats the above procedure.

The randomized delay function $Delay(i)$ for a dominatee i is calculated as follows.

$$Delay(i) = ((1 - \frac{ER_i}{EI_i}) + P) \times E(c_i) \times T$$

P is a random variable uniformly distributed between 0 and 1, T is the upper bound on the message delay between two neighbors, ER_i and EI_i are the remaining and total energy in i respectively. If a node has more remaining energy, it is more likely to be a connector. $E(c_i)$ is the expected number of nodes that can connect the same pair of dominators as i does. Now, we calculate the value of $E(c_i)$.

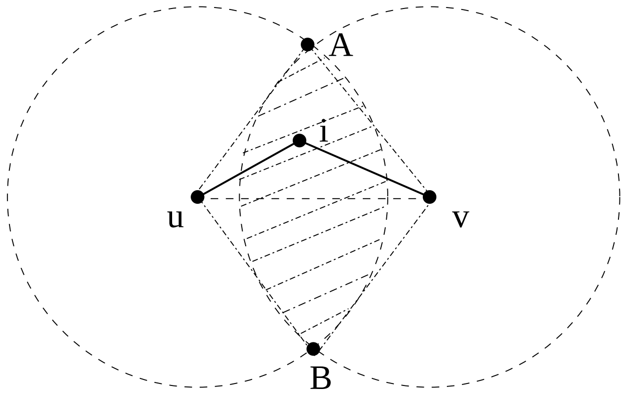


Fig. 17. An Illustration of the Connector Selection: Case 1

Case 1: Node i has two dominator neighbors, u and v, which are two hops away from each other, as shown in Figure 17. First notice the minimum distance between u and v is greater than 1, otherwise they could communicate with each other directly. So the maximum area in which connectors could reside is the intersection lune of the disks centered at u and v, which is hatched in Figure 17. Define the Euclidean distance between two nodes u and v as $dist(u, v)$. Since $dist(u, v)$ is at least 1 and $dist(u, A)$, $dist(u, B)$, $dist(v, A)$ and $dist(v, B)$ are all 1, it is easy to verify that the angles of $\angle Auv$, $\angle Avu$, $\angle Buv$ and $\angle Bvu$ are at most $\frac{\pi}{3}$. Let $S_{\widehat{AuB}}$ be the area surrounded by the line uA, the line uB and the arc AB. Let $S_{\triangle}AuB$ be the area of the triangle AuB. Then the maximum area of the lune (the hatched region) is $2 \times (S_{\widehat{AuB}} - S_{\triangle AuB}) = \frac{2\pi}{3} - \frac{\sqrt{3}}{2}$.

Case 2: A dominatee node i has one dominator neighbor u, but it has another dominatee neighbor j whose dominator is w, so u and w are three hops away from each other, as illustrated in Figure 18. Obviously, i and j can not be in the intersection

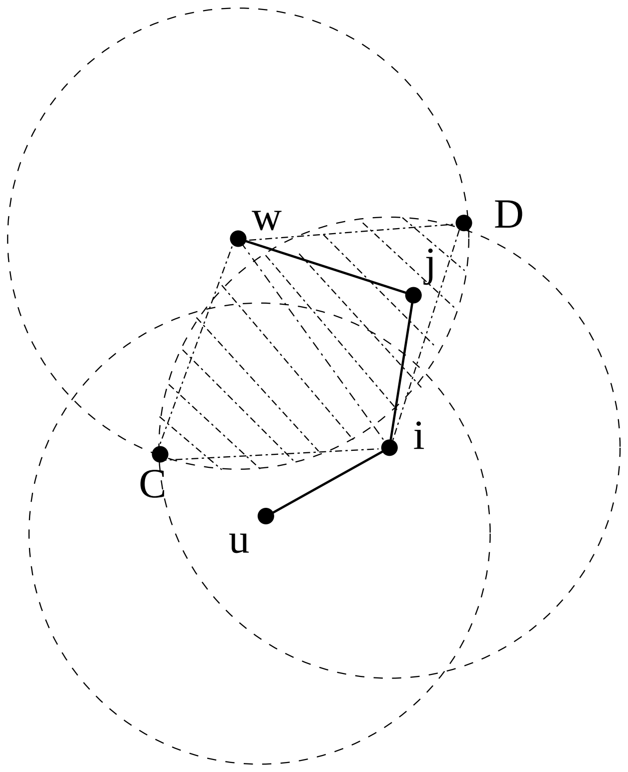


Fig. 18. An Illustration of the Connector Selection: Case 2

lune of the disks centered at u and w. And $dist(w, i)$ and $dist(u, j)$ are both at least 1. The maximum area that could have a node connecting i and w is the intersection lune of the disks centered at w and i, as hatched in Figure 18. Since $dist(w, i)$ is at least 1, and $dist(w, C)$ and $dist(w, D)$ are both 1, then the maximum area of the intersection lune (the hatched region) is $2 \times (S_{\widehat{AuB}} - S_{\triangle AuB}) = \frac{2\pi}{3} - \frac{\sqrt{3}}{2}$, which is the same as Case 1.

Since each node knows the density of the network is λ, the random delay function for node i is

$$Delay(i) = ((1 - \frac{E_{r_i}}{E_{m_i}}) + P) \times (\frac{2\pi}{3} - \frac{\sqrt{3}}{2}) \times \lambda \times T$$

Figure 19 illustrates the process of step 2. Node 11 connects 9 and 10, node 12 connects 7 and 10. There are two eligible connectors for 7 and 9, which are nodes 6 and 13. By applying the random delay function, node 6 has a better chance to be the connector because it has more remaining energy. After step 2, nodes 6, 11 and 12 are added into the CDS as connectors, which are colored in gray.

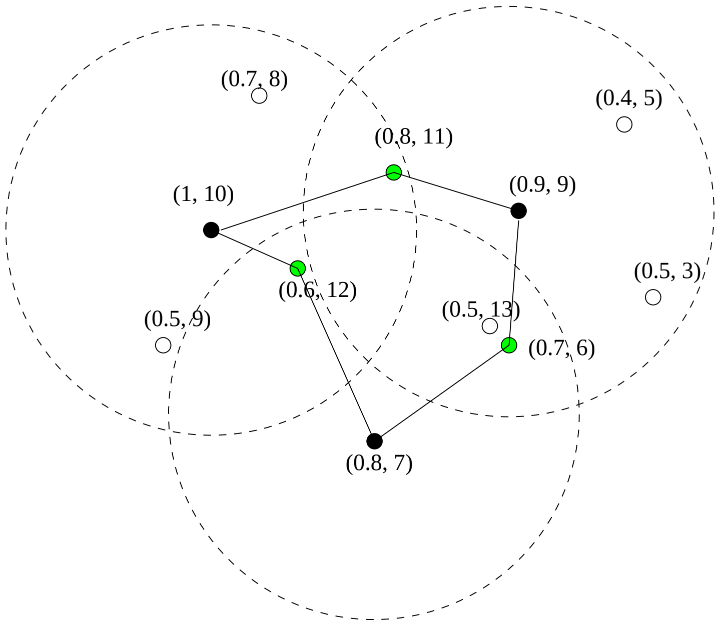


Fig. 19. Step 2: Finding Connectors

After finding the connectors, the maximal independent set and connectors form a connected dominating set. Notice that for each pair of dominators, our algorithm may find more than one pair of connectors. However, as we show next, the number of connectors for each pair is a constant.

Theorem 7 *The size of the randomized connected dominating set is within a constant multiple of the optimal.*

Proof. According to our algorithm, if there are multiple nodes that can connect the same pair of dominators u and v, the node which has the least random delay will announce its legitimacy first, making all its neighboring nodes abort their attempt to be a connector. But there may be other connector nodes which are not in its neighborhood. So the problem is reduced to finding a maximal independent set of the set of nodes that are adjacent to both u and v.

If the two dominator nodes are two hops away from each other, there are at most two connectors which are independent. To see why, consider the lune intersected by the disk of u and v in Figure 17. Since $\angle Auv$ and $\angle Buv$ are both at most $\frac{\pi}{3}$, $\angle AuB$ is at most $\frac{2\pi}{3}$. $dist(u, A)$ and $dist(u, B)$ are 1. The maximum distance between any two nodes in the lune is $dist(A, B)$, which is

$$\sqrt{dist(u,A)^2 + dist(u,B)^2 - 2 \times dist(u,A) \times dist(u,B) \times \cos(\angle AuB)} = \sqrt{3}.$$

So there are at most two connectors.

If the two dominator nodes are three hops away from each other, according to [51], the size of the maximal independent set is a constant, but the exact value of the constant is still to be discovered.

Therefore, for each pair of nodes in the maximal independent set which are two or three hops away, there are a constant number of connectors in between. So, the total number of connectors is within a constant multiple of the size of the maximal independent set at the first step. We know that the size of a minimum connected dominating set is within a constant factor of the size of any maximal independent set ([43]), so the size of the CDS by our algorithm is within a constant of the optimal.

■

Theorem 8 *The message complexity of our algorithm to find a connected dominating set is* $O(n \log n)$.

Proof. It is easy to see that in the first step, each node sends one message indicating whether it is a dominator or a dominatee. At the second step, only the connector nodes will send a message. So the total number of messages sent in our algorithm is $O(n)$. Because each message contains the sender's id and ids of the sender's dominator neighbors up to two hops, which is constant according to the result [43], the size of each message is within $O(\log n)$. So the total message complexity is $O(n \log n)$. ■

Although the message complexity of our algorithm is asymptotically the same as many other algorithms, our simulation results in Section F indicate that in practice the message complexity of our algorithm could be much better than that of existing algorithms. In fact, many of the dominatee nodes only send one dominatee message. And many connector nodes only send one dominatee message at the first step, because their attempt to be a connector is canceled when overhearing connector messages with the same purpose from neighboring nodes. Therefore, the total number of messages to find a CDS for our algorithm is much smaller than for other algorithms, especially when the network size is big, as shown in the simulation results in Section F.

E. CDS Recomputation

As CDS nodes usually consume energy faster than the other nodes, having a fixed CDS will make the battery of those nodes deplete very quickly, and as a result, partition the network. Therefore, the CDS needs to be recomputed in order to let each node take the responsibility of being a CDS node to balance the energy load. A dominator node, after serving T time, can quit the CDS by sending out a quit message, which triggers its neighboring nodes to recompute a new dominator and its connectors to update their local data structure. T is a system parameter that can be tuned according to various conditions, including the node's battery level, and the network density. A key observation here is CDS nodes should be recomputed asynchronously in order to avoid abrupt backbone change. The dominator nodes will jitter themselves in quitting the CDS, either by setting a randomized jitter value, or using a hash function to assign each node a specific time slot, which is the approach in [56]. After its dominator quits the CDS, all dominatees become eligible dominator candidates, and they use the same procedure in Section D to reselect a new dominator and connectors. The original connectors will update their local data structure and become dominatee if they do not connect any pair of dominators. Here we assume each node can get the update information of neighboring nodes by receiving HELLO messages, and the HELLO messages are accurate enough to reflect the latest state of the neighboring nodes.

Another factor triggering the CDS recomputation is the mobility. Mobility can cause the formed connected dominating set to become invalid, redundant, or even partitioned. Recomputation of the CDS should keep the impact of the mobility as small as possible. The change of the CDS due to the movement of a single node should be as small as possible. The idea is to recompute the dominator nodes first, and then find the connectors.

- Dominatee movement. A dominatee's movement does not impact the connectivity of the CDS. If it moves to the neighborhood of another dominator, it remains in the dominatee state and sets its dominator to the new dominator neighbor. If it moves to a place with no dominator neighbor, it enters an undecided state and uses our randomized CDS algorithm in Section D to decide its next state. If its original dominator node finds it no longer has any dominatees, it changes to an undecided state and recompute its next state.

- Dominator movement. A dominator's movement will change the states of its dominatees and connectors. If a dominatee node learns that its dominator node has moved away, it changes its state to undecided and recomputes a new dominator. A connector will remove the related entry in its local data structure and if it connects other pairs of dominators, it will not change its state, otherwise it will change its state to undecided and recompute a new dominator.

- If a connector moves out of the range of the dominators it connects, it changes its state to undecided and decides its new state according to our algorithm in Section D. All eligible connectors use the random delay function to decide a new connector.

F. Simulation

In this section, we discuss the simulation results of our randomized CDS algorithm. As we mentioned earlier, the main focus of our simulation is the performance of our CDS algorithm in terms of CDS size, number of messages sent to find the CDS, and energy efficiency. A lot of work in the literature studied the routing performance using CDS [47, 32, 49]. We will discuss a routing algorithm and its performance in Section G. In this work, we skipped the simulation of the routing performance with CDS.

We developed an event driven simulator with a graphical user interface that can show the geometric display of the CDS. Our simulation is set up in a 1000m by 1000m grid. Nodes are uniform randomly distributed in the simulation grid initially. Each node has a transmission radius of 250m, and each node follows the random way point mobility model ([45]). In this model, a node uniform picks a destination at random inside the grid and moves to that location at a uniformly random speed between 0 and 10m/s. After reaching the destination, the node picks another random destination and moves again. The number of nodes in the network scales from 50 to 400. To measure the energy consumption of our randomized connected dominating set, we set a different energy consumption rate for different types of nodes. A dominator consumes 0.05% of the total energy per time unit, a connector consumes 0.02%, a dominatee node consumes 0.01%. This is comparable to existing works, e.g. [53, 56].

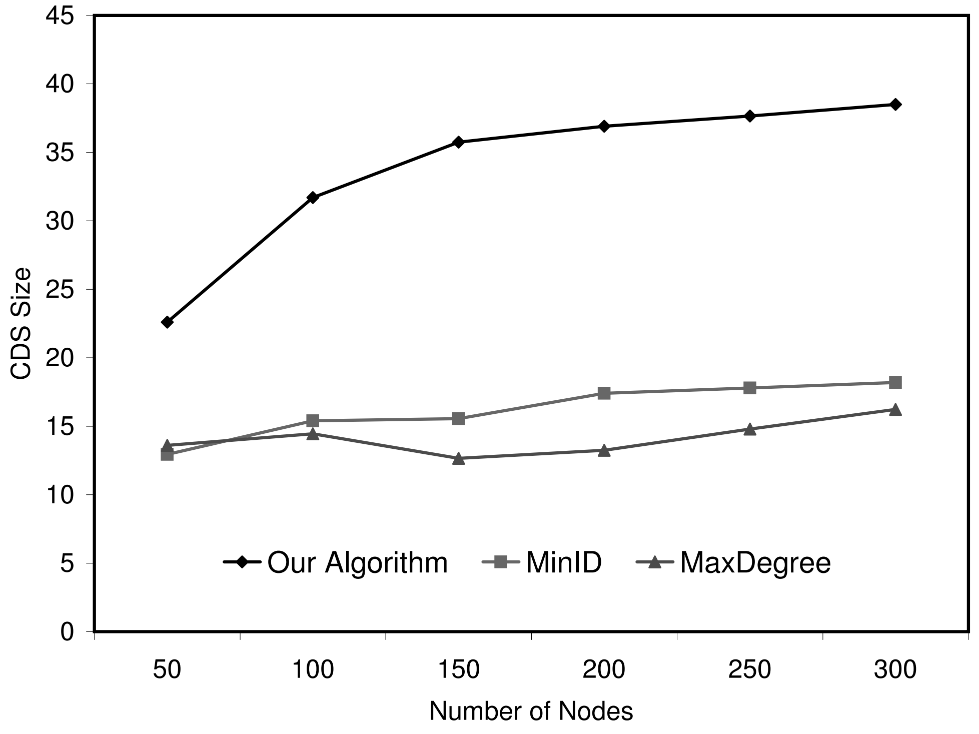


Fig. 20. CDS Size vs. Number of Nodes

In order to evaluate the performance, we compare our algorithm with two CDS algorithms in [8], which claims to have the best approximation and performance in the literature. Both algorithms in [8] assume a spanning tree with a root node in the network, and compute the CDS based on each node's id or degree. We denote these two algorithms as MinID and MaxDegree. The simulation data plotted in our figures

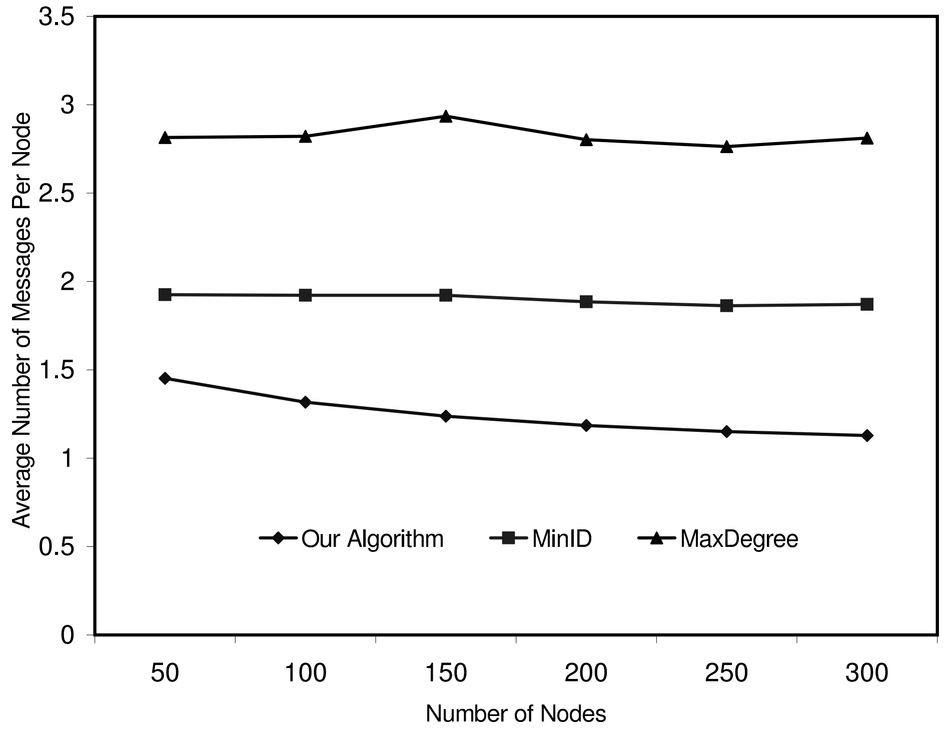


Fig. 21. Average Message vs. Number of Nodes

is the average of 20 executions, each of which runs for 10,000 seconds. To measure the CDS size, we take the snapshot of the network by having each node report its status (dominator, connector or dominatee) 10 seconds after each CDS recomputation. The CDS recomputation interval is set to 50 seconds in the simulation. The CDS size is calculated by counting the number of dominator nodes. The message is the average number of messages taken over in each CDS recomputation.

Figure 20 shows the simulation results of the CDS sizes. Not surprisingly, the CDS size of our algorithm is around twice the CDS size of MinID and MaxDegree in most cases. The reason is that our algorithm finds more than one connector for the

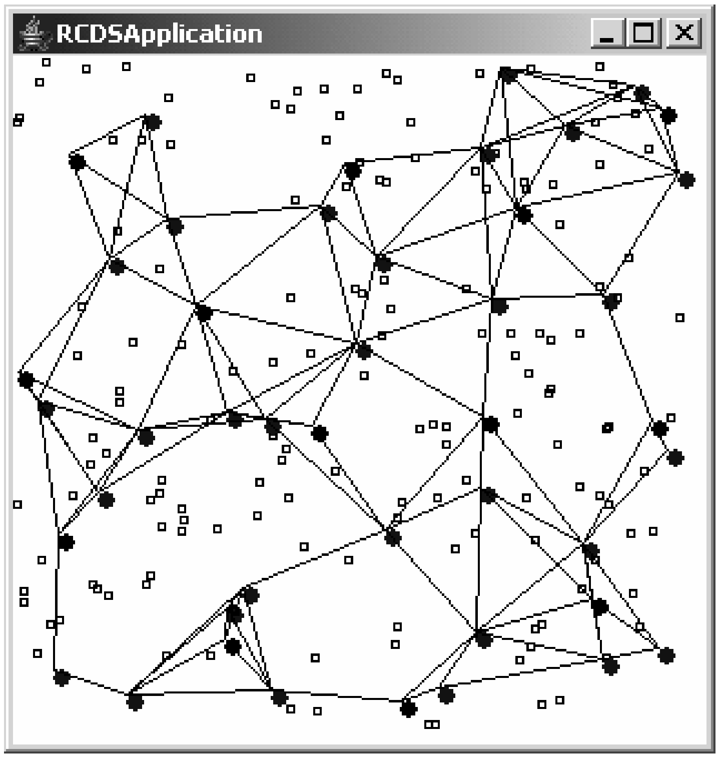


Fig. 22. A Snapshot of the Network

same pair of dominator nodes as discussed earlier, while both MinID and MaxDegree find the CDS along with a spanning tree, so the CDS subgraph is a tree essentially. But on the other hand, since our algorithm usually finds more than one connector, it has more disjoint paths for routing, which provides more robustness and redundancy in a mobile ad hoc network.

Figure 21 shows the number of messages per node to find a CDS (the total number of messages divided by the number of nodes). We do not count the HELLO messages because all CDS algorithms assume neighbor information is available at each node. Our algorithm uses only about half of the messages of MinID and MaxDegree to find a CDS. This is because in our algorithm, after each node sends a DOMINATOR or DOMINATEE message, only the connector nodes will eventually send another message. As the density of the network increases, the number of connectors is roughly the same, which makes the average number of messages even smaller. On the contrary, since MinID and MaxDegree find the CDS by establishing a spanning tree, each node has to be activated before the CDS selection, and each node at least sends two messages. In the MaxDegree algorithm, each node sends an extra message containing its degree after it is activated.

Figure 22 shows a snapshot of an execution of our algorithm. In the snapshot, we only draw links between dominator nodes. Notice the subgraph consisting of the CDS nodes is not a planar graph in general, but it provides a useful infrastructure for further optimization.

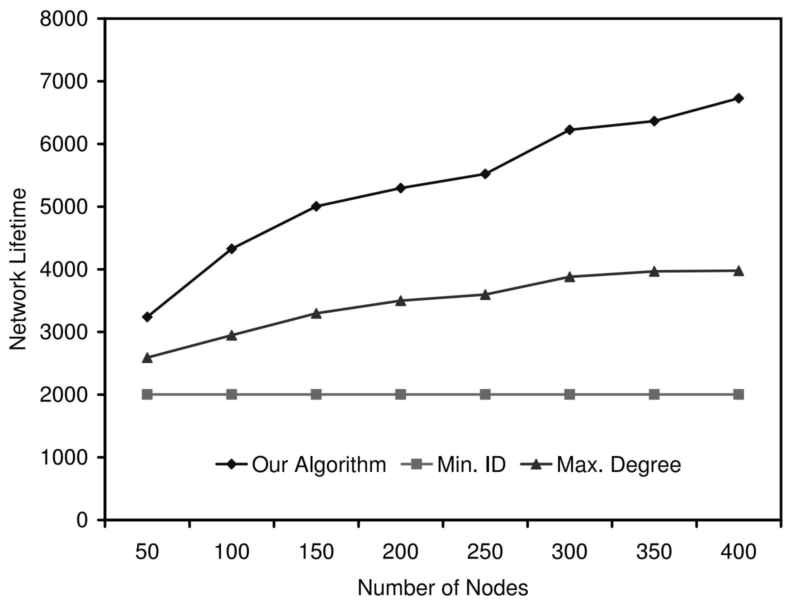


Fig. 23. Network Lifetime vs. Number of Nodes

Figure 23 shows the difference in network lifetime. The network lifetime is the time elapsed in the simulation till the first node runs out of energy. As shown in the graph, the network lifetime for our algorithm is significantly longer than for MinID and MaxDegree. Also, our algorithm extends the network lifetime as the number of nodes increases. This is because as the number of nodes in the network increases, more nodes are eligible to be a dominator or connector, which distributes the energy consumption of the CDS more evenly across the entire network, preventing a particular node from running out of energy quickly. The network lifetime for MinID does not increase with the number of nodes at all. This is because the MinID uses the id of nodes to compute the CDS. The node with the minimum id in the network is always a dominator node, and its energy is consumed at exactly the same rate in any scenario. The network lifetime for MaxDegree increases slightly as the number of nodes. MaxDegree adjusts the CDS nodes by each node's degree, which may change over time due to the mobility, but it does not capture the energy consumption nature of the CDS nodes.

Figure 24 shows the standard deviation of the remaining energy for each node when one node runs out of battery. The standard deviation indicates the differences of the remaining energy for each node in the network. A small standard deviation is desirable because it means the difference of each node's energy consumption is small. As shown in Figure 24, the standard deviation of the remaining energy for our algorithm is significantly less than MinID and MaxDegree, because our algorithm rotates the CDS nodes periodically according to nodes' remaining energy, therefore balancing the energy consumption of the CDS among all network nodes. MaxDegree has slightly less deviation than MinID because more nodes get a chance to be dominators due to the mobility.

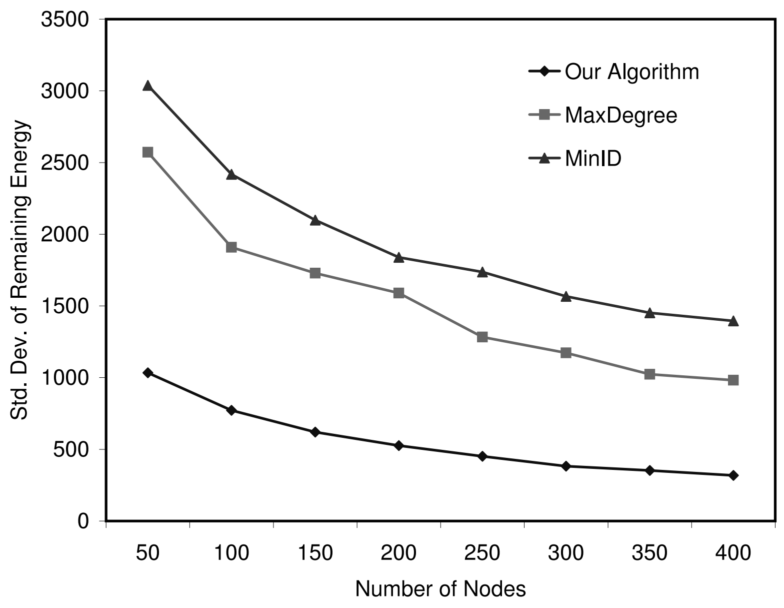


Fig. 24. Standard Deviation of Remaining Energy vs. Number of Nodes

G. A Routing Algorithm Based on Randomized CDS

In this section, we describe a routing algorithm based on our randomized connected dominating set. The number of hops found by this routing algorithm is within a constant factor of that by using the same routing algorithm on the whole network. Most of the routing work is done on the backbone of the connected dominating set. To find a path between a source s and a destination t, the routing algorithm works as follows.

- If s and t are immediate neighbors, the path is (s, t).
- If s and t are not immediate neighbors, s sends a routing request to its dominator node u.

- After receiving a request from its dominatee, the dominator node u finds a path to t's dominator node v using any existing routing protocol for mobile ad hoc networks such as DSR([59]) or AODV ([60]).
- Messages are forwarded along the path $(s, P(u, v), t)$, where $P(u, v)$ is the path found in the previous step from u to v.

Theorem 9 *The number of hops in the path found by the routing algorithm applied on our RCDS is within 3 times the number of hops in the path found by using the same routing algorithm on the whole network.*

Proof. Suppose there is a path between the source s and t in a routing algorithm, denoted as $p = (u_1, u_2, ..., u_n)$, where $u_1 = s$ and $u_n = t$. We denote the alternative path as $p' = (s, v_1, ..., v_2, .., v_3, ..., v_n, t)$ for routing with CDS, where v_1 is s's dominator, and v_n is u_n's dominator. Now we look at any hop (u_i, u_{i+1}) in the path p.

- Case 1: Both u_i and u_{i+1} are CDS nodes. Then (u_i, u_{i+1}) is equal to (v_i, v_{i+1}) because u_i=v_i and u_{i+1}=v_{i+1}.
- Case 2: Either u_i or u_{i+1} is a CDS node, but the other is not. Without loss of generality, suppose u_i is a CDS node. Since u_{i+1} is within one hop of u_i, u_{i+1}'s dominator, denoted as v_{i+1}, must be within the two-hop neighborhood of u_i. According to our algorithm, there must be a connector connecting u_i and v_{i+1}, denoted as c_i. Therefore, for (u_i, u_{i+1}) in path p, we can find a path (u_i, c_i, v_{i+1}), which is twice as long as (u_i, u_{i+1}) in the number of hops.
- Case 3: Neither u_i or u_{i+1} is a CDS node. Then u_i and u_{i+1}'s dominator v_i and v_{i+1} are at most three hops away. There must be two connectors c_i and c_{i+1}

that connect v_i and v_{i+1}. So for (u_i, u_{i+1}), we can find a path $(u_i, c_i, c_{i+1}, v_{i+1})$ in p', which is three times as long as (u_i, u_{i+1}) in the number of hops

Therefore, for each hop (u_i, u_{i+1}) in the path $(u_1, u_2, ..., u_n)$, the number of hops in the path $(v_i, ..., v_{i+1})$ is at most 3. So the total number of hops in the path $(s, v_1, ..., v_2, .., v_3, ..., v_n, t)$ is at most 3 times the number of hops in the path $(u_1, u_2, ..., u_n)$. ■

Theorem 9 shows that a routing algorithm using a CDS as a backbone can always find a routing path whose length is within a constant of that of the path found by using the same algorithm on the whole network. However, since only backbone nodes keep the routing table and send routing requests, and the CDS is only small portion of the network nodes given enough density of the network, the overall messages and space usage is reduced. [47] gives a more detailed simulation results for backbone based routing. In this chapter, we are more focused on the CDS construction, therefore the simulation of the routing performance is omitted.

CHAPTER IV

A TOKEN FORWARDING K-MUTUAL EXCLUSION ALGORITHM FOR AD HOC NETWORKS

A. Introduction

Providing shared access to resources through mutual exclusion is a fundamental problem in computer science, and is therefore worth considering for any type of dynamic network In particular, since wireless mobile nodes are power constrained and are often deployed in emergency situations, the processors may have the need to restrict concurrent access to a limited set of services, e.g., wireless channels [61] or remote sensors. The k-mutual exclusion problem is an extension of the mutual exclusion problem, which is a fundamental concept in concurrent and shared resource allocation and access in distributed systems. The k-mutual exclusion problem involves a group of n processes, each of which intermittently requires access to an identical resource or piece of code called the *critical section* (CS). At most k, $1 \leq k \leq n$, processes may be in the CS at any given time. A typical application of k-mutual exclusion problem in a mobile ad hoc network is data fusion when there are k data centers in the network and each wireless node updates the data centers exclusively such that there is at most one update for each data center at any time.

In this chapter, we propose a k-mutual exclusion algorithm called K-Reversal Link with token Forwarding (KRLF). KRLF extends the K-Reversal Link algorithm (KRL) proposed by Walter [4]. KRL induces a logical directed acyclic graph (DAG) on the network, dynamically modifying the logical structure to correspond to the actual physical topology in the ad hoc environment. The algorithm ensures that all requesting processors eventually gain access to the CS once the network topology changes cease. This algorithm is intended to run on top of a lower level protocol that provides reliable message delivery on an ad hoc network (e.g., TCP[62, 63]). However, we show that the KRL algorithm may cause tokens not to be accessed when there are nodes requesting the token. KRLF adds a new token forwarding technique to the original KRL algorithm. If a token is idle, the token will be forwarded in the network according to some designated rules. When the token is forwarded to a node which is requesting a token, it is accessed by the node immediately. We prove that KRLF provides more concurrency than KRL by using the token forwarding technique. Through simulation, we show that by forwarding the token even when it is idle, the average access time of the token is reduced, which means the waiting time for each node to enter the critical section is reduced.

The next section discusses related work. In Section C, we briefly describe our system models and assumptions. Section D describes our KRLF algorithm. Simulation results are presented in Section F.

B. Related Work

Distributed k-mutual exclusion algorithms are studied in both static and dynamic networks. In *permission based* algorithms (e.g., [64, 65]), a processor requesting access to the CS must ask for and be granted explicit permission from all or some subset of the processors in the system. In *token based* algorithms (e.g., [66, 67, 68]), the possession of a unique token or tokens allows access to the CS. Existing work ([69]) shows that token based algorithms outperform permission based algorithms in terms of waiting time to get in the critical section and message complexity, because token based algorithms usually require less direct inter-processor communication. However, each of the existing distributed token based algorithms ([66, 67, 68]) for static, wired networks assume that the network is reliable and logically fully connected (i.e., the dynamic nature of the network is hidden by the routing protocol, making the network appear fully connected to the application), allowing any processor to directly communicate with any other. These assumptions make them poorly suited to the ad hoc environment, where links form and fail as a consequence of mobility.

Related work on development of mutual exclusion algorithms for ad hoc networks includes recent work by Chen and Welch [70], Baldoni et al. [61] and Jiang [71]. Chen and Welch [70] present a self-stabilizing mutual exclusion algorithm that circulates the token on a dynamic ring topology. Baldoni et al. [61] present another mutual exclusion algorithm for dynamic logical rings. Both these algorithms are distributed token-based mutual exclusion algorithms, but differ from ours by the fact that they maintain a logical ring on the network. Another difference from the work in this paper is that the algorithm in [61] requires messages to be sent to a round coordinator which uses an underlying routing protocol to determine the next node to receive the token. Jiang [71] presents an algorithm derived from the algorithm of Walter et al. [69] for the h-out-of-k-mutual exclusion problem on ad hoc networks. In this paper, only one token is used to allocate up to k resources, whereas our algorithm uses k separate tokens, providing greater system fault tolerance.

The token based 1-mutual exclusion algorithm of [69] uses the partial reversal technique from [72] to maintain a token oriented DAG with a dynamic destination. Like the algorithms of [73, 74, 75], each processor in this algorithm maintains a request queue containing the identifiers of neighboring processors from which it has received requests for the token.

The KRL algorithm maintains k tokens in the system as in [66, 68]. When $k = 1$, the algorithm ensures that the current token holder is a "sink" toward which all requests are sent (the exact technique by which this is accomplished is explained later). When $k > 1$, there may be multiple sinks in the system. However, our algorithm ensures that all non-token holding processors will always have a directed path to some token holding processor and that all token holders have an incoming directed path. In the KRL algorithm, each node dynamically chooses a neighboring node as its preferred link to a token holder (cf. [69]). Nodes sense link changes to immediate neighbors and reroute requests based on the status of the previous preferred link to the token holder and the current contents of the local request queue. All requests reaching a token holder are treated symmetrically, so that requests are continually serviced while the DAG is being re-oriented and blocked requests are being rerouted. It is possible for processors to receive requests while they are in the CS. If this happens, the processors may satisfy these requests immediately if they hold multiple tokens, increasing concurrent access to the CS.

The k-mutual exclusion problem shares some similar properties such as mutual exclusion, no starvation and concurrency with the dining and drinking philosophers problems ([76]), which are well known resource allocation problems in distributed systems. However, in the dining and drinking philosophers problems, a processor requests exclusive access to a specific set of resources to enter the critical section. In the k-mutual exclusion problem, there is only one resource, but up to k processors may access the resource concurrently.

C. System Model and Assumptions

The system contains a set of n independent mobile nodes, communicating by message passing over a wireless network. We make the following assumptions about the mobile nodes and the network.

1. Nodes have unique node identifiers in the range $0 \ldots n-1$.

2. Each node is aware of its *neighbors*, i.e., the set of nodes with which it can currently directly communicate. A link layer protocol provides indications of link formations and failures. Also, nodes can only communicate with their current neighbors.

3. Communication links are bidirectional and FIFO.

4. Incipient link failures and message collisions are detectable at the link layer, providing reliable neighbor-to-neighbor communication,

5. Nodes do not fail.

Note that assumptions 4 and 5 ensure that no token will be lost during execution. Our algorithm is robust to network partitions, in the sense that any connected component that contains k' tokens will solve k'-mutual exclusion.

Each mobile node runs an application process and a k-mutual exclusion process that communicate with each other to ensure that the node cycles between its REMAINDER section (not interested in the CS), its WAITING section (waiting for access to the CS), and its CRITICAL section. The k-mutual exclusion runs on top of the network layer. Communication between different processes are modeled as events in the system.

The k-mutual exclusion process is modeled as a state machine (see Fig. 25), with the usual set of states, some of which are initial states, and a transition function.

A step of the k-mutual exclusion process at node i is triggered by the occurrence of an *input event*. The effect of a *step* is to apply the process' transition function, taking as input the current state of the process and the input event, and producing as output a (possibly empty) set of *output events* and a new state for the process. Referring to Fig. 25, the *application I/O events* at the k-mutual exclusion process

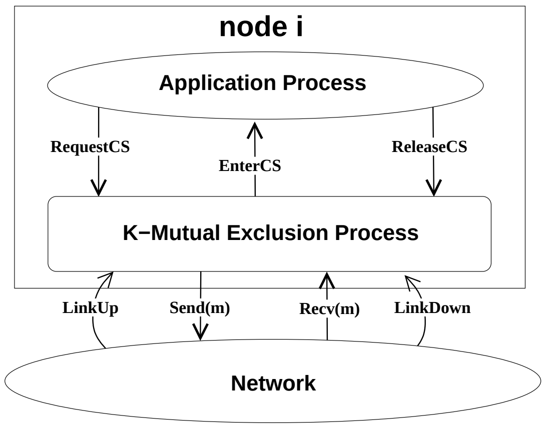


Fig. 25. The k-Mutual Exclusion System Architecture

are:

1. RequestCS$_i$: (input) request for access to CS.

2. ReleaseCS$_i$: (input) release of CS.

3. EnterCS$_i$: (output) permission to enter CS.

The *network I/O events* at the k-mutual exclusion process are:

1. Send$_i(j, m)$: (output) node i sends message m to j.

2. Recv$_i(j, m)$: (input) message m from node j received at i.

3. LinkUp$_i(l)$: (input) link l incident on i has formed.

4. LinkDown$_i(l)$: (input) link l incident on i has failed.

An *execution* is a sequence of the form $C_0, in_1, out_1, C_1, in_2, out_2, C_2, \ldots$, where the C_i's are configurations, the in_i's are input events, and the out_i's are sets of output events. The execution must satisfy the assumptions in Section C. An execution must end in a configuration if it is finite. A positive real number is associated with each in_i, representing the time at which that event occurs.

We require that any k-mutual exclusion algorithm satisfies the following properties:

1. *k-mutual exclusion:* At any time during the execution of the algorithm, at most k processes can be in the CS.

2. *no starvation with concurrency*: Consider any infinite execution C_0, in_1, out_1, C_1, in_2, out_2, C_2, ... with a finite number of link changes. For every $m \leq 1$, if $in_m = RequestCS_i$, and if at most $k-1$ processors stay in the critical section forever, then there exists $l > m$ such that out_l contains $EnterCS_i$.

As we show later in the chapter, KRL does not satisfy the no starvation with concurrency property. In KRL, a token currently not being used by any node may not be accessed by a requesting node. KRLF adds a token forwarding technique to allow the token to be forwarded in the network and eventually the token reaches the requesting processor. Thus KRLF does satisfy the no starvation with concurrency property.

D. KRLF Algorithm

In this section, we introduce our KRLF algorithm. As we mentioned earlier, KRLF is based on the KRL algorithm ([69]). First, we give a general overview of the operation of the KRL algorithm. Then we discusses a scenario which KRL can cause a free token not to be accessed. We then introduce our KRLF algorithm and discuss how the token forwarding technique of KRLF solves the problem. Then we show that KRLF satisfies the no starvation with concurrency property of the k-mutual exclusion problem.

1. Overview of KRL

In KRL, each processor maintains a number of local data structures as part of the k-mutual exclusion process. Details are given in [4]. Here we only list some key data structures for the overview of the KRL algorithm.

- *status*: Indicates whether node is in the WAITING, CRITICAL, or REMAINDER section. Initially, *status* = REMAINDER.
- N: The set of all nodes in direct wireless contact with node i. Initially, N contains all of node i's neighbors.
- *myHeight*: A three-tuple $(h1, h2, i)$ representing the height of node i. Links are considered to be directed from nodes with higher height toward nodes with lower height, based on lexicographic ordering. E.g., if $myHeight_1$ = (2, 3, 1) and $myHeight_2$ = (2, 2, 2), then $myHeight_1 > myHeight_2$ and the link between these nodes would be directed from node 1 to node 2. Initially at node 0, $myHeight_0$ = (0, 0, 0) and, for all $i \neq 0$, $myHeight_i$ is initialized so that the directed links form a DAG in which every node has a directed path to some token holder and in which every token holder has at least one higher neighbor.
- *height*[j]: An array of tuples representing node i's view of $myHeight_j$ for all $j \in N_i$. Initially, *height*[j] = $myHeight_j$, for all $j \in N_i$. In node i's viewpoint, if $j \in N$, then the link between i and j is *incoming* to node i if *height*[j] > *myHeight*, and *outgoing* from node i if *height*[j] < *myHeight*.
- *tokenHolder*: Flag set to true if node holds token and set to false otherwise. Initially, *tokenHolder* = true if $0 \leq i < k$, and *tokenHolder* = false otherwise.
- *next*: Indicates the location of a token in relation to i or the node to which i

last sent a request. When node i holds the token, $next = i$, otherwise $next$ is the node on an outgoing link. Initially, $next =$ i if $0 \leq i < k$ (i.e., if i holds a token), and $next$ is an outgoing neighbor otherwise.

- Q: "Request queue", containing identifiers of requesting neighbors and i if RequestCS$_i$ was last application input event.

The key idea of KRL is to maintain a logical Directed Acyclic Graph (DAG) in the network. In KRL, A DAG is maintained on the physical wireless links of the network throughout algorithm execution as the result of a three-tuple, or triple, of integers representing the "height" of the node, as in [72]. Links are considered to be directed from nodes with higher height toward nodes with lower height, based on lexicographic ordering of the triples. A link between two nodes is *outgoing* at the higher height node and *incoming* at the lower height node.

Node i's height triple is included with every message sent by the k-mutual exclusion process on processor i, $0 \leq i < n$, where n is the number of participating processors. The three types of messages recognized by the algorithm are *Request*, *Token*, and *LinkInfo*. The purpose of each type of message should become clear in the discussion and examples below.

The algorithm maintains k tokens in the system. Initially the token holders are nodes $0 \ldots k-1$. We assume that $k < n$.

As described in the last section, the k-mutual exclusion algorithm is event-driven. When the application process on node i makes a request for the CS, i's identifier is enqueued on its own request queue (Q_i). *Request* messages received at node i from "higher" physical neighbors cause the k-mutual exclusion process at i to enqueue the identifiers of those neighbors on Q_i in the order in which the *Request*s were received. A non-token holding node i sends a *Request* message to its lowest neighbor whenever an identifier is enqueued on an empty request queue at i. When i receives a *Token* message, it dequeues the top element on its request queue and either gives permission for its application process to enter the CS (if its own identifier was just dequeued) or sends a *Token* message to its neighboring node whose identifier was just dequeued.

Each token recipient i modifies the first two integers in its height triple if necessary each time it receives a *Token* message so that its height is lower than the height of the node that sent the *Token* message. This is not necessary when the node receiving the *Token* is already lower than the sender (which may be the case if the receiver holds or previously held a token that gave it a lower height.)

Non-token holding nodes must ensure that they have at least one "lower" neighbor at all times because requests for the token are always sent on outgoing paths. If a non-token holding node finds itself with no "lower" neighbor, it uses the *partial reversal* technique of Gafni and Bertsekas [72] (found inside the *RaiseHeight()* procedure of Fig. 40) to change the first two integers in its height triple, raising its height in relation to at least one of its neighbors and creating at least one outgoing link. Each time a node raises its height, it sends *LinkInfo* messages to all its neighbors containing its new height. Request queue entries are deleted when the link to the requester fails or reverses. The reason requests are not lost as a result of these deletions is that a processor never deletes its own id from its request queue. Therefore, the request always has a chance to "repropagate" on a new route toward a token holder.

Token holders must ensure that they have at least one "higher" neighbor at all times, since *Request* messages will not reach a token holder with only outgoing links. If a token holder finds itself with no "higher" neighbors, it uses the "reverse" of the Gafni and Bertsekas partial reversal technique [72] (found inside the *LowerHeight()* procedure of Fig. 40) to change the first two integers in its height triple, lowering its height in relation to at least one of its neighbors and creating at least at least one incoming link. Each time a node lowers its height (including when it receives a *Token* message), it sends *LinkInfo* messages on all its outgoing links. Pseudocode for the KRL algorithm can be found in Figs. 35 through 40 of Appendix A.

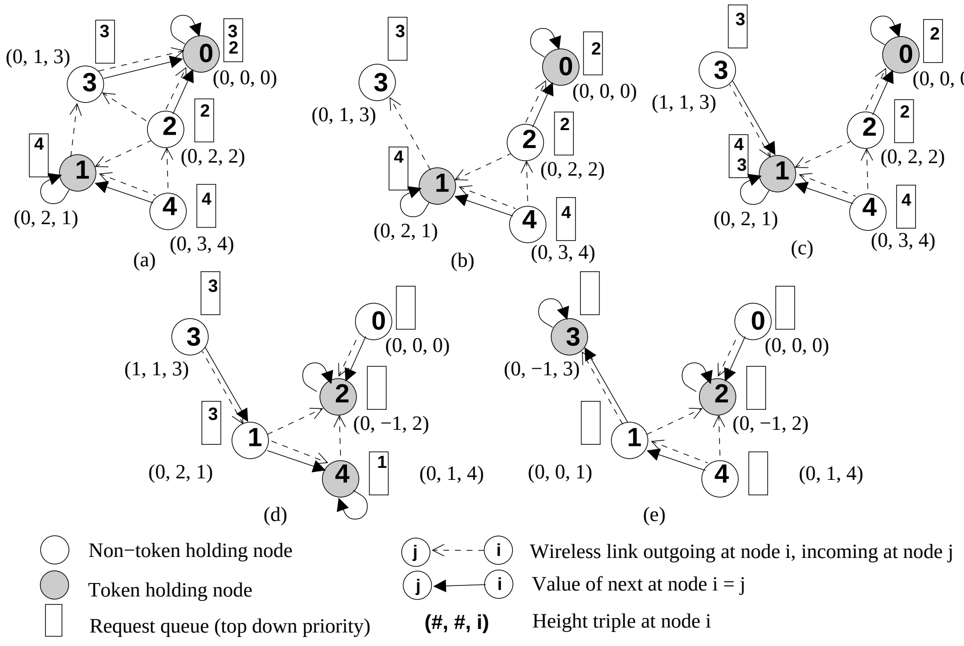


Fig. 26. Operation of KRL Algorithm on a Dynamic Network with 2 Tokens

An illustration of algorithm operation on a dynamic network with two tokens is shown in Fig. 26. In Fig. 26(a), nodes 2, 3, and 4 have requested access to the CS (note that nodes 2, 3, and 4 have enqueued themselves on Q_2, Q_3, and Q_4, respectively) and nodes 2 and 3 have sent *Request* messages to node 0, which enqueued them on Q_0 in the order in which the *Request* messages were received. Node 4 sent a *Request* to node 1, since node 1 is node 4's lowest neighbor. Fig. 26(b) depicts the system configuration after node 3 has moved in relation to the other nodes in the system, resulting in a network that is temporarily not token oriented, since node 3 has no outgoing links. Node 0 has adapted to the lost link to node 3 by removing 3 from its request queue. Node 2 takes no action as a result of the loss of its link to node

3, since the link to $next_2$ was not affected and node 2 still has one outgoing link. In part (c), node 3 has adapted to the loss of its link to node 0 by raising its height and has sent a *Request* message to node 1. Parts (d) and (e) show the system after node 0 has sent a token to node 2 and node 4 has sent a token to node 1, which then sent it to node 3.

2. KRLF Algorithm

The goal of KRLF is to increase the concurrent access of tokens in the network. Within a connected component of the network, a token is *idle* if it is at some node i that is in its REMAINDER section with $|Q_i| = 0$ but there is a non-token holding processor j in its WAITING section.

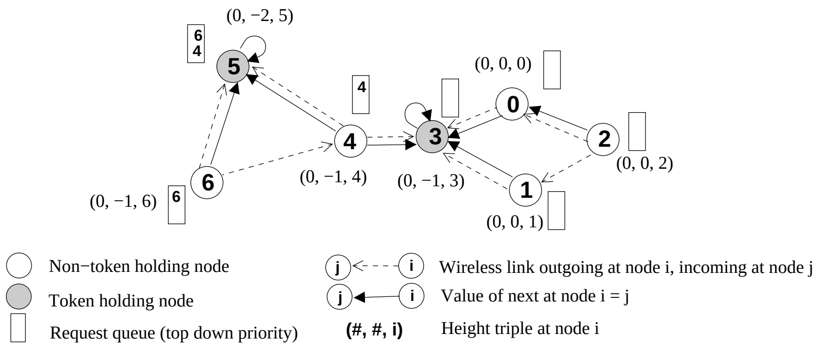


Fig. 27. Idle Token Problem in KRL Algorithm

Fig. 27 shows why tokens may frequently be idle in the KRL algorithm when demand for tokens is low. The figure gives a snapshot of KRL execution in which there are 2 token holders, nodes 3 and 5. Nodes 4 and 6 have made requests and have sent *Request* messages to node 5. The application process at node 5 is currently in the CS, so node 5 has enqueued the identifiers of node 6 and node 4 on its request queue. The application process at node 3 has already released the CS and node 3 is in its REMAINDER section with an empty request queue. Therefore, node 3 holds an idle token. Node 3 will not send its token to any other node until it receives a *Request* message. Meanwhile, nodes 6 and 4 must wait their turns for the token being used by node 5.

We try to alleviate the idle token problem by having each token holder forward the token to other parts of the network in case no processor close to it needs access to the CS. The strategy we use is to mimic the action taken by processors when forwarding a request for a token, i.e., choose the "lowest" neighboring node and send the token to that neighbor. Choosing the lowest height neighbor results in the lowest number of link reversals because the lower the height of a neighbor, the fewer outgoing links that neighbor will need to reverse when it receives the token. Nodes keep track of which of their neighbors they have forwarded tokens to or received tokens from when their request queue is empty by marking the link "visited".

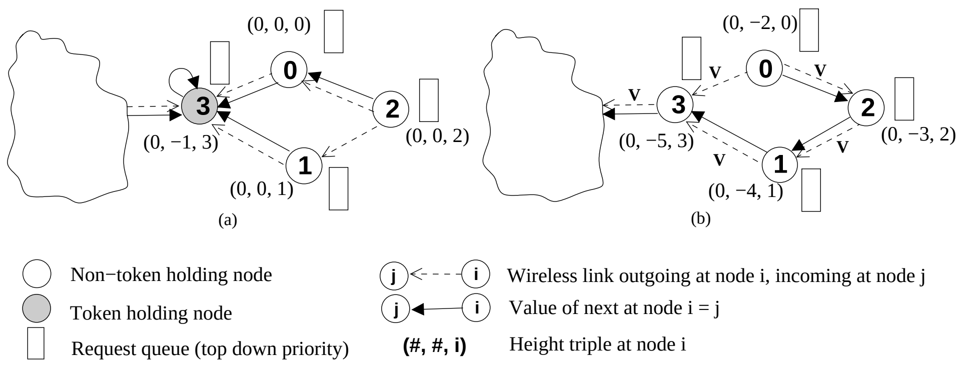


Fig. 28. Operation of KRL Algorithm with Token Forwarding. "V" on Wireless Link Indicates That Token Has Previously Been Forwarded over Link

Fig. 28 illustrates this modification during an execution of the algorithm. In Fig. 28(a), node 3 is a token holder but no neighbor of node 3 needs access to the CS. Fig. 28(b) shows a snapshot of the algorithm execution after the application process on node 3 has released the CS, and the token has been forwarded through processors 0, 2, 1, and 3 to the left portion of the network. The V on each wireless link signifies that the link has been marked "visited" by both the node forwarding and the node receiving the token. If node 3 receives the token at a later time, while it has an empty request queue, it will mark all its links as "unvisited" and start the forwarding process over.

The modifications made to the pseudocode of the KRL algorithm to implement token forwarding are shown in Fig. 29. For every node, we add the following local data structure:

- *visited[j]*: boolean array indicating whether a token has been circulated to node j. Initially set to false for all $j \in N$.

E. Correctness of KRL and KRLF Algorithms

Now we will show the correctness of the KRLF algorithm. We first show that KRLF does preserve the correctness of the KRL algorithm, which means that KRLF satisfies all the properties which KRL satisfies. Then we show that KRLF ensures the no starvation with concurrency property.

Theorem 10 *KRLF does not violate the correctness of the KRL algorithm.*

Proof. The detailed proof of the correctness of the KRL algorithm is given in [4]. Here we give a sketch of the proof. First, [4] shows that after link changes cease, eventually processors will stop raising their heights and the DAG will be token oriented. Then any sequence of propagated requests (i.e., every "request chain") beginning at any requesting processor will eventually include some token holder. Last, using a variant function argument, it is shown that a token will be delivered to every requesting node.

```
When node $i$ releases the CS:
 1. if ($|Q| > 0$) $GiveTokenToNext$()
 2. else
 3.   $PickLowest\&ForwardToken$()
 4. $status$ := REMAINDER

When $Token(h)$ received at node $i$ from node $j$:
 // $h$ denotes $j$'s height when message was sent
 1.  $visited[j]$ := true
 2.  $tokenHolder$ := true
 3.  $numTokens$++
 4.  $height[j]$ := $h$
 5.  if ($myHeight > h$)
 6.     Send $LinkInfo(h.h1, h.h2$ - 1,$i$) $\forall$ outgoing $k \in N \neq j$
 7.     $myHeight.h1$ := $h.h1$
 8.     $myHeight.h2$ := $h.h2$ - 1   // lower my height
 9.  Send $LinkInfo(h.h1, h.h2$ - 1, $i$) to $j$
10.  if ($|Q| > 0$) $GiveTokenToNext$()
11.  else
12.    $PickLowest\&ForwardToken$()

When formation of link to $j$ detected at node $i$:
 1.  Send $LinkInfo(myHeight)$ to $j$
 2.  $forming[j]$ := true
 3.  $formHeight[j]$ := $myHeight$
 4.  $visited[j]$ := false

Procedure $PickLowest\&ForwardToken$():
 1.  if ($visited[j]$ = true $\forall j \in N$)
 2.     $visited[j]$ := false $\forall j \in N$
 3.  $next$ := $l \in N$ : (($height[l] \leq height[j]$)
          and ($visited[j]$ = false) $\forall j \in N$)
 4.  $visited[next]$ := true
 5.  $numTokens$--
 6.  if ($numTokens$ = 0)
 7.     $tokenHolder$ := false
 8.  $height[next]$ := ($myHeight.h1$, $myHeight.h2-1$, $next$)
 9.  $receivedLI[next]$ := false
10.  Send $Token(myHeight)$ to $next$
```

Fig. 29. Pseudocode Modifications for Token Forwarding

As shown in the pseudocode in Fig. 29, KRLF ensures that a node that receives an idle token will lower its height relative to the node that forwarded the token. Therefore, the DAG is still token oriented in KRLF, which ensures that every request chain beginning at any requesting processor will eventually include some token holder. The rest of the proof is the same as in [4]. ■

Theorem 11 *KRLF algorithm ensures no starvation with concurrency.*

Proof. Consider an infinite execution with finite number of link changes, in which at most $k-1$ nodes are in the critical section forever, and some node m is requesting to enter the critical section. Each node stuck in its critical section has a token, so there is at least one additional token. Suppose an additional token is at node i in the network.

Now we argue that this token will be forwarded to each node in the network. Suppose in contradiction that the token is only forwarded among a part of nodes in the network. There must be a node j in the network which has some neighbors that receive the token and some neighbors that never receive the token. According to the pseudocode of KRLF, the token will be forwarded to node j's "lowest neighbor" and that neighbor will be marked as "visited". The next time node j receives a token, it will choose the "lowest unvisited" neighbor to forward the token to each time it receives a token. Eventually, the idle token will be forwarded to each unvisited neighbor of j. Therefore, the token will be forwarded to each node in the network.

Therefore, if there is a request, eventually the token will be forwarded to that requesting node and the request to enter the critical section is granted, even the other $k-1$ tokens are held by other nodes forever. ■

F. Simulation Results

In this section, we present the results of experiments designed to test the performance of our algorithms on two different simulators.

Section 1 presents the results of experiments comparing the performance of the KRL and KRLF algorithms to a "static" distributed k-mutual exclusion algorithm that is not topology sensitive, developed by Bulgannawar and Vaidya [66] (hereafter called the BV algorithm). For this comparison, we used a simulator first presented in [69] and used on the Reverse Link mutual exclusion algorithm. We simulated the BV algorithm as if it were running on top of an ideal routing protocol that always found shortest routes between any two nodes in the network. Since the KRL and KRLF algorithms need only neighbor-to-neighbor communication and require no underlying routing protocol, *all* costs (time per CS entry and number of messages sent) incurred during the simulation were recorded during executions of both the KRL and KRLF algorithms. Conversely, we did not charge the BV algorithm for the total cost used by the ideal routing protocol to find shortest path routes when links were disrupted by topology changes. Instead, we charged the BV algorithm for only the number of messages and time units required for re-routing the algorithm messages "in-transit" on disrupted shortest paths over new shortest paths. Our intention in using our "perfect world" simulator for this set of experiments was that it would reduce the effects of the routing protocol on algorithm performance. Therefore, if our algorithms performed better than the BV algorithm in this simulation, KRL and KRLF would certainly have better performance when the routing protocol could not guarantee finding shortest paths.

In Section 2, we present the results of simulating our algorithms on the *ns-2* simulator [77] with CMU extensions for mobility [78]. The intention of using the ns-2 simulator is to get a more "realistic" result since ns-2 has a well implemented MAC layer that reflects the message delay in a real mobile ad hoc network.

1. Comparison of KRL and KRLF to Static Distributed k-Mutex Algorithm

In this section, we show results produced by a discrete event simulator first developed and tested in [69]. We simulated a 30 node system with $k = 3$ tokens under varying scenarios. We chose to simulate on a 30 node system because we envision ad hoc networks to be much smaller scale than wired networks like the Internet and because typical numbers of nodes used for simulations of ad hoc networks range from 10 to 50 [61, 79, 78, 80, 81, 82, 83]. In our simulator, the network was modeled as a graph with processors for nodes and edges between nodes that currently had a wireless connection. Edges were initially created at random at the desired connectivity. Link formations and failures occurred when edges were added to or removed from the graph.

Connectivity was measured as the percentage of possible links that were present in the graph throughout the execution. In our simulation, connectivity was maintained at approximately 20% and link failures did not partition the network. This represented a fairly well-connected network in which each node had an average of 6 neighbors.

We varied the following parameters during the simulation trials:

- **Mobility**: Link changes were modeled as a Poisson process. The expected intervals for link formations and failures are 500 seconds representing low mobility and 50 seconds representing high mobility. Each change to the graph consisted of the deletion of a link chosen at random (whose loss did not disconnect the graph) and the formation of a link chosen at random.

- **System load**: The inter-arrival time of requests for the CS was modeled as a Poisson process. We varied the expected interval between requests, recording executions with intervals of 100, 80, 60, 40, 20, 10, and 1 seconds between requests.

In our experiments, each CS execution took one time unit and each message delay was one time unit. In the graphs of the results in this section, each plotted point represents the average of 20 repetitions of the simulation. The variance between executions was small and is not shown in the figures.

In each execution, we measured the average waiting time for CS entry, that is, the average number of time units that nodes spent in their WAITING sections, and the average number of messages sent per CS entry. Each algorithm started with nodes 1, 2 and 3 holding tokens. In KRL and KRLF, we initially adjusted the height of each token holder to ensure that it had at least one incoming link. A connected graph whose initial edges were chosen at random with the desired number of links was generated, node heights and link directions were initialized, and then the algorithm and performance measurements were started. For BV, we initially created 3 logical trees of depth 1 on the network, rooted at each token holder, with an edge directed from each non-token holder to some root node. The BV algorithm maintained these logical trees (with links changing as token holders changed) throughout the execution, with the underlying routing protocol masking link changes from the application.

Figs. 30 and 31 are plots of the average number of time units and the average number of messages per CS entry for the BV, KRL, and KRLF simulations, respectively. We chose 1 for the highest load value because at this rate each node would have a request pending almost all the time. The low load value represents a much less busy network, with requests rarely pending at all nodes at the same time. Our choice for the value of the low mobility parameter corresponds to the situation where nodes remain stationary for up to a few minutes after moving and prior to making another move. The value of the high mobility parameter represents a much more volatile network, where nodes remain static for less than a minute between moves.

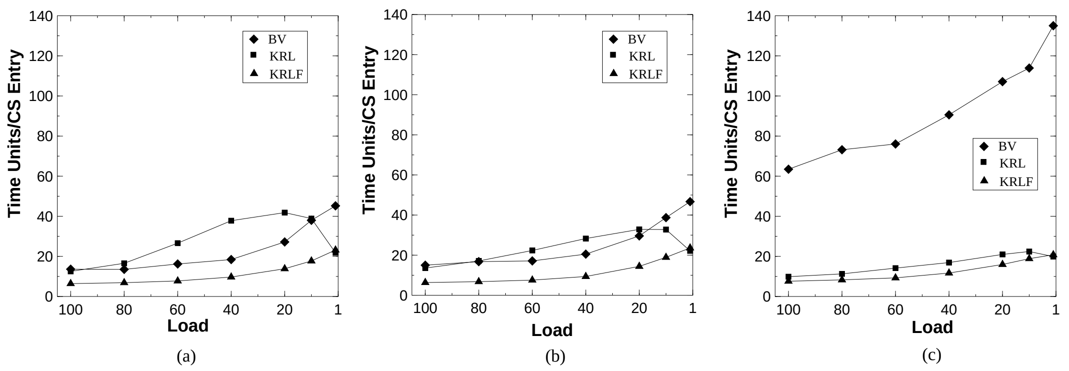


Fig. 30. Load vs. Time Units/CS Entry for 20% Connectivity at (a) Zero, (b) Low (1 Link Change every 500 Time Units), and (c) High (1 Link Change every 50 Time Units) Mobility with $k = 3$ (BV = Bulgannawar and Vaidya Algorithm, KRL = Basic k-Mutual Exclusion Algorithm, KRLF = KRL with Token Forwarding)

Fig. 30 shows that KRLF results in executions with lower average time per CS entry than either the KRL or BV algorithms.

Part (c) of Fig. 30 also shows that the BV algorithm is not well-suited for high mobility scenarios, whereas the time per CS entry for the KRLF algorithm is unaffected by increasing mobility and the time per CS entry for the KRL algorithm is actually *improved* when nodes are mobile. This reflects the ability of the KRL and KRLF algorithms to maintain high levels of token usage even when the topology is very dynamic. By allowing a token holder to "forget" requests that arrive on links that subsequently fail, our algorithms are able to provide uninterrupted access to other neighbors of that token holder. Notice that KRL's time per CS entry goes down at the very high load. The reason is the newly generated requests in the high load scenarios will follow the request chain of the previous requests, which shorten the time of maintaining the token oriented DAG.

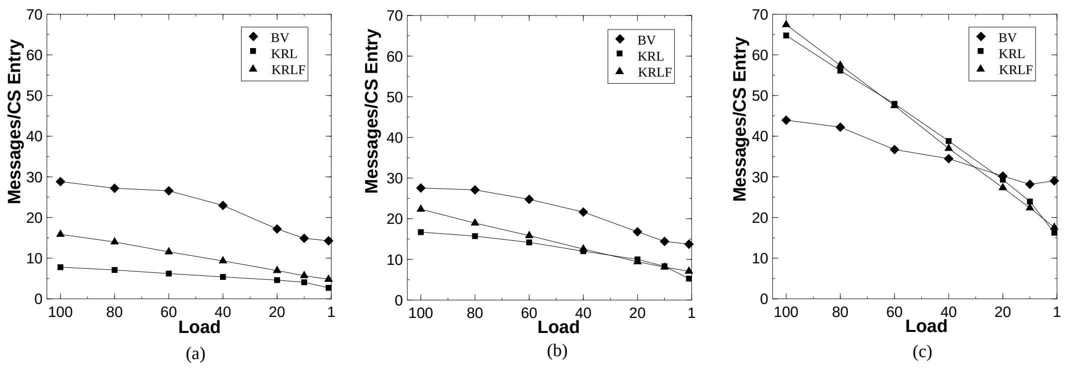


Fig. 31. Load vs. Messages/CS entry for 20% Connectivity at (a) Zero, (b) Low (1 Link Change every 500 Time Units), and (c) High (1 Link Change every 50 Time Units) Mobility with $k = 3$ (BV = Bulgannawar and Vaidya Algorithm, KRL = Basic k-Mutual Exclusion Algorithm, KRLF = KRL with Token Forwarding)

Fig. 31 shows that the number of messages sent in the BV simulation is not affected to a large extent by mobility. This is because the BV algorithm was intended for static networks and is oblivious to topology changes. Also, the number of messages used to find new routes in the BV simulation was not counted. The KRL and KRLF algorithms both send fewer messages per CS entry than the BV algorithm, except at the highest mobility when system load is low. When load is low and mobility is high, most messages in KRL and KRLF are *LinkInfo* messages, sent to repair the paths to token holders.

2. Comparison of KRL and KRLF on *ns-2*

To evaluate the relative performance of the KRL and KRLF algorithms in a "real world" scenario, we used the *ns-2* simulator [77] with CMU extensions for mobility [78]. We did not simulate the BV algorithm on the *ns-2* simulator since the simulation results from Section 1 indicate that both the KRL and KRLF algorithms outperform the BV algorithm in most situations and it is likely that the performance of the BV algorithm would be worse when run on top of a real routing protocol. Also, the BV algorithm assumes that that each node knows the global information of the network, which makes it poorly suited for ad hoc networks.

To model the mobility of the nodes, we used the *random waypoint* mobility model [78]. In each mobility scenario generated, the 30 nodes were initially placed at random positions on a two-dimensional grid. Nodes had identifiers $1 \ldots n$ and initially, nodes $1, 2$ and 3 were token holders. The transmission radius at each node was uniform with a length of 250m. Communication links formed when nodes were separated by 250m or less and links failed when the distance between them became greater than 250m. The algorithms ran as applications on top of the Dynamic Source Routing protocol (DSR) [59] and the IEEE 802.11 MAC layer. Each node gets its neighbor information by broadcasting *Hello* messages. The Hello interval is set to 2 seconds. The simulation box size was chosen to 500m $\times$ 500m to represent moderate connectivity. Average node speeds of 10, 5, and 0m/sec were used to represent high, low, and no mobility, respectively. We modeled the inter-arrival time of requests for the CS at each node as a Poisson process, and varied the expected interval between requests, recording executions with intervals of 100, 80, 60, 40, 20, and 10 seconds between requests.

In our performance evaluation, we measured: the average time, in seconds, for nodes to enter the CS after making a request (average time spent in WAITING section), the standard deviation of the waiting time to enter the CS and the average number of messages sent per CS entry. Each node spent exactly 1 second in the CS before releasing the token. Results reported here are averaged over 20 executions, each simulation run lasts for 1000 seconds. Simulation results using *ns-2* are plotted in the graphs of Figures 32, 33 and 34.

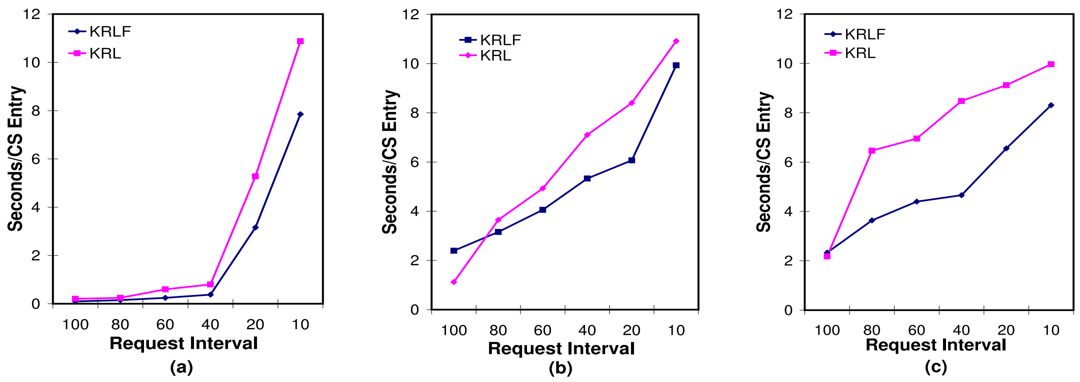


Fig. 32. Request Interval vs. Waiting Time/CS Entry for 500m × 500m Box at (a) Zero, (b) Low (5 m/sec), and (c) High (10 m/sec) Mobility with $k = 3$ (KRL = Basic k-Mutual Exclusion Algorithm, KRLF = KRL with Token Forwarding)

Figure 32 shows the average waiting time for each node to enter the CS under different request intervals and mobililties. Part (a) indicates that at 0 mobility and long request interval, nodes enter the CS with very little delay in both KRL and KRLF. This is caused, to a large extent, by the continual circulation of each token within its own disjoint subset of the network in KRL. Because nodes send requests to their lowest neighbors, the same place they last sent the token, requests tend to follow the same routes and nodes tend to repeatedly receive the same token. Since tokens are confined to a subset of processors in KRL, the paths over which *Request* and *Token* messages travel are short. Also, token holding nodes often enter the CS after negligible message delivery time, particularly at long request interval due to lack of contention. As mobility gets higher, the average waiting time increases. This is because the mobility causes the topology of the network and the DAG changed frequently, making the original request chain invalid. In most cases, the waiting time in KRLF is less than KRL because KRLF keeps forwarding the idle tokens in the network. KRLF does not help to reduce the waiting time in the low request interval because there are few number of requests in the system and the possibility of having idle tokens is less, token forwarding in this case seems to be an overkill.

Figure 33 shows the standard deviation of the waiting time for each node under different request intervals and mobilities. The standard deviation of both KRL and KRLF increases with system load and mobility. KRLF has smaller standard deviation value than KRL under the same request interval and mobility, especially in moderate system loads (request intervals of 80, 60 and 40s). This is because idle tokens in the system is forwarded by KRLF, which reduces the waiting time for requesting nodes. KRL does not forward tokens if there is no request in the queue. This may cause part of the network is jamed with requests, and the other part of the network with tokens does not have request, which results larger deviation in the waiting time.

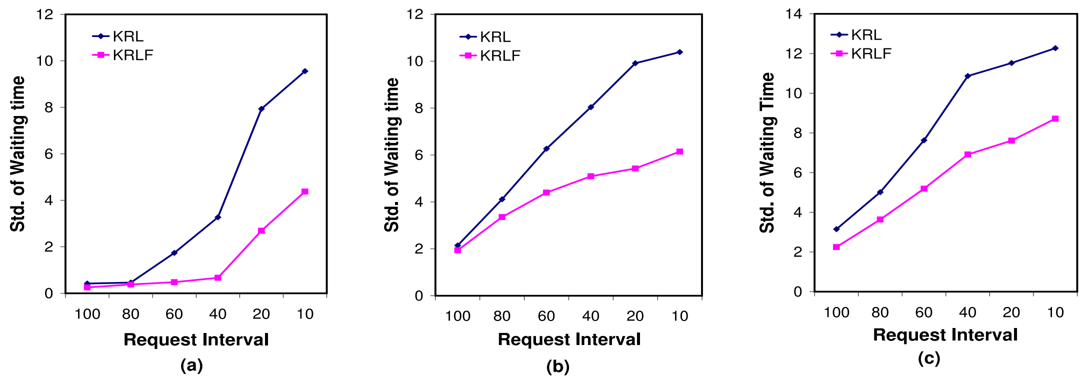


Fig. 33. Request Interval vs. Std. of the Waiting Time/CS Entry for 500m $\times$ 500m Box at (a) Zero, (b) Low (5 m/sec), and (c) High (10 m/sec) Mobility with k = 3 (KRL = Basic k-Mutual Exclusion Algorithm, KRLF = KRL with Token Forwarding)

Figure 34 shows the average number of messages to enter the critical section. KRLF sends more messages than KRL, because KRLF keeps forwarding the token even if there is no request in the network in order to reduces the waiting time to enter the critical section. Especially in low load cases, the number of messages sent by KRLF is significantly higher than KRL. But it does not really help reducing the waiting time as shown in Figure 32. This suggests a "smarter" token forwarding heuristic which is aware of the system load is more desirable. The average number of messages in high mobility case is higher because there are more link change messages due to the mobility. As the request interval gets shorter, the average number of messages decreases.

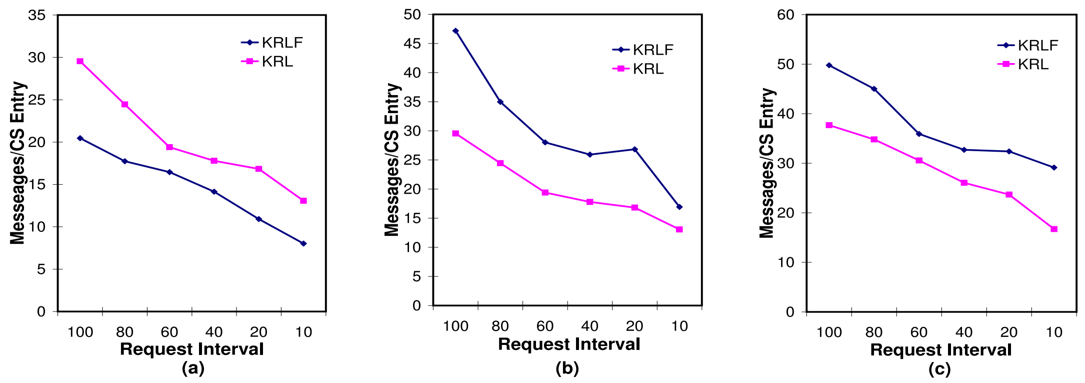


Fig. 34. Request Interval vs. Messages/CS Entry for 500m × 500m Box at (a) Zero, (b) Low (5 m/sec), and (c) High (10 m/sec) Mobility with $k = 3$ (KRL = Basic k-Mutual Exclusion algorithm, KRLF = KRL with Token Forwarding)

It should be noted that the difference between the results for the KRL and KRLF algorithms shown in Section 1 and those of the *ns-2* simulator in this section are in part due to the following factors:

- The simulation in Section 1 had no message contention or collision. Messages delay in the simulations in Section 1 were always delayed by 1 second, which is a normalized value in the simulator, whereas in the *ns-2* simulation, the message delay was variable, depending on the state of the wireless channel. In general, the message delay in *ns-2* was much less than 1 second when the system load was low.

- Whereas the connectivity in our simulation was allowed to vary from 10 to 30%, nearly any connectivity was possible in *ns-2*, depending on the distribution of nodes on the two-dimensional grid.

In summary, our results from using the *ns-2* simulator suggest that token forwarding improves performance, in terms of time per CS entry, when the network the topology is changing. However, there is a trade-off involved with this improvement because nodes send more messages when the topology is changing.

CHAPTER V

CONCLUSION AND FUTURE WORK

Studying fundamental services is necessary and important for building new distributed systems for mobile ad hoc networks. In this work, we studied some distributed services for mobile ad hoc networks, namely the clock synchronization, connected dominating set and k-mutual exclusion problems.

A. Clock Synchronization

Our clock synchronization algorithm exploits an advantage of the wireless communication medium and extends high synchronization accuracy to the whole network. Our algorithm takes advantage of the Reference Broadcast Synchronization (RBS), which can achieve high precision in synchronizing clocks within one broadcast domain, and generates a connected dominating set. By using the CDS as the synchronization backbone, synchronization is extended to multiple hops. This approach is adaptive to the mobile situations through the maintenance of the approximate MCDS. Through simulation, we show this algorithm can achieve good accuracy in synchronization (in microseconds). Our paper is published in the 2004 International Workshop on Mobile and Wireless Networking [5] (MWN04).

In the future work, we would like to prove a tight upper bound for our algorithm. Although our algorithm can achieve high accuracy in the synchronization, many applications in ad hoc networks do not need synchronized clocks all the time. We would like to develop an on-demand synchronization algorithm based on our present algorithm that still preserves the high quality in synchronization, but rather smaller energy consumption. Also, we are interested in combining techniques for estimating clock drift with our algorithm to provide even better synchronization accuracy.

B. Connected Dominating Set

In the second part of this work, we presented a randomized connected dominating set algorithm in mobile ad hoc networks. Our algorithm uses a randomized delay function for each node based only on its local information. Our algorithm uses fewer messages to find and maintain a CDS. We show that our algorithm can still find a CDS with a constant approximation of the optimal in terms of size. Moreover, our algorithm rotates the CDS nodes and distributes the energy consumption evenly in the whole network. Our simulation results show that our randomized CDS can extend the network lifetime and balance the energy consumption as compared to previous works. We also show how our CDS can be used as a routing backbone. This work is published in the International Conference on Wireless Networking [7] (ICWN04).

In future work, we will continue to study energy efficient communication in ad hoc networks. Energy efficiency is the most important issue in wireless networking. Currently there are two categories of approaches for energy efficient communication. The first category is topology control algorithms by power control, in which each individual node computes its own transmission power in order to generate a desired network topology for communication. However, there are some drawbacks in power control algorithms. First, most of power control algorithms generate unidrectional links in the network, which makes many of the existing routing algorithms inapplicable. In order to generate bidirectional links, extra steps have to be taken which brings more complexity to the algorithms. Second, it is hard to tell whether power control algorithms really benefit in terms of performance. Some distributed implementations of power control algorithms can hardly improve the throughput of mobile ad hoc networks [55].

The second category is based on a virtual backbone in the network. Backbone based communication has an advantage over power control techniques in that it avoids unidirectional links, which is important for many routing algorithms. Our current work on randomized CDS algorithm falls in this category. The simplicity and efficiency of randomized algorithms makes them more applicable to dynamic networks like mobile ad hoc and sensor networks. However, different communication primitives may require different backbones. Our future work is to develop efficient backbone algorithms that are efficiently adaptive to different service specifications and comprehensive performance analysis of the two energy efficient communication approaches. This can also be applied to sensor networks, which share numerous similarities with mobile ad hoc networks. Another issue is load balancing of the backbone. Since backbone nodes consume more energy than the other nodes in the network, rotation of the backbone is needed to prolong the overall network lifetime. The current work on randomized CDS rotates backbone nodes to achieve load balancing. However, the performance is evaluated only through simulation. Future work will include mathematical analysis on the load balancing of different backbone algorithms and developing backbone algorithms that can optimize the load balance.

C. k-Mutual Exclusion

In the third part of work, we studied the k-mutual exclusion problem. We presented a topology sensitive k-Reversal Link with Token Forwarding (KRLF) algorithm. KRLF is an extension of the k-Reversal Link (KRL) algorithm ([69]). The key idea of KRLF is to add a token forwarding technique to an idle token. By forwarding the idle token, nodes can access tokens more concurrently. We show that an idle token is guaranteed to be forwarded to all nodes in the network. Through simulation, we showed that at mid-range loads, the token forwarding technique does improve the time per CS entry without using more messages than the basic KRL algorithm when nodes are mobile. This work is published in the ACM Workshop on Principles of Mobile Computing in 2001 [9] (POMC01).

In future work, we would like to continue to experiment with heuristic modifications to the token forwarding strategy. A "self-adjusting" version of KRLF in which processors hold the token before forwarding it would allow processors to dynamically adjust the amount of time they hold an idle token before forwarding, based on their local view of the load on the system. The lower the detected system load, the longer the nodes would wait prior to forwarding the token. Our simulation results suggest that this strategy will not degrade performance even if the node has a temporarily incorrect view of the system load.

REFERENCES

[1] I. Stojmenovic, *Handbook of Wireless Networks and Mobile Computing*, New York, NY: Wiley Series on Parallel and Distributed Computing, 2002.

[2] J. Broch, D. A. Maltz, Y. Hu D. B. Johnson, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proceedings of the Fourth ACM Annual International Conference on Mobile Computing and Networking (MobiCom 98)*, Dallas,TX, October 1998.

[3] The Transis Project, "The transis group communication system," http://www.cs.huji.ac.il/labs/transis/. Accessed on July, 2004.

[4] Jennifer E. Walter, "Distributed algorithms for mobile computing systems," Ph.D. dissertation, Department of Computer Science, Texas A&M University, College Station, December, 2000.

[5] G. Cao and J. L. Welch, "Accurate multihop clock synchronization in mobile ad hoc networks," in *Proceedings of the 2004 International Workshop on Mobile Wireless Networking (MWN04)*, Montreal, Quebec, Canada, August 15-18, 2004, pp. 13–20.

[6] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation(OSDI)*, Boston, MA, 2002, pp. 147–164.

[7] G. Cao, "A randomized connected dominating set algorithm in wireless ad hoc networks," in *Proceedings of the 2004 International Conference on Wireless Networks (ICWN04)*, Las Vegas, Nevada, USA, June 21-24, 2004, pp. 16–21.

[8] X. Cheng and D.-Z. Du, "Virtual backbone-based routing in ad hoc wireless networks," Department of Computer Science and Engineering, University of Minnesota, Technical Report 02-002, 2002.

[9] J. E. Walter, G. Cao, and M. Mohanty, "A k-mutual exclusion algorithm for wireless ad hoc networks," in *Workshop on Principles of Mobile Computing*, Newport, RI, 2000.

[10] B. Aiken, J. Strassner, Cisco Systems, B. Carpenter, IBM, I. Foster, Argonne National Laboratory, C. Lynch, Coalition for Networked Information, J. Mambretti, ICAIR, R. Moore, UCSD, B. Teitelbaum, and Inc. Advanced Networks & Services, "Internet RFC 2768," February 2000, http://www.faqs.org/rfcs/rfc2768.html. Accessed on August 31, 2004.

[11] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," *Communication of the ACM*, vol. 5, no. 43, pp. 51–58, 2000.

[12] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *First ACM Workshop on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, GA, 2002, pp. 88–97.

[13] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transaction on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[14] S. PalChaudhuri, A. Saha, and D. B. Johnson, "Adaptive clock synchronization in sensor networks," in *3rd Symposium on Information Processiong in Sensor Networks (IPSN)*, Berkeley, CA, 2004, pp. 340–348.

[15] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 7, no. 21, pp. 558–565, 1978.

[16] J. Lundelius and N. A. Lynch, "An upper and lower bound for clock synchronization," *Information and Control*, vol. 62, pp. 190–204, 1984.

[17] J. Y. Halpern, N. Megiddo, and A. A. Munshi, "Optimal precision in the presence of uncertainty," *Journal of Complexity*, vol. 1, pp. 170–196, 1985.

[18] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, pp. 146–158, 1989.

[19] B. Patt-Shamir and S. Rajsbaum, "A theory of clock synchronization," in *Proceedings of 26th Annual ACM Symposium of Theory of Computing (STOC)*, 1994, pp. 810–819.

[20] Nelson Minar, "A survey of the ntp network," MIT Media Lab, 1999, http://www.media.mit.edu/ nelson.

[21] P. Veríssímo, L. Rodrigues, and A. Casimiro, "Cesiumspray: a precise and accurate global time service for large-scale systems," *Journal of Real-Time Systems*, vol. 3, no. 12, pp. 243–294, May 1997.

[22] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, "Clock synchronization for wireless local area networks," in *Proceedings of the 12th Euromicro Conference on Real Time Systems*, Stockholm, 2000, pp. 183–189.

[23] S. Mitra and J. Rabek, "Power efficient clustering for clock synchronization in dynamic multi-hop sensor networks," 2003, http://theory.mit.edu/ mitras/courses/6829/project/final-report.ps.

[24] R. Wattenhofer, L. Li, P. Bahl, and Y. M. Wang, "Distributed topology control for power efficient operation in multihop wireless ad hoc networks," in *IEEE INFOCOM*, 2001, pp. 1388–1397.

[25] H. Dai and R. Han, “Tsync: A lightweight bidirectional time synchronization service for wireless sensor networks,” *Mobile Computing and Communications Review*, vol. 8, no. 1, pp. 125–139, 2004.

[26] S. Ganeriwal, R. Kumar, S. Adlakha, and M. Srivastava, “Network-wide time synchronization in sensor networks,” Department of Electrical Engineering, UCLA, NESL Technical Report 01-01-2003, 2003.

[27] J. V. Greunen and J. Rabaey, “Lightweight time synchronization for sensor networks,” in *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA)*, 2003, pp. 11–19.

[28] M. L. Sichitiu and C. Veerarittiphan, “Simple, accurate time synchronization for wireless sensor networks,” in *IEEE Wireless Communications and Networking Conference (WCNC)*, New Orleans, LA, March 2003, vol. 2, pp. 1266–1273.

[29] Kay Römer, “Time synchronization in ad hoc networks,” in *The ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, Long Beach, CA, 2001, pp. 173–182.

[30] R. Fan and N. Lynch, “Gradient clock synchronization,” in *23rd Annual Symposium on Principles of Distributed Computing (PODC04)*, St. John’s, Canada, 2004, pp. 320–327.

[31] S. Ni, Y. Tseng, Y. Chen, and J. Sheu, “The broadcast storm problem in a mobile ad hoc network,” in *Proceedings of the ACM/IEEE Internation Conference on Mobile Computing and Networking (MOBICOM)*, 1999, pp. 151–162.

[32] P. Sinha, R. Sivakumar, and V. Bharghavan, “Enhancing ad hoc routing with dynamic virtual infrastructures,” in *INFOCOM*, Anchorage, Alaska, 2001, pp.

1763–1772.

[33] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, San Francisco: Freeman, 1978.

[34] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz, “Simple heuristics for unit disk graphs,” *Networks*, vol. 25, pp. 59–68, 1995.

[35] V. Chvatal, “A greedy heuristic for the set-covering problem,” *Mathematics of Operation Research*, vol. 4, no. 3, pp. 233–235, 1979.

[36] S. Guha and S. Khuller, “Approximation algorithms for connected dominating sets,” in *Proceedings of 4th Annual European Symposium on Algorithyms*, 1996, pp. 179–193.

[37] T. Grossman and A. Wool, “Computational experience with approximation algorithms for the set covering problem,” *European Journal of Operational Research*, vol. 101, no. 1, pp. 81–92, 1997.

[38] L. Jia, R. Rajaraman, and T. Suel, “An efficient distributed algorithm for constucting small dominating sets,” in *Twentieth ACM Symposium on Principles of Distributed Computing (PODC)*, Newport, RI, 2001, pp. 33–42.

[39] K. M. Alzoubi, P. J Wan, and O. Frieder, “Distributed heuristics for connected dominating sets in wireless ad hoc networks,” *Journal of Communications and Networks*, vol. 4, no. 1, pp. 22–29, 2002.

[40] J. Wu and H. Li, “On calculating connected dominating set for efficient routing in ad hoc wireless networks,” in *Proceedings of the 3rd international workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Seattle, Washington, United States, 1999, pp. 7–14.

[41] I. Stojmenovic, M. Seddigh, and J. Zunic, "Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 1, pp. 14–25, 2002.

[42] H. Kopetz and W. Schwabl, "Global time in distributed real-time systems," Technische Universität Wien, Technical Report 15/89, 1989.

[43] K. M. Alzoubi, P. J Wan, and O. Frieder, "Message-optimal connected dominating sets in mobile ad hoc networks," in *Proceedings of The Third ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, 2002, pp. 157–164.

[44] G. Cao and J. L. Welch, "A clock synchronization service in mobile ad hoc networks," Tech. Rep., Dept. of Computer Science, Texas A&M University, 2003, http://students.cs.tamu.edu/g0c7670/TechReport03.pdf.

[45] VINT Group, "The network simulator - ns-2," http://www.isi.edu/nsnam/ns. Accessed on November, 2003.

[46] The CMU Monarch Project, "Wireless and mobility extension to ns," http://www.monarch.cs.cmu.edu. Accessed on October, 2004.

[47] B. Das and V. Bharghavan, "Routing in ad-hoc networks using minimum connected dominating sets," in *IEEE International Conference on Communications (ICC)*, Richardson, TX, 1997, pp. 376–380.

[48] B. Liang and Z. J. Haas, "Virtual backbone generation and maintenance for ad hoc network mobility management," in *Proceedings of 2000 IEEE INFOCOM*, Tel Aviv, Israel, 2000, pp. 1293–1302.

[49] R. Sivakumar, P. Sinha, and V. Bharghavan, "Cedar: a core-extraction distributed ad hoc routing algorithm," *IEEE Journal of Selected Areas in Communications*, vol. 17, no. 8, pp. 1454–1465, August 1999.

[50] B. Williams and T. Camp, "Comparison of broadcasting techniques for mobile ad hoc networks," in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobileHOC)*, 2002, pp. 194–205.

[51] K. Alzoubi, X. Y. Li, Y. Wang, P. J. Wan, and O. Frieder, "Geometric spanners for wireless ad hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, pp. 408–421, 2003.

[52] A. D. Amis and R. Prakash, "Load-balancing clusters in wireless ad hoc networks," in *The 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET)*, Richardson, TX, 2000, pp. 25–32.

[53] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks," *ACM Wireless Networks Journal*, vol. 8, no. 5, pp. 481–494, 2002.

[54] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in *Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, Rome, Italy, 2001, pp. 70–84.

[55] R. Ramanathan and R. Hain, "Topology control of multihop wireless networks using transmit power adjustment," in *IEEE INFOCOM*, Tel-Aviv, Israel, 2000, pp. 404–413.

[56] L. Bao and J. J. Garcia-Luna-Aceves, "Topology management in ad hoc networks," in *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking*

& Computing (MobiHoc), 2003, pp. 129–140.

[57] C. C. Chiang, H. K. Wu, W. Liu, and M. Gerla, "Routing in clustered multihop, mobile wireless networks with fading channel," in *IEEE Singapore International Conference on Networks (SICON'97)*, 1997, pp. 14–17.

[58] Y. Xu, J. Heidemann, and D. Estrin, "Adaptive energy-conserving routing for multihop ad hoc networks," Information Sciences Institute, University of Southern California, Technical Report 527, 2000.

[59] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*, T. Imielinski and H. Korth, Eds., chapter 5, pp. 153–181. Boston, MA: Kluwer Academic Publishers, 1996.

[60] C. E. Perkins and M. Belding-Royer, "Ad-hoc on-demand distance vector routing," in *Second Workshop on Mobile Computing Systems and Applications (WMCSA99)*, New Orleans, LA, 1999, pp. 90–100.

[61] R. Baldoni, A. Virgillito, and R. Petrassi, "A distributed mutual exclusion algorithm for mobile ad hoc networks," in *Proc. of 7th Intl. Symp. on Computers and Communications*, 2002, pp. 539–544.

[62] D. Kim, C.-K. Toh, and Y. Choi, "Improving tcp performance in wireless ad hoc networks," in *Proc. of IEEE Intl. Conf. on Communications*, 2000, vol. 3, pp. 1707–1713.

[63] D. Sun and H. Man, "Enic - an improved reliable transport scheme for mobile ad hoc networks," in *Proc. of IEEE Global Telecommunications Conf.*, 2001, vol. 5, pp. 2852–2856.

[64] S. T. Huang, J. R. Jiang, and Y. C. Kuo, "K-coteries for fault-tolerant k entries to a critical section," in *Proc of IEEE Intl. Conf. on Distributed Computing Systems*, 1993, pp. 74–81.

[65] K. Raymond, "A distributed algorithm for multiple entries to a critical section," *Information Processing Letters*, vol. 30, pp. 189–193, Feb. 1989.

[66] S. Bulgannawar and N. H. Vaidya, "A distributed k-mutual exclusion algorithm," in *Proc. of IEEE Intl. Conf. on Distributed Computing Systems*, 1995, pp. 153–160.

[67] K. Makki, P. Banta, K. Been, N. Pissinou, and E. Park, "A token based distributed k mutual exclusion algorithm," in *Proc. of IEEE Symp. on Parallel and Distributed Processing*, 1992, pp. 408–411.

[68] P. K. Srimani and R. L. Reddy, "Another distributed algorithm for multiple entries to a critical section," *Information Processing Letters*, vol. 41, pp. 51–57, 1992.

[69] J. E. Walter, J. L. Welch, and N. H. Vaidya, "A mutual exclusion algorithm for ad hoc mobile networks," *Wireless Networks*, vol. 4, no. 6, pp. 585–600, 2001.

[70] Y. Chen and J. Welch, "Self-stabilizing mutual exclusion using tokens in mobile ad hoc networks," in *Proc. of Dial-M for Mobility Workshop*, 2002, pp. 34–42.

[71] J. R. Jiang, "A distributed h-out of-k mutual exclusions algorithm for ad hoc mobile networks," in *Proc. of the Intl. Parallel and Distributed Processing Symp.*, 2002, pp. 196–202.

[72] E. Gafni and D. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Transactions on*

Communications, vol. C-29, no. 1, pp. 11–18, 1981.

[73] Y. Chang, M. Singhal, and M. Liu, "A fault tolerant algorithm for distributed mutual exclusion," in *Proc. of 9th IEEE Symp. on Reliable Dist. Systems*, 1990, pp. 146–154.

[74] D. M. Dhamdhere and S. S. Kulkarni, "A token based k-resilient mutual exclusion algorithm for distributed systems," *Information Processing Letters*, vol. 50, pp. 151–157, 1994.

[75] K. Raymond, "A tree-based algorithm for distributed mutual exclusion," *ACM Transactions on Computer Systems*, vol. 7, no. 1, pp. 61–77, 1989.

[76] J. L. Welch and N. A. Lynch, "A modular drinking philosophers algorithm," *Distributed Computing*, vol. 6, pp. 233 – 244, 1993.

[77] V. P. Team, "The network simulator – ns-2," VINT Project Team, available at http://www.isi.edu/nsnam/ns/, 2000.

[78] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proc. of ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, 1998, pp. 85–97.

[79] S. Basagni, I. Chlamtac, and V. R. Syrotiuk, "A distance routing effect algorithm for mobility (dream)," in *Proc. of ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, 1998, pp. 76–84.

[80] R. Casteneda and S. R. Das, "Query localization techniques for on-demand routing protocols in ad oc networks," in *Proc. ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, 1999, pp. 186–194.

[81] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek, and M. Degermark, "Scenario-based performance analysis of routing protocols for mobile ad hoc networks," in *Proc. ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, 1999, pp. 195–206.

[82] Y. B. Ko and V. H. Vaidya, "Location-aided routing (lar) in mobile ad hoc networks," in *Proc. of ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, 1998, pp. 66–75.

[83] E. M. Royer and C. E. Perkins, "Multicast operation of the ad hoc on-demand vector routing protocol," in *Proc. ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, 1999, pp. 207–218.

APPENDIX A

PSEUDOCODE FOR KRL ALGORITHM

```
When node $i$ requests access to the CS:

 1. $status$ := WAITING
 2. $Enqueue(Q, i)$
 3. if (!$tokenHolder$)
 4.      if ($|Q|$ = 1) $ForwardRequest()$
 5. else $GiveTokenToNext()$
```

```
When node $i$ releases the CS:

 1. if ($|Q| > 0$) $GiveTokenToNext()$
 2. $status$ := REMAINDER
 3. if ($myHeight$ > $height[k]$, $\forall k \in N$)
 4.        $LowerHeight()$
```

Fig. 35. Pseudocode Triggered by Input Events from Application Process

```
$Request(h)$ received at node $i$ from node $j$:
($h$ denotes $j$'s height when message was sent)
 1. if ($receivedLI[j]$)
 2.   $height[j]$ := $h$
 3.   if ($myHeight$ < $height[j]$) $Enqueue(Q, j)$
 4.     if ($tokenHolder$)
 5.       if (($|Q| > 0$) and
          (($status$ = REMAINDER) or (($status$ = CRITICAL) and ($numTokens > 1$))))
 6.         $GiveTokenToNext()$
 7.     else // not tokenHolder
 8.       if ($myHeight$ < $height[k]$, $\forall\ k \in\ N$)
 9.         $RaiseHeight()$
10.       else if (($Q = [j]$) or (($|Q| > 0$) and ($myHeight$ < $height[next]$))
11.          $ForwardRequest()$  //reroute request
```

Fig. 36. Pseudocode Triggered by *Recv*(j, *Request*) Network Input Events

```
$Token(h)$ received at node $i$ from node $j$:
$(h$ denotes $j$'s height when message was sent$)$
 1. $tokenHolder$ := true
 2. $numTokens$++
 3. $height[j]$ := $h$
 4. if ($myHeight > h$)
 5.   Send $LinkInfo(h.h1, h.h2$ - 1,$i$) to all outgoing $k \in N$ except $j$
 6.     $myHeight.h1$ := $h.h1$
 7.     $myHeight.h2$ := $h.h2$ - 1 // lower height
 8. Send $LinkInfo(h.h1, h.h2$ - 1, $i$) to $j$
 9. if ($|Q| > 0$) $GiveTokenToNext()$
10. else $next$ := $i$
```

Fig. 37. Pseudocode Triggered by *Recv*(j, *Token*) Network Input Event

```
$LinkInfo(h)$ received at node $i$ from node $j$:
$(h$ denotes $j$'s height when message was sent$)$
 1. $N$ := $N \cup \{j\}$
 2. if (($forming[j]$) and ($myHeight \neq formHeight[j]$))
 3.   Send $LinkInfo(myHeight)$ to $j$
 4. $forming[j]$ := false
 5. if ($receivedLI[j]$) $height[j]$ := $h$
 6. else if ($height[j] = h$) $receivedLI[j]$ := true
 7. if ($myHeight > height[j]$) $Delete(Q, j)$
 8. if ($tokenHolder$)
 9.   if ($myHeight > height[k], \forall k \in N$)
10.     $LowerHeight()$
11. if (($myHeight < height[k], \forall k \in N$) and (not $tokenHolder$))
12.   $RaiseHeight()$
13. else if (($|Q| > 0$) and ($myHeight < height[next]$))
14.   $ForwardRequest()$  // reroute request
```

Fig. 38. Pseudocode Triggered by *Recv*(j, *LinkInfo*) Network Input Event

```
Failure of link to $j$ detected at node $i$:
 1. $N$ := $N - \{j\}$
 2. $Delete(Q, j)$
 3. $receivedLI[j]$ := true
 4. if (not $tokenHolder$)
 5.   if ($myHeight < height[k], \forall k \in N$)
 6.     $RaiseHeight()$   // reroute request
 7.   else if (($|Q| > 0$) and ($next \notin N$))
 8.     $ForwardRequest()$
 9. else if ($myHeight > height[k], \forall k \in N$)
10.   $LowerHeight()$
Formation of link to $j$ detected at node $i$:
 1. Send $LinkInfo(myHeight)$ to $j$
 2. $forming[j]$ := true
 3. $formHeight[j]$ := $myHeight$
```

Fig. 39. Pseudocode Triggered by *LinkDown* and *LinkUp* Network Enput Events

```
Procedure $ForwardRequest$():
 1.  $next$ := $l \in N$ : $height[l] \leq height[j]\ \forall\ j \in N$
 2.  Send $Request(myHeight)$ to $next$

Procedure $LowerHeight$():
 1. $myHeight.h1$ := $\max_{k \in N}\{height[k].h1\} - 1$
 2. $S := \{l \in N : height[l].h1 = myHeight.h1\}$
 3. if ($S \neq \emptyset$) $myHeight.h2$ := $\max_{l \in S}\{height[l].h2\} + 1$
 4. Send $LinkInfo(myHeight)$ to all incoming $k \in N$

Procedure $RaiseHeight$():
 1. $myHeight.h1$ := 1 + $\min_{k \in N}\{height[k].h1\}$
 2. $S := \{l \in N : height[l].h1 = myHeight.h1\}$
 3. if ($S \neq \emptyset$) $myHeight.h2$ := $\min_{l \in S}\{height[l].h2\} - 1$
 4. Send $LinkInfo(myHeight)$ to all $k \in N$
 5. for (all $k \in N$ such that $myHeight > height[k]$) do
 6.     $Delete(Q, k)$
 7. if ($|Q| > 0$) $ForwardRequest$()

Procedure $GiveTokenToNext$():
 1.  $next$ := $Dequeue(Q)$
 2.  if ($next \neq i$)   // send token
 3.      $numTokens$--
 4.      if ($numTokens$ = 0)
 5.          $tokenHolder$ := false
 6.      $height[next]$ := ($myHeight.h1$, $myHeight.h2 - 1$, $next$)
 7.      $receivedLI[next]$ := false
 8.      Send $Token(myHeight)$ to $next$
 9.      if (($numTokens$ = 0) and ($|Q| > 0$))
10.        Send $Request(myHeight)$ to $next$
11.  else  // keep token
12.     $status$ := CRITICAL
13.     Enter CS
```

Fig. 40. Procedures of KRL Algorithm

VITA

Guangtong Cao was born in Jia MuSi, Hei LongJiang Province, China. He received his B.E. and M.E. degree in Computer Science from Beijing University of Aeronautics & Astronautics in 1994 and 1997. He began pursuing a Ph.D. degree in Computer Science at Texas A&M University in 2000. Since then, he has worked as a graduate teaching assistant and research assistant for Dr. Jennifer L. Welch in the Department of Computer Science, Texas A&M University. His permenant address is: Department of Computer Science, Texas A&M University, 301 Harvey R. Bright Bldg, College Station, TX 77843-3112.