AN ℓ–*EXCLUSION* ALGORITHM FOR MOBILE AD HOC NETWORKS

by

Salahuddin Mohammad Masum

A Thesis Submitted to the Department of Computer Science and Engineering in the Partial Fulfillment of the Requirements for the Degree of Master of Science in Engineering (Computer Science and Engineering)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DHAKA, BANGLADESH

NOVEMBER 2005

The thesis "**An *ℓ–Exclusion* Algorithm for Mobile Ad Hoc Networks**", submitted by Salahuddin Mohammad Masum, Roll No. 040205051P, Registration No. 94277, Session April 2002, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Master of Science in Engineering (Computer Science and Engineering) and approved as to its style and contents. Examination held on November 15, 2005.

Board of Examiners

1. Dr. Md. Mostofa Akbar
Assistant Professor
Department of CSE
BUET, Dhaka–1000
— Chairman (Supervisor)

2. Dr. Muhammad Masroor Ali
Professor and Head
Department of CSE
BUET, Dhaka–1000
— Member (Ex–officio)

3. Dr. Abu Sayed Md. Latiful Hoque
Associate Professor
Department of CSE
BUET, Dhaka–1000
— Member

4. Dr. Md. Saidur Rahman
Associate Professor
Department of CSE
University of Dhaka
Dhaka–1000
— Member

5. Dr. M. Lutfar Rahman
Professor
Department of CSE
University of Dhaka
Dhaka–1000
— Member (External)

DECLARATION

I, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by me under the supervision of Dr. Md. Mostofa Akbar, Assistant Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. I also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Dr. Md. Mostofa Akbar)
Supervisor

Signature

(Salahuddin Mohammad Masum)
Candidate

ACKNOWLEDGEMENT

First I express my heartiest thanks and gratefulness to Almighty Allah for His divine blessings, which made me possible to complete this thesis successfully.

I feel grateful to and wish to acknowledge my profound indebtedness to Dr. Md. Mostofa Akbar, Assistant Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology. Deep knowledge and keen interest of Dr. Akbar in the field of system design influenced me to carry out this project and thesis. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stage have made it possible to complete this thesis.

I would like to thank the members of the graduate committee, Dr. Muhammad Masroor Ali, Professor and Head, Dr. Abu Sayed Md. Latiful Hoque, Associate Professor, Dr. Md. Saidur Rahman, Associate Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, and Dr. M. Lutfar Rahman, Professor, Department of Computer Science and Engineering, University of Dhaka for their valuable suggestions.

I am also very much grateful to Professor Dr. Aminul Islam, Vice Chancellor, Daffodil International University, who inspired me many times to complete this thesis.

I would also like to express my heartiest gratitude to other faculty members, the staffs of the CSE Department of BUET, and the same of Daffodil International University.

And last but not the least, I must acknowledge with due respect the constant support and patience of my mother, my father, my father–in–law, my mother–in–law, my wife, Sifat, and my younger brother, Mamun for completing the thesis.

ABSTRACT

With the proliferation of portable computing platforms and small wireless devices, ad hoc mobile networks have received a substantial attention from the research community as a means for providing data communications among devices regardless of their physical locations. Much of the research attention has spanned the design and standardization of routing, medium access control protocols, wireless channel allocation algorithms, protocols for broadcasting and multicasting. Instead, this thesis work addresses the *ℓ–Exclusion* problem for mobile ad hoc networks. The *ℓ–Exclusion* problem, a generalization of distributed mutual exclusion problem, involves a group of processes, each of which intermittently requires access to one of ℓ identical resources or pieces of code called the critical section (CS). However, characteristics of mobile ad hoc networks, such as concurrent and unpredictable topology changes due to arbitrary mobility pattern of nodes, shared broadcast channel, dynamic wireless link formation and removal, network partitioning and disconnections, location dependent errors, highly variable message delay, bandwidth, energy and battery power limitations, make devising of any distributed solution to the *ℓ*–exclusion problem in such networks very challenging and exciting.

In literature, few token–based solutions to this problem are available. Nevertheless, these solutions suffer from poor failure resiliency, as these do consider failures associated with mobile ad hoc networks, such as loss or regeneration of tokens, crash or sudden recovery of nodes. This research work presents a consensus–based mobility–aware *ℓ*–exclusion (LE) algorithm that operates asynchronously and copes explicitly with arbitrary (possibly concurrent) topology changes associated with such networks. The algorithm is fault–resilient in the sense that it can tolerate loss of messages, link changes or failures, sudden crashes or recoveries of at most $\ell-1$ mobile nodes. The algorithm is based on collection enough consensuses for a mobile node intending to enter CS, and uses diffusing computations for this purpose. The algorithm requires nodes to communicate only with their current neighbors, making it well–suited for use in mobile ad hoc networks. This thesis presents proofs of correctness to exhibit the fairness of the algorithm. This work is concluded by an extensive simulation study considering several performance metrics that significantly

impact the behavior of such an algorithm in various ad hoc settings. Simulation study demonstrates that the proposed algorithm is quite effective to variety of operating conditions, and is highly adaptive to frequent and unpredictable topology changes due to loss of messages, link changes or failures or formations, sudden crashes or recoveries of at most ℓ–1 mobile nodes, under different mobility settings. This research work also presents a simulation–based performance comparison between the proposed ℓ–exclusion (LE) algorithm and the *k–Reverse Link* (KRL) algorithm. Simulation results sdemonstrate that our algorithm performs favorably at high crash/merge rate in terms of *Message Overhead*, particularly when nodes are mobile. The performance of our algorithm in terms of *Average Waiting Time per CS Entry* is remarkable, particularly under mobility and vulnerability. Simulation results also shows that the ℓ–exclusion algorithm always allows more than 90% of nodes intending to enter CS to access and to control CS under a variety of vulnerable mobility settings, where at most ℓ–1 node(s) can crash independently or concurrently, and/or crashed node(s) may recover from failures.

TABLE OF CONTENTS

LIST OF FIGURES

To my beloved son

Mufrad Sharaf Bin Salahuddin

CHAPTER 1

INTRODUCTION

1.1. Motivation

Numerous factors associated with technology, business, regulation and social behavior naturally and logically speak in favor of mobile ad hoc networking. Mobile wireless data communication, which is advancing both in terms of technology and usage/penetration, is a driving force of truly ubiquitous computing and communication. In the near future, the role and capabilities of short–range data transaction are expected to grow, serving as a complement to traditional large–scale communication: most man–machine communication as well as oral communication between human beings occurs at distances of less than 10 meters; also, as a result of this communication, the two communicating parties often have a need to exchange data. As an enabling factor, license–exempted frequency bands invite the use of developing radio technologies (such as Bluetooth) that admit effortless and inexpensive deployment of wireless communication. Traditional cellular and mobile networks are still, in some sense, limited by their need for infrastructure (i.e., base stations, routers). For mobile ad hoc networks, this final limitation is eliminated. Mobile ad hoc networks are the key to the evolution of wireless networks [109].

But perhaps the most widespread notion of a mobile ad hoc network (MANET) is a network formed without any central administration. This kind of network provides a flexible way of developing ubiquitous broadband wireless access, allowing mobile networks to be readily deployed unpredictably without using any previous network infrastructure. Such networks are multi–hop wireless networks that consist of mobile nodes using a wireless interface to send packet data. Since the nodes in a network of this kind can serve as routers and hosts, they can forward packets on behalf of other nodes and run user applications, but they may disappear from, appear into or move within the network at any time.

Mobile ad hoc networks have mainly been considered for military tactical communications in automated battlefields, where a decentralized network configuration is an operative advantage or even a necessity. However, interest in this type of networks continues to grow. Applications such as rescue missions in times of natural disaster recovery, law enforcement operations, crowd control, search and rescue, commercial and educational use, home networking, personal area networking, and sensor networks are just a few possible examples.

In recent years, the vision of nomadic computing with its ubiquitous access has stimulated much research interest in the mobile ad hoc networking technology. Much of this research activity has focused on the design and standardization of routing [30, 35, 43, 49, 63, 68, 70, 76–78, 113, 117, 118, 126] medium access control protocols [14, 134] wireless channel allocation algorithms [53], protocols for broadcasting and multicasting [30, 44, 111, 125]. However, there has been a limited research works on distributed algorithms designed for mobile ad hoc networks while implementing distributed algorithms that ensure control and ordering is a critical point for many specific problems (e.g. resource allocation; assigning channels, IP addresses; autoconfiguration; logical structure; etc.). For traditional static and dynamic networks, there is a rich history of works on distributed algorithms for various problems including clock synchronization [81, 82], mutual exclusion [39, 84], leader election [52], Byzantine agreement [41, 42] etc. In this research work, we intend to fill in this gap for MANET, addressing a distributed protocol for the *ℓ–Exclusion* problem, which is a generalization of 1–mutual exclusion [39, 84], first defined and solved by Fisher et al. [45, 46] for static networks. The *ℓ–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 ℓ, $1 \leq \ell \leq n$, processes may be in the CS at any given time. Providing shared access to resources through mutual exclusion is a fundamental problem in computer science, and is worth considering for the ad hoc locale, where stripped down mobile nodes may need to share resources.

Nevertheless, implementing distributed algorithms (e.g. distributed *ℓ–Exclusion* algorithm) in mobile ad hoc wireless networks entails complications, as mobile ad hoc networks inherit the traditional problems of wireless and mobile communications.

Following issues signify mobile ad hoc network as a challenging research domain for implementing distributed algorithms:

- Networks don't (necessarily) have a pre–existing infrastructure.
- The topology is highly dynamic and frequent changes in the topology may be hard to predict.
- The mobility pattern of nodes is arbitrary.
- Mobile ad hoc networks are based on wireless links, which will continue to have a significantly lower capacity than their wired counterparts.
- Routes between nodes may potentially contain multiple hops.
- Messages may need to traverse multiple links to reach a destination (See Figure 1.1). Hence, some nodes may receive messages that are not intended for those nodes.
- Broadcast nature of wireless medium causes hidden terminal problem (See Figure 1.2).
- Mobility causes route changes (See Figure 1.3).
- Mobility induces packet losses.
- Packets are lost due to transmission errors.
- Wireless link can be formed and removed dynamically.
- Wireless transmission ranges of mobile nodes are limited.
- Mobile nodes may have limited bandwidth.

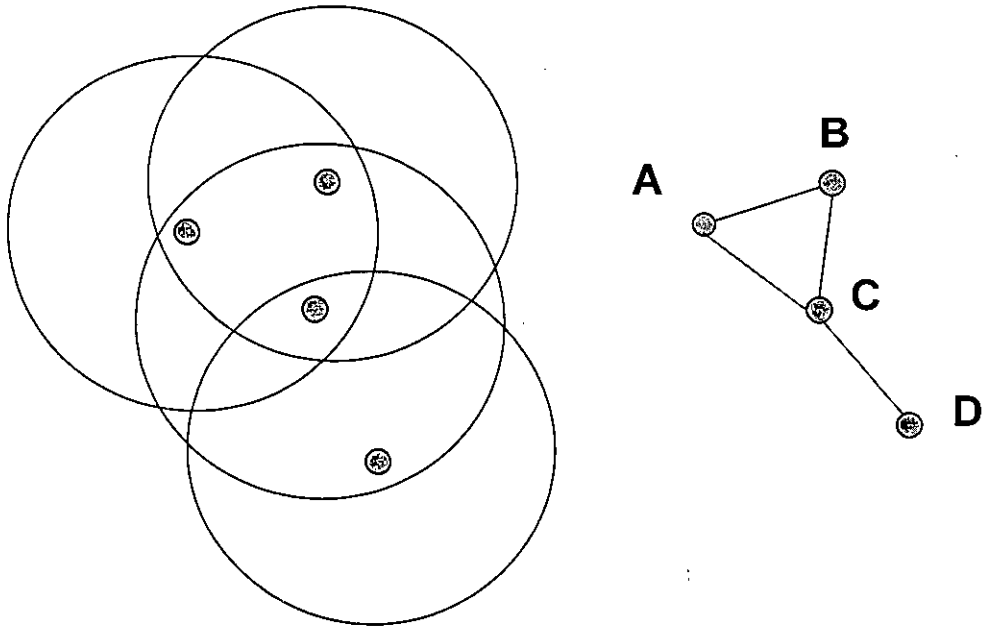


Figure 1.1. Multiple links to reach a destination.

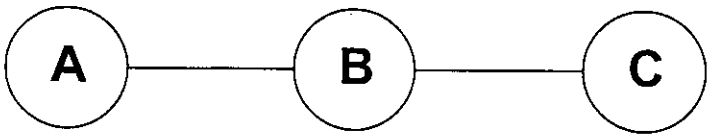


Figure 1.2. Hidden Terminal Problem. Nodes A and C cannot hear each other. Transmissions by nodes A and C can collide at node B. Nodes A and C are hidden from each other.

- Message delays may be highly variable.
- Physical security is limited due to the wireless transmission.
- Mobile ad hoc networks are affected by higher loss rates, and can experience higher delays and jitter than fixed networks due to the wireless transmission.
- Snooping on wireless transmissions (security hazard) is easy.
- Network partitioning and potential disconnections may occur randomly.
- Mobile ad hoc network nodes rely on batteries or other exhaustible power supplies for their energy. As a consequence, energy savings are an important system design criterion. Furthermore, nodes have to be power–aware: the set of functions offered by a node depends on its available power (CPU, memory, etc.).

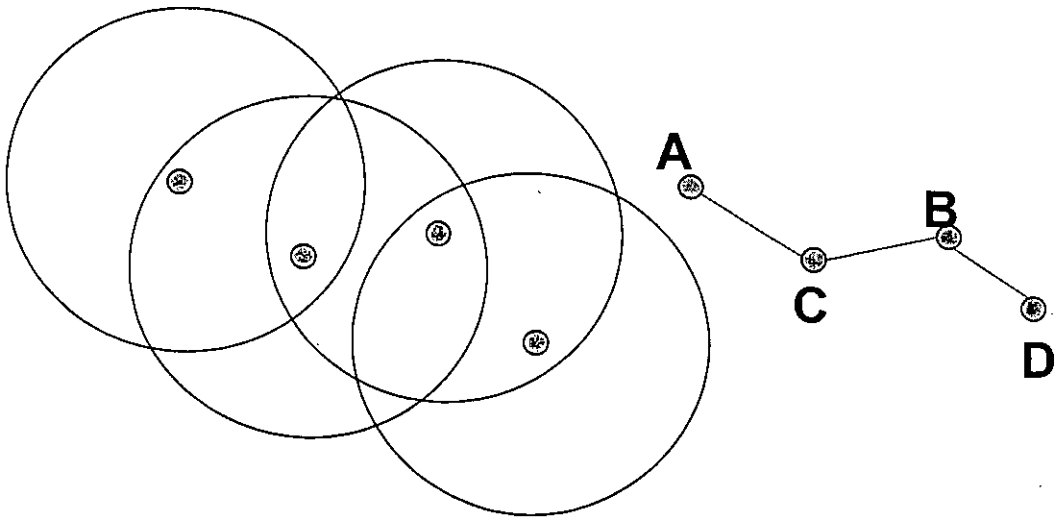


Figure 1.3. Mobility causes route changes.

Therefore, resilient and adaptive distributed algorithms that can continue to perform potentially and effectively under mobile ad hoc locale can significantly enhance MANET operations from a user's perspective. Such algorithms can also significantly

ease the design pressure in complex engineering areas such as quality of service (QoS) [89]. So, to design and to develop a fault–resilient and mobility aware distributed *ℓ–Exclusion* algorithm for mobile ad hoc domain has become a focus of recent research and development efforts under mobile ad hoc domain.

1.2. Problem Definition

Mutual exclusion [39, 84] is one of most classical paradigms of distributed computing. This problem consists in devising a protocol run by a set of communicating asynchronous parallel processes which want to coordinate themselves to access and to control intermittently a designated piece of code, namely critical section (CS) that can be used only by one process at a time. Code of critical section might manipulate a common resource, in which case access to the critical section corresponds to allocation of the resource, such as non–sharable, reusable line printer or a tape drive or other output device that requires exclusive access in order to ensure that the output is sensible, or a database or other data structure that requires exclusive access in order to avoid interference among the operations of different users.

Figure 1.4 shows a model that we can use for examining approaches to mutual exclusion in a distributed context. We assume some number of systems interconnected by some type of networking facility. Within each system, we assume that some function or process is responsible for resource allocation. Each such process controls a number of resources, and serves a number of user processes. Any solution to distributed mutual exclusion devises a protocol by which these processes may cooperate in enforcing mutual exclusion.

The *ℓ–Exclusion* problem, first defined and solved by Fisher et al. [45, 46], is a generalization of mutual exclusion (1–mutual exclusion) for multiple entries to CS. The *ℓ–Exclusion* problem generalizes the mutual exclusion problem to the case where some number $\ell \geq 1$ of processes (but not more) are permitted to be simultaneously in their critical sections. Regarded as a resource–allocation problem, we consider ℓ identical copies of a non–sharable reusable resource, where each process can request at most one copy of that resource. Again entry to the critical section corresponds to

allocation of a resource copy, but the problem excludes the questions of just how the individual copies of the resource are managed.

For example, imagine that each process controls some device which from time to time needs to enter a mode of high electrical power consumption. The main circuit breaker can withstand at most E devices at high electrical power consumption. By allowing each process to switch its device on only when it is in its critical section, an *ℓ–Exclusion* solution will protect the circuit breaker from burning out. Applications of *ℓ–Exclusion* include resource allocation, assigning channels, assigning IP addresses etc.

To illustrate the problem, assume a system that consists of n processes $\{1, ..., n\}$. Each process can be described by a program that consists of two distinguished sections: a *remainder section* and a *critical section*. Each process alternates between executing its remainder and its critical section as follows:

```
Process i:
        repeat forever
                remainder_section_i
                critical_section_i
        end repeat
```

The *ℓ–Exclusion* problem is to guarantee that the system does not enter a state in which more than ℓ processes are in their CS. Roughly, if fewer than ℓ processes are in their critical sections, it is possible for another process to enter its critical section, even though no process leaves its critical section in the meantime. To coordinate the entrance to the critical section, entry and exit sections are added to the program of each process:

```
Process i:
        repeat forever
                remainder_section_i
                entry_i
                critical_section_i
                exit_i
        end repeat
```

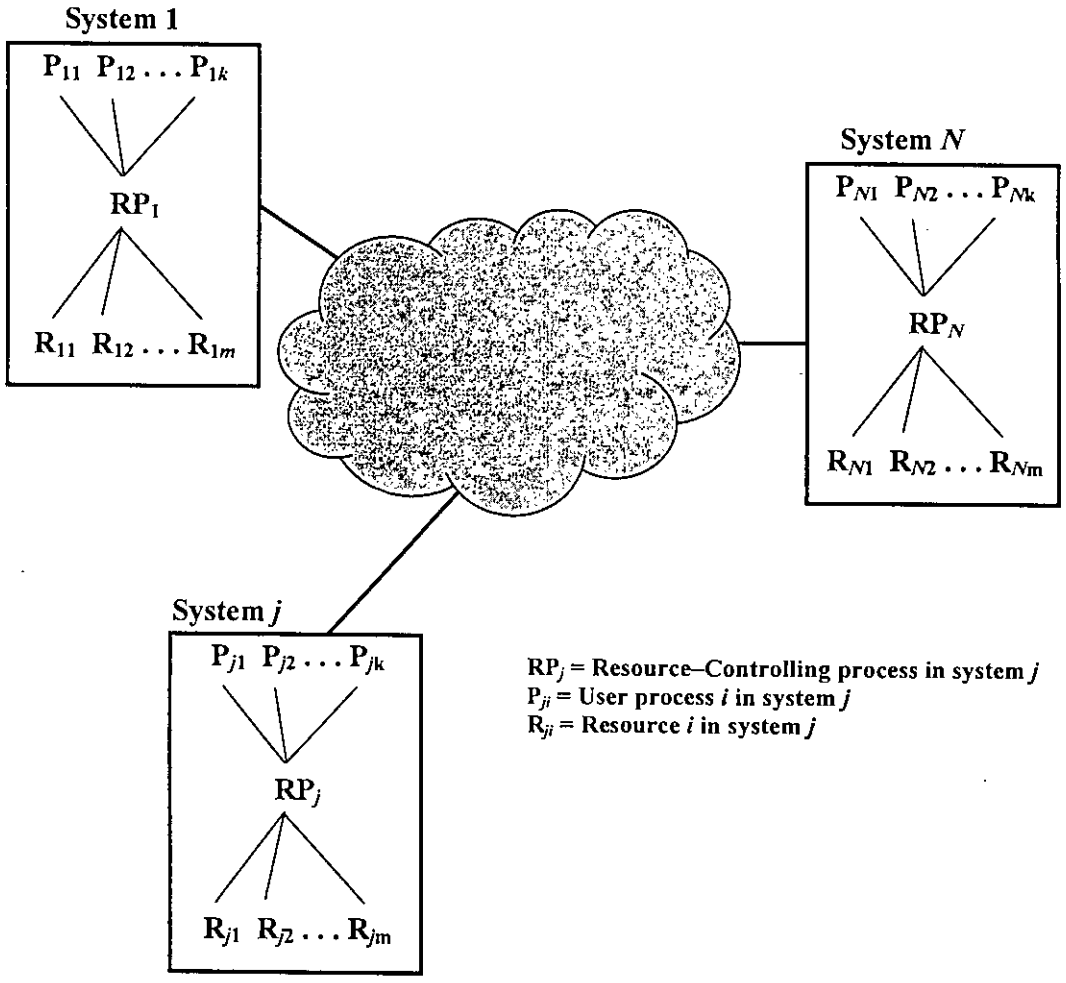


Figure 1.4. Model Mutual Exclusion Problem in Distributed Process Management.

Following properties must be ensured by any solution to *ℓ–Exclusion* problem [1]:

Definition 1.1. (*ℓ–Exclusion*): *No more than ℓ processes are ever concurrently in their critical sections.*

Definition 1.2. (*ℓ*–Lockout Avoidance): *If fewer than ℓ processes are faulty, any requesting process that is not faulty enters its critical section in a finite time.*

In the context of mobile ad hoc networks, node movements may result in frequent and unpredictable topological changes. Complications may arise due to sudden crash or merge of the mobile nodes. Complications get extended since node failures or crashes

are not detectable. Hence, any solution to *ℓ–Exclusion* problem for mobile ad hoc networks must tolerate such issues. On the contrary, it is also important to realize that no solution to the *ℓ–Exclusion* problem for mobile ad hoc networks ever succeeds if topology changes very rapidly (i.e. if node moves, fails, or merges very frequently). Considering dilemmas associated with mobile ad hoc networks, we refine the *ℓ–Exclusion* problem for mobile ad hoc networks as: *Given an ad hoc network of mobile nodes each with a priority index, any connected component of the network whose topology remains static sufficiently long does not enter a global state in which more than ℓ nodes are in their CS.* We refine the properties that must be ensured by any solution to the *ℓ–Exclusion* problem in MANET as follows:

Definition 1.3. (*ℓ–Exclusion*): *No more than ℓ mobile nodes are ever concurrently in their critical sections.*

Definition 1.4. (*ℓ*–Lockout Avoidance): *Let there be fewer than ℓ mobile nodes in failed or crashed state. Any mobile node succeeds to enter in a finite time.*

Note that if *ℓ* or more nodes remain failed or crashed, the *ℓ–Exclusion* condition requires that no other node enters its critical section. Hence, the definition of *ℓ*–Lockout Avoidance only requires progress when less than *ℓ* nodes remain failed or crashed. However, *ℓ*–Lockout Avoidance states that all requests for entry to the CS are eventually satisfied. For static networks, Hadzilacos [57] notes that it may be desirable to ensure a stronger fairness property, i.e. Requests for entry to the CS are satisfied in the order in which they are made. Hence, we refine the First–Come–First–Served (FCFS) property, first introduced by Lamport [80] for mutual exclusion in static networks, as the First–Come–First–Guaranteed property for *ℓ–Exclusion* in mobile ad hoc networks:

Definition 1.5. (First–Come–First–Guaranteed): *If any mobile node requests to enter the critical section earlier than the other, then available entry to the critical section is first guaranteed for the node requested earlier.*

1.3. Related Work

The problem of mutual exclusion has been extensively studied for static networks in distributed systems. But, most of these algorithms may be inefficient for MANET. However, complications arise in designing algorithm for mutual exclusion on MANET, where topology can potentially change with every node movement, message or token can be lost, communication links can be failed, nodes may be crashed or failed node can be recovered, and network can be partitioned permanently.

In literature, few token–based algorithms exist to implement mutual exclusion specially designed for mobile ad hoc or cellular networks. The solutions can be split into two families: "Requesting Token" [139, 140, 142] and "Circulating Token" [4, 8, 28]. We now present a brief outline of those solutions. Note that we explore the solutions extensively in the following chapter.

In 1993, Badrinath et al. [4] proposed two distributed mutual exclusion algorithms for cellular networks. Both are adaptations for cellular networks of the algorithm proposed by Lamport [83], and Le Lann [87]. These adaptations avoid communicating frequently with hosts and finally reduce considerably the cost (for circulating token) compared to [83, 87]. In 1997, Walter and Kini [142] proposed an algorithm derived from [27, 36, 49, 120]. This algorithm defines a structure mapped on real topology of the network, which is represented by a DAG of token–oriented pointers, maintaining multiple paths leading to the node holding the token. Later on, Walter et al. [139] proposed a revised solution of [142], but used [120] as a basis. In addition to previous assumptions, it assumed that communication channels are FIFO with neither loss nor duplication of messages. Following that, Walter et al. [140] again proposed a token–based solution for multiple–entry mutual exclusion based on [142]. This solution is a generalization of 1–mutual exclusion algorithm presented in [139]. Baldoni et al. [8] presented an algorithm based on a dynamic logical ring and combined the best from "Requesting Token" and "Circulating Token". Under heavy request load the number of hops traversed per CS is very close to an optimal value in this algorithm. Very recently, Malpani et al. [90, 91] presented a parametric algorithm with many variants. All these variants have the same framework, but differ in the selection of the successor. A self–stabilizing algorithm based on [38, 91, 137] has

been proposed by Chen et al. [28]. It requires that the topology should be static while the algorithm is converging.

The reader should note that all the aforementioned algorithms presented for MANET may induce catastrophic recitals, as none of these do consider loss of messages or tokens or communication links, crash failure or sudden recovery of nodes, and partition of networks [96, 97].

1.4. Scope and Focus of the Thesis

This thesis work focuses on the following issues:

- To devise a consensus–based mobility aware algorithm for the *ℓ–Exclusion* problem in mobile ad hoc networks. The algorithm would have the following characteristics:
  - To enter the *ℓ*–entry CS, the algorithm would require mobile nodes to communicate only with their immediate neighbors chosen dynamically,
  - The algorithm would ensure the fairness and the correctness defined in Section 1.2),
  - The algorithm would operate asynchronously to tolerate loss of messages, sudden crash or recovery of mobile nodes, and lossy communication links, as long as the link failures do not partition the communication networks.

- To prove formally the correctness and the fairness properties of the *ℓ–Exclusion* algorithm (such as, *ℓ–Exclusion*, *ℓ*–Lockout Avoidance and First–Come–First–Guaranteed properties defined in Section 1.2).

- To simulate, to analyze, and to evaluate the sensitivity, behavior, and performance of the algorithm using PARSEC [5] considering the following performance indices–
  - Message overhead to enter CS per request,
  - Average waiting time to enter CS per request,
  - Success rate to enter CS per request.

- To present the simulation results as a function of request load, the number of identical resources, crash–merge rate under following variants –
  - Zero, low, and high mobility without node failures/merges,
  - Zero, low, and high mobility with node failures/merges.

1.5. Organization of the Thesis

The thesis is organized in chapters which span various topics related to the background, design, analysis, implementation, and simulation issues of the *ℓ–Exclusion* algorithm. Outlines of different chapters are as follows:
The next chapter provides a precise sketch on mobile ad hoc networks, MAC layer used for constructing a mobile ad hoc network, with an outline of routing, addressing, multicasting and QoS in mobile ad hoc network. This chapter also reviews the solutions to distributed mutual exclusion problem for both static and mobile ad hoc networks. Then a preface on MANET simulators concludes the chapter.

Chapter 3 depicts the system model, assumptions on system, mobile nodes and networks, to devise the *ℓ–Exclusion* Algorithm. Next a brief outline of the algorithm followed by a detailed description along with required data structures has been illustrated. A state diagram and some operational examples of the algorithm have also been presented in this chapter. At the end, the chapter proves the correctness of the proposed algorithm.

In chapter 4, we evaluate the performance and sensitivity of the proposed algorithm in a mobile ad hoc environment by carrying out an extensive simulation. Through our simulation, we consider several performance metrics that can significantly impact the behavior of such an algorithm in different ad hoc settings. We also discuss the performance of the *ℓ–Exclusion* Algorithm compared to the *KRL* algorithm, and demonstrate that the performance of our algorithm is remarkable in several scenarios, particularly under mobility and vulnerability.

Finally, chapter 5 concludes this thesis work with the discourse on major contributions of this research work, followed by some thoughts related to future extensions of this work.

CHAPTER 2

BACKGROUND

2.1. Introduction

This chapter presents a concise introduction on mobile ad hoc networks, followed by an overview of MAC layer that can be used for constructing a mobile ad hoc network, with an outline of routing, addressing, multicasting and QoS in mobile ad hoc network. This chapter also presents a literature review on solutions to distributed mutual exclusion problem for static networks. This chapter concludes an extensive study on the existing solutions of distributed mutual exclusion for mobile ad hoc networks, followed by a brief introduction on MANET simulators.

2.2. Mobile Ad Hoc Networks (MANET)

Mobile ad hoc network (MANET) is a collection of potentially mobile nodes that act as processors to route messages or to forward packet data and to run user applications either over a direct wireless interface, or over a sequence of wireless links including one or more intermediate nodes (See Figure 2.1). Direct or indirect communication between mobile nodes depends on their relative positions and transmission radius. Communication topology may postpone spontaneously with time as the nodes move into and go out of each other's transmission radius. In fact, mobile ad hoc networks have the unique characteristic of being totally independent from any authority or infrastructure, providing great potential for the users. Roughly speaking, two or more users can become a mobile ad hoc network simply by being close enough to meet the radio constraints, without any external intervention.

Applications of MANET include –

- Personal area networking –
  - Cell phone, laptop, ear phone, wrist watch.

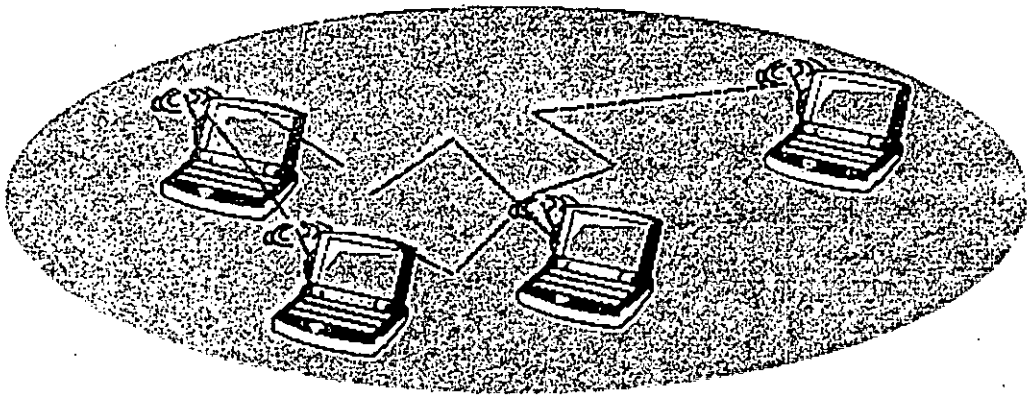


Figure 2.1. A Mobile Ad Hoc Network.

- Military environments –
  - Soldiers, tanks, planes.
- Civilian environments –
  - Taxi cab network.
  - Meeting rooms.
  - Sports stadiums.
  - Boats, small aircraft.
- Emergency operations –
  - Search–and–rescue.
  - Policing and fire fighting.

Characteristics that distinguish mobile ad hoc networks from existing distributed networks include concurrent and unpredictable topology changes due to arbitrary mobility pattern of nodes, shared broadcast channel, dynamic wireless link formation and removal, network partitioning and disconnections, limited bandwidth and energy, location dependent errors, and highly variable message delay. Despite these characteristics, case and speed of deployment, decreased dependence on infrastructure signify mobile ad hoc networks exciting. Moreover, variations in capabilities & responsibilities, variations in traffic characteristics, mobility models, and performance criteria (e.g., throughput, energy, security) increases research activities on mobile ad hoc networks significantly.

Variations of MANET include –

- Fully Symmetric Environment –

- All nodes have identical capabilities and responsibilities.

- Asymmetric Capabilities –
  - Battery life at different nodes may differ.
  - Transmission ranges and radios may differ.
  - Processing capacity may be different at different nodes.
  - Speed of movement.
- Asymmetric Responsibilities –
  - Only some nodes may route packets.
  - Some nodes may act as leaders of nearby nodes (e.g., cluster head).
- Traffic characteristics may differ in different ad hoc networks –
  - Timeliness constraints.
  - Bit rate.
  - Reliability requirements.
  - Unicast / multicast / geocast.
  - Host–based addressing/content–based addressing/capability–based addressing.
- May co–exist (and co–operate) with an infrastructure–based network.
- Mobility patterns may be different –
  - Taxi cabs.
  - People sitting at an airport lounge.
  - Kids playing.
  - Military movements.
  - Personal area network.
- Mobility characteristics –
  - Speed.
  - Predictability –
    - Direction of movement.
    - Pattern of movement.
  - Uniformity (or lack thereof) of mobility characteristics among different nodes.

2.3. MAC Layer

IEEE 802.11 is a digital wireless data transmission standard in the 2.4 GHz ISM band aimed at providing wireless LANs between portable computers and a fixed network infrastructure. This standard defines a physical layer and a MAC layer. Three different technologies are used as an air interface physical layer for contention– based and contention–free access control: infrared, frequency hopping, and direct sequence spread spectrum. The most popular technology is the direct sequence spread spectrum, which can offer a bit rate of up to 11 Mbps in the 2.4 GHz band, and, in the future, up to 54 Mbps in the 5 GHz band. The basic access method in the IEEE 802.11 MAC protocol is the distributed coordination function (DCF) which is a carrier sense multiple access with collision avoidance (CSMA/CA) MAC protocol.

802.11 can be used to implement either an infrastructure–based W–LAN architecture or an ad hoc W–LAN architecture (see Figure 2.2). In an infrastructure–based network, there is a centralized controller for each cell, often referred to as an access point. The access point is normally connected to the wired network, thus providing the Internet access to mobile devices. All traffic goes through the access point, even when it is sent to a destination that belongs to the same cell. Neighboring cells can use different frequencies to avoid interference and increase the cell's capacity. All the cells are linked together to form a single broadcast medium at the LLC layer. A so–called distribution system handles the packet forwarding toward destination devices outside the cell across the wired network infrastructure. The distribution medium that forwards packets among the access points is not defined by the standard. It is possible to use a wireless link to connect the different access points, for example an 802.11 ad hoc link in another frequency. Such a feature permits the implementation of a two–level, multihop architecture.

In the ad hoc mode, every 802.11 device in the same cell, or independent basic service set (IBSS), can directly communicate with every other 802.11 device within the cell, without the intervention of a centralized entity or an infrastructure. In an ad hoc cell, identified by an identification number (IBSSID) that is locally managed, all devices must use a predefined frequency. Due to the flexibility of the CSMA/CA algorithm, it

is sufficient to synchronize devices to a common clock for them to receive or transmit data correctly.

Achieving synchronization is a scanning procedure used by an 802.11 device for joining an existing IBSS. If the scanning procedure does not result in finding any IBSSs, the station may initialize a new IBSS. Synchronization maintenance is implemented via a distributed algorithm, based on the transmission of beacon frames at a known nominal rate, which is performed by all of the members of the IBSS. Additionally, given the constraints on power consumption in mobile networks, 802.11 offers power saving (PS) policies. The policy adopted within an IBSS should be completely distributed for preserving the selforganizing behavior.

The 802.11 standard is an interesting platform to experiment with multihop networking. This standard cannot do multihop networking in the current format. The development of a number of protocols is required.

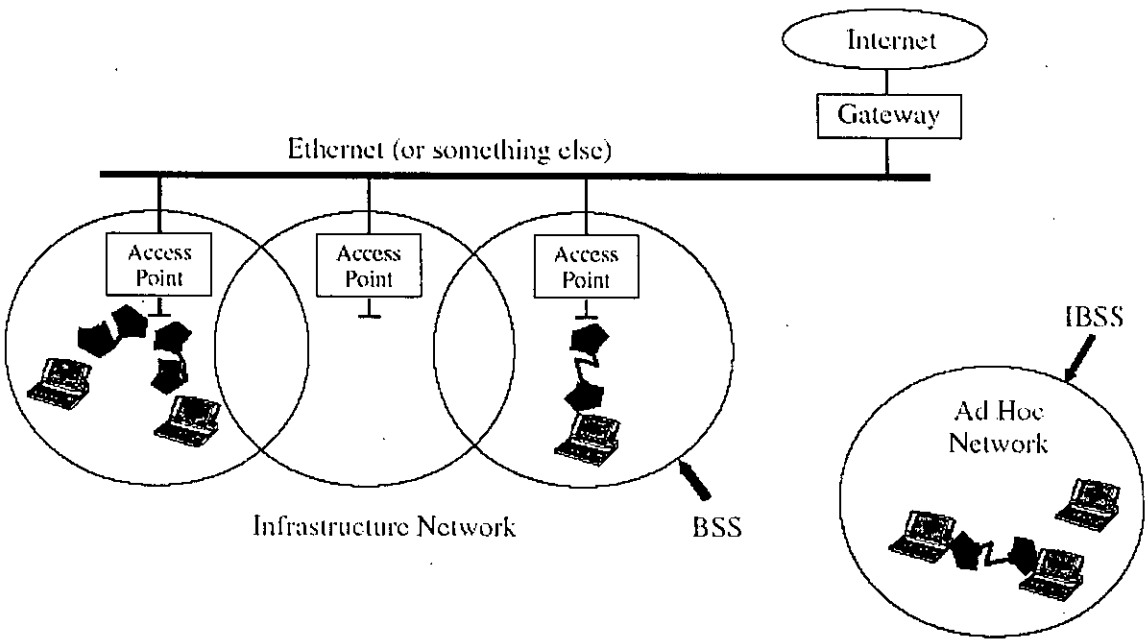


Figure 2.2. Infrastructure and Ad Hoc Networks.

It must be noted that, as illustrated via simulation in [20], depending on the network configuration, the standard protocol can operate very far from the theoretical throughput limit. In particular, it is shown that the distance between the IEEE 802.11 and the analytical bound increases with the number of active networks. In IEEE

802.11 protocol, due to its back off algorithm, the average number of stations that transmit in a slot increases with the number of active participants, and this causes an increase in the collision probability. A significant improvement of the 1EEE 802.11 performance can thus be obtained by controlling the number of stations that transmit in the same slot.

2.4. Routing in MANET

Routing protocols fall into the class of interior gateway protocols, i.e., protocols used to route within a (mobile wireless) network or a set of interconnected (mobile wireless) networks under the same administration authority. Several of these protocols [12, 18, 50, 64, 85, 112, 114, 115] have been implemented in prototypes [95] and, although they are not yet available commercially, some of them are under commercial consideration [47].

In [34], a list of desirable qualitative properties of mobile ad hoc networks routing protocols for the Internet as well as a list of quantitative metrics that can be used to assess the performance of any of these routing protocols has been presented. The properties are distributed operation, loop–freedom, demand–based operation, proactive operation, security, "Sleep" period operation, unidirectional link support. And the metrics are end–to–end data throughput and delay, route acquisition time, percentage out–of–order delivery, efficiency.

Routing protocols are generally categorized according to their method of discovering and maintaining routes between all source–destination pairs, such as proactive protocols, and reactive protocols.

Proactive protocols (also referred to as table–driven protocols) attempt to maintain routes continuously, so that the route is already available when it is needed for forwarding a packet. In such protocols, routing tables are exchanged among neighboring nodes each time a change occurs in the network topology. In contrast, the basic idea of reactive protocols (also referred to as source–initiated protocols) is to send a control message for discovering a route between a given source–destination pair only when necessary.

Most of the existing protocols for mobile ad hoc networks are not univocally proactive or reactive, as some of the protocols have a hybrid proactive and reactive design or simply present elements of both approaches.

The proactive approach is similar to the connectionless approach of traditional datagram networks, which is based on a constant update of the routing information [126]. Maintaining consistent and up–to–date routes between each source–destination pair requires the propagation of a large amount of routing information, whether needed or not. As a consequence, in proactive protocols, a route between any source–destination pair is always available, but such protocols cannot perform properly when the mobility rate in the network is high or when there are a large number of nodes in the network. In fact, the control overhead, in terms of both traffic and power consumption, is a serious limitation in mobile ad hoc networks, in which the bandwidth and power are scarce resources [126]. The proactive approaches are more similar in design to traditional IP routing protocols; thus, they are more likely to retain the behavior features of presently used routing protocols. Existing transport protocols and applications are more likely to operate as designed using proactive routing approaches than on–demand routing approaches [89].

A reactive protocol creates and maintains routes between a source–destination pair only when necessary, in general when requested by the source (on–demand approach). Therefore, in contrast to the proactive approach, in reactive protocols the control overhead is drastically reduced. However, similar to connection–oriented communications, a route is not initially available and this generates a latency period due to the route discovery procedure. The on–demand design is based (1) on the observation that in a dynamic topology routes expire frequently and (2) on the assumption that not all the routes are used at the same time. Therefore, the overhead expended to establish and/or maintain a route between a given source–destination pair will be wasted if the source does not require the route due to topological changes.

As previously stated, some protocols can combine both proactive and reactive characteristics in order to benefit from the short response time provided by the proactive approach under route request and to limit the control overhead as in reactive protocols. An obvious, advantageous approach is to proactively handle all the routes

that are known to be more frequently used and to create on demand all the other routes. Achieving the right balance between reactive and proactive operation in a hybrid approach may require some a priori knowledge of the networking environment or additional mechanisms to adaptively control the mode of operation [89]. A general comparison of the two protocols categories is presented in both [89] and [126].

2.5. Addressing in MANET

One critical aspect of Internet–based mobile ad hoc networks is the addressing. In fact, the addressing approach used in wired networks, as well as its adaptation for mobile IP [116], do not work properly. Approaches based on fixed addressing cannot reflect the topological network for mobile networks. The approach used in mobile IP networks is based on home use, and it is not suitable for ad hoc networks in which there is no fixed infrastructure. Therefore, a new addressing approach for such networks is required. Moreover, given that, in the foreseen topology of the addressing approach, the interaction among different routing protocols could easily happen, a common addressing approach is necessary. This issue is still a matter of ongoing research. The IETF document describing Internet mobile ad hoc networks states that [34]: The development of such an approach is underway, which permits routing through a multi–technology fabric, permits multiple hosts per router and ensures long–term interoperability through adherence to the IP addressing architecture. Supporting these features appears only to require identifying host and router interfaces with IP addresses, identifying a router with a separate Router ID, and permitting routers to have multiple wired and wireless interfaces.

Geographical location of nodes, i.e., node coordinates in two– or three–dimensional space, has been suggested, among other purposes, for simplifying the addressing issue in combination with the Internet addressing scheme. The existing location–based routing protocols propose to use location information for reducing the propagation of control messages, thus reducing the intermediate system functions or for making packet–forwarding decisions.

Geographical routing allows nodes in the network to be nearly stateless; the only information that nodes in the network have to maintain is about their one–hop neighbors.

There are also solutions that do not rely on Internet addresses. A solution in where each node has a permanent, unique end system identifier and a temporary, location–dependent address is proposed in [62]. The location–dependent addresses management, which is based on the association of each end–system identifier to an area of geographical coordinates that acts as a distributed location database. It allows a node to obtain a probable location of any other node with a known end system identifier [61, 88].

The work proposed in the context of Internet mobile routing considers networks traditionally classified as small networks. However, even in networks of one hundred nodes, scalability is an important performance. One approach for achieving scalability in mobile ad hoc networks is clustering. With this approach, the network is partitioned into subsets (clusters). Within a cluster, a traditional MANET algorithm is assumed, and the communication between clusters is done by means of clusterheads and border nodes. The clusterheads form a dominant set that works as backbone for the network. In cluster–based algorithms, one of the main issues is the determination of the clusters and, consequently, of the clusterheads in such a way that the reconfigurations of the network topology are minimized. However, choosing clusterheads optimally is an NP–hard problem [11]. Other strategies for dominating sets for ad hoc networks, which are able to build better performing dominating sets than clustering, are described in [144].

2.6. Multicasting in MANET

Multicast routing is a strategy that could allow optimization of resource usage; this is seen to be as an important feature for energy and bandwidth constrained networks, such as mobile ad hoc networks. Additionally, the underlying layer has a broadcast nature that can be exploited by an integrated design, as done, for example, in [143]. Several multicast routing algorithms have been proposed and evaluated [86]. Although there is the conviction that multicast mobile routing technology is a

relatively immature technology area and much of what is developed for unicast mobile routing—if proven effective—can be extended to develop multicast mobile routing variants [89], some work has assessed the definition of network protocols with an integrated approach, thereby permitting improved energy efficiency [143]. However, shortest–path–based multicast algorithms require too much power (quadratic network size), because they require that each node maintain a global view of the network. Real power savings can be obtained only with localized algorithms in which nodes only know their neighbors' information and make decisions based only on that [131].

2.7. QoS in MANET

Mobile ad hoc networks do not provide QoS by design. Although there has been controversial discussion about whether or not QoS mechanisms are needed in the Internet, it is indisputable that some applications can be supported by wireless networks only under QoS provisioning. Architecture for supporting QoS in mobile ad hoc networks should have two primary attributes [35]:

- Flexibility. Necessary for the heterogeneity of the physical and MAC layers, as well as multiple routing protocols.
- Efficiency. Necessary for the limited processing power and storage capabilities of nodes, as well as the scarce bandwidth available.

In the literature, several approaches have been adopted for assessing the issue of QoS in mobile ad hoc networks, such as the QoS routing protocol [29, 59, 119], and signaling systems for resource reservation [85]. Although some work has been done to date, more studies need to be conducted to further explore the problem of QoS provisioning for mobile ad hoc networks.

2.8. Mutual Exclusion Protocols for Fixed Network

The ℓ–*Exclusion* problem (k–mutual exclusion in some articles) is an extension of mutual exclusion (where $\ell = 1$), a classic problem in concurrency control [39, 84]. This extension was first defined and solved by Fischer et al. [45, 46]. A solution is required to withstand the slowdown or even the crash of processes (up to ℓ–1 of

them), and should not require the active collaboration of processes that are not requesting a resource at that time.

Problem of mutual exclusion has been extensively studied in distributed systems [13]. Mutual exclusion solutions that have been proposed for fixed networks can be classified in two types: centralized approach in which a node is designated as coordinator to deliver permission to the other nodes to access their CS, and the distributed approach in which the permission is obtained from consensus among all network nodes. Due to the symmetry role of nodes and the characteristics of the networks, the first approach is not suitable for mobile ad hoc networks.
The distributed mutual exclusion algorithms are mainly classified in two categories: Permission based [2, 15, 17, 24, 58, 92, 94, 102, 124, 132] and token based [9, 16, 25, 32, 36, 98, 99, 100, 104, 104–107, 120, 122, 127]. Permission based mutual exclusion algorithms impose that a requesting node is required to receive permissions from other nodes (a set of nodes or all other nodes). In the token based algorithms, a unique token is shared among the set of nodes. The node holding the token is allowed to enter its critical section. We now present a literature review of mutual exclusion protocols for static networks, but we limit our discussion only within several distinguished ones.

In 1981, Ricart and Agrawala proposed a distributed algorithm [124] which needs $2\times(N-1)$ messages for a node to enter the critical section. When a node wants to enter the critical section, it broadcasts a request message to all other nodes. On receiving a request message, a node sends back a reply message if it does not want to enter the critical section; otherwise it may defer sending the reply message. Logical timestamps [79] are attached to request messages for nodes to decide whether they should defer replying or not. Only the node whose timestamp is earlier than that of the received request message should defer replying. When a node receives reply messages from all other $N-1$ nodes, it may then enter the critical section. Although this algorithm is deadlock–free and starvation–free, it is vulnerable to node and communication failures and is expensive in communication cost because it requires a node to communicate with all other nodes to enter the critical section.

In 1986, Lamport [80] observed that a first–come, first–served mutual exclusion algorithm can be constructed by preceding a mutual exclusion mechanism with a

FIFO queue. The possibility of faults precludes the straightforward use of any such first–come, first–served mechanism—the first process to enter the mechanism may fail, preventing progress by those following [40, 46]. To resolve this dilemma, the notion of process enabling was introduced: a process is enabled to enter the critical section if sufficiently many steps of that process carry it into the critical section, independent of the actions of other processes. However, the first–in, first–enabled condition was introduced as the natural fairness condition for *ℓ–Exclusion*, generalizing the first–come, first–served condition for mutual exclusion.

Based on Ricart and Agarwala's algorithm, Raymond [121] proposed a distributed algorithm which allows k nodes to access the critical section simultaneously. This algorithm resembles Ricart and Agarwala's except that only $N–k$ reply messages are sufficient for a node to enter the critical section. So, the lower bound of the communication cost for each entrance of the critical section is $2\times N–k–1$. However, each request message will incur a reply message, and thus $2\times(N–1)$ is the upperbound of the communication cost. Raymond's algorithm also suffers the same drawbacks as Ricart and Agrawala's.

There is a large class of algorithms using the token passing concept for the access control of the critical section. The basic idea of this type of algorithms is simple – a node must own the unique token (named privilege message in some papers) before entering the critical section. So, in the best case, if a node has already owned the token, it can enter the critical section immediately without any communication overhead. Otherwise, a mechanism is needed to locate the token. In Suzuki and Kasami's algorithm [132], a node sends out $N–1$ request messages to all other nodes and waits until the token is received. Raymond [120] utilized a spanning tree of the network to locate the token and showed that the average communication cost is $O(log\ N)$. Singhal [129] tried to reduce the communication cost by using heuristics to locate the token. The degree of fault–tolerance for token–based algorithms is low. If the token is lost, complex token–loss detection and token regeneration algorithms must be executed [103].

Based on Suzuki and Kasami's algorithm, Srimani et al. [130] proposed another distributed algorithm which can allow k nodes to access the critical section

simultaneously. This algorithm uses k tokens and equates possession of any token with the ability to access the CS. This algorithm requires a node seeking access to the CS to send requests to n–1 other nodes in order to gain access to the CS, if it does not already hold a token. This flooding of requests makes it poorly suited to ad hoc low energy demands. Thus, Srimani and Reddy's algorithm has shown low degree of fault–tolerance like Suzuki and Kasami's algorithm. Another solution proposed by Kakugawa et al. [73] is resilient to node failure and/or network partitioning, but, as shown in [73], the degree of resilience (especially in terms of all entries to critical sections being available) is unsatisfactory. Moreover, it incurs a large $O(N)$ message overhead.

The algorithm of [93] uses a single token with a counter to keep track of how many processes are currently in the CS. In [21, 22], Bulgannawar and Vaidya showed that under high load this system behaves like a single token system. Also, it requires all nodes to communicate in certain stages of the algorithm, making it too message inefficient for wireless ad hoc environments. Note that the algorithm presented in [21, 22] uses k separate tokens in the system. Requests for a token follow one of k dynamic spanning trees. As requests traverse a tree from leaf to root, path compression is used to keep the tree as shallow as possible. While nodes request tokens by unique token identifiers, the algorithm allows the substitution of any token to grant access to the CS. However, presence of a complex data struture makes this algorithm not scalable. Like other token–baesd algorithms, this algorithm also suffers from poor fault–resiliency as this does not consider loss of token.

There is another class of algorithms using an elegant concept – quorum – to achieve mutually exclusive access of the critical section. They are usually called quorum–based algorithms. Quorum–based algorithms are resilient to node failures and/or network partitioning and usually have lower communication cost. The basic idea of this type of algorithms is "to collect enough permissions (votes) to form a quorum to enter the critical section".

Mutual exclusion is guaranteed if we can assure that only one quorum can be formed at any instance. Garcia–Molina and Barbara have proposed "coterie" to generalize the concept of quorums [51]. A coterie is a set of sets with the property that any two

members of a coterie have a non–empty intersection. By the intersection property, the members in a coterie can be used as quorums to guarantee mutual exclusion in distributed systems.

The majority quorum algorithm [136], the $\sqrt{N}$ algorithm [92], the tree quorum algorithm [2], and the hierarchical quorum algorithm [79], are all quorum–based algorithms. The communication cost of the quorum–based algorithm is proportional to the quorum size, so all of the quorum–based algorithms try to reduce the quorum size while keeping the high degree of fault–tolerance.

To form a quorum in the majority quorum algorithm requires permissions from over half of the nodes. It is easy to show that any two quorums in the majority quorum algorithm have a non empty intersection and the size of a quorum is $\left\lceil \frac{N+1}{2} \right\rceil$. In the $\sqrt{N}$ algorithm [92], Maekawa used the concept of finite projective plane to assure the intersection property and the fully distributed property – every quorum is of the same size and every node bears an equal amount of responsibility for mutual exclusion control. As the title of the algorithm suggests, the quorum in $\sqrt{N}$ algorithm is of size $\sqrt{N}$.

Some algorithms [2, 79, 128] utilize logical structures to assist in forming quorums. Assuming the system is logically organized into a binary tree, the tree–quorum algorithm [2] can have the size $\lceil \log N \rceil$ in the best case for a quorum. The quorum forming in this algorithm is recursive. It can be regarded as attempting to obtain permissions from nodes along a root–to–leaf path. If the root fails, then the obtaining permissions should follow two paths: one root–to–leaf path on the left subtree and one root–to–leaf path on the right subtree. The largest tree–quorum, which is of size $\left\lceil \frac{N+1}{2} \right\rceil$, is the one comprising all leaf nodes. The hierarchical quorum algorithm uses multilevel tree to aid the quorum forming. The concept is simple – a quorum of a node at level i is formed only if enough (over half) quorums of its child nodes at level i+1 are formed. As shown in [79], the hierarchical quorums may have size $N^{0.63}$. By logically organizing nodes in different levels, the level quorum algorithm [128] tries

to form a quorum by obtaining permissions from all nodes of some level (say level i) and one node of each level before level i. In an extreme case, all nodes in the first level can form a level quorum whose size is a constant independent of N.

To suit to the access control of k ($k > 0$) entries to a critical section, at least two papers [48, 60] discussed the extension from coteries into k–coteries, which allow up to k quorums to be formed simultaneously. The k–coterie concept sustains the advantages of the coterie concept – fault–tolerance and low message cost. But [48] gave a more restrict definition of the k–coterie than [60] did. Adopting the definition of [48], Jiang et al. [67], modified the majority quorum as shown in [73], and proposed a structure Cohorts for the construction of Cohorts quorums – quorums in a k–coterie, where permissions from $\left\lceil \frac{N+1}{k+1} \right\rceil$ nodes form a quorum. Cohorts quorums have size $O(N)$.

However, abovementioned permission–based algorithms are poorly suited to the energy poor wireless environments since they require communication between each node in networks for each access to CS, and nodes have symmetric characteristics. Whereas each of the distributed, token based algorithms assumes that the network is reliable and fully connected, 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. On the other hand, quorum–based algorithms don't suit ad hoc environment as fixed logical structures are unrealistic in MANET.

2.9. Static Mutual Exclusion Protocols Adapted for MANET

We now recall some algorithms developed in the literature for fixed networks on which recent mutual exclusion algorithms for mobile ad hoc networks are based.

In the token based approach, two methods are usually used circulating token and requesting token. In the requesting token method, a node requesting the CS has to obtain the token. The basic problem is how to reach the token holder. In some algorithms, the request is sent to all the nodes because the token holder is unknown [123], in others a logical structure is defined to point the token holder, for example a

Direct Acyclic Graph [120]: The request is sent over the branch of the DAG which leads to the node holding the token.

In the solution proposed by Le Lann [87], all nodes are logically organized in a unidirectional ring and the token circulates following this ring. When the token is received by a node, it enters its CS if it is requesting, and after executing its CS, it sends the token to its successor in the ring.

Ricart et al. [123] proposed an algorithm which requires at most N messages to achieve mutual exclusion. The requesting node sends the request message to all the other N–1 nodes and waits for responses. When the process holding the token has to send the token, the next process is chosen in a circular manner if any; otherwise, it keeps the token in an idle state. Based on the Ricart–Agrawala's algorithm, Suzuki et al. [132] proposed an algorithm in which the queue of requesting nodes is piggybacked within the token. This queue is updated by a local queue of each visited node in an ascending node number in order to ensure the liveness property. Singhal [129] has improved the performance of the algorithm proposed in [132]. This algorithm requires at most N messages in heavy loads. Singhal used some heuristics to guess what nodes of the system are probably holding or are likely to have the token and sends a token request message only to those nodes rather than to all the nodes. To achieve this, the knowledge of each node about the requesting nodes is passed through the token.

Raymond [120] developed an algorithm, based on a logical tree on the network rooted by the token holder node, which requires at most $O(LogN)$ messages to enter the CS. The tree is maintained by the logical pointers distributed over the nodes and directed to the node holding the token. When a node wants to access its CS, it enqueues its identity and sends a request message to the next node in the direction of the token holder, this message is then routed successively to the token holder. The token is sent back over the reverse path to the requesting node. The direction of the link of the token sending nodes must be reversed to point always the token holder. Chang et al. [27] developed an algorithm which improves Raymond's algorithm which tolerates link and node failures by maintaining multiple paths to search the token. The algorithm tries also to avoid cycles when the token returns to the requester along the

reversal links. Dhamdhere et al. [36] developed an algorithm which aims to resolve the problem of still remaining cycle in the algorithm presented in [27], and it is *k*–resilient, that is it tolerates *k* node/link failures.

2.10. Mutual Exclusion Algorithms for MANET

We now present existing distributed mutual exclusion algorithms developed for mobile ad hoc and cellular networks.

2.10.1. B. R. Badrinath, A. Acharya, and T. Imelinski Algorithm

Badrinath et al. [4] proposed two distributed mutual exclusion algorithms for cellular networks. The first one is an adaptation for cellular networks of the algorithm proposed by Lamport [83], the second one is for cellular networks of the algorithm proposed by Le Lann [87].

The Lamport algorithm is adapted to mobile computing environment by shifting the communication and computation requirement of the algorithm to the static part of the network, i.e. mutual exclusion is achieved with better message complexity because the base station acts as proxy for nodes attached to it. Each host is replaced by its base station which maintains necessary data structures. The base stations exchange time–stamped requests, reply and release between them and ensure mutual exclusion on behalf the hosts. Each host is then limited to send its request to only its base station, receive the grant from only its base station, and sends a release message to its base station when it exits its critical section. If the host disconnects prior to receiving the grant request, the base station releases the resource to another host attached to it. If the host disconnects after receiving the grant request but without sending release resource, it must reconnect to send release resource.

The adaptation of Le Lann's algorithm to a mobile environment is made by arranging the base stations (instead of hosts) in a logical ring and each base holds a FIFO queue which contains the requests of hosts attached to it. The token circulates on the logical ring and when it is received by base station, it is sent to the requesting hosts but to one at a time until the queue is empty (the queue maintained by the base station), and then it is passed to its successor in the ring. To ensure that a host enters its CS at most once

during one tour (and finally to ensure fairness), a counter of rounds is included in the token message.

2.10.2. J. E. Walter and S. Kini Algorithm

J. Walter et al. [142] proposed a token based mutual exclusion algorithm derived from several other algorithms: the tree based routing protocol of Gafni et al. [49], the algorithm presented in [27] and others ideas from [36, 120]. This algorithm defines a structure mapped on real topology of the network which is represented by a Direct Acyclic Graph (DAG) of token–oriented pointers, maintaining multiple paths leading to the node holding the token. The algorithm is well suited to the distributed mobile setting because it requires nodes to keep information only about their immediate neighbors. In the absence of node's movement, the algorithm acts as Raymond's algorithm [120] to forward requests to the node holding the token and sends back the token to the requesting node. Moreover, each node keeps an elevation representing the height of a node. Links are directed from nodes with higher height towards nodes with lower height. This elevation is used to update the DAG structure in case of link failure in order to have always the token holder node as a root of this tree. This algorithm assumes that failures occur only due to node movement. It is assumed that the token cannot be lost and communication links are bidirectional. The nodes move with a limited speed so it can not disconnect from the network during activation of the algorithm and during the message transmission.

2.10.3. J. E. Walter, J. L. Welch, and N. H. Vaidya Algorithm

J. Walter et al. [139] again proposed an alternative solution of the algorithm described in [142], but uses the Raymond algorithm [120] as a basis. The same assumptions as the previous algorithm are done. In addition to that, it is assumed that communication channels are FIFO with no loss and no duplication of messages. The proposed algorithm uses a structure mapped on physical topology of the network which is represented by a direct acyclic graph (DAG) of token–oriented pointers, maintaining multiple paths leading to the node holding the token. Nodes keep information only about their immediate neighbors. Each node dynamically chooses its neighbor with lowest elevation as its preferred next link to the token holder. When a link fails, the concerned node reroutes its request to another path. All requests reaching the token

holder are treated symmetrically. However, requests are forwarded to the token holder over the tree, as in [142]. The token is delivered over the reverse tree path to the requesting node. The token holder will always be the lowest node in the DAG. Sometimes, links may fail and may be created.

The reader should note that the main difference between this algorithm and the algorithm presented in [142] is that in this algorithm when the process requires tosend a token, it finds necessary the return path (because no partitioning of the network occurs). So, the failure is treated by the requesting node (because each node is aware of its neighboring nodes). This eliminates the overhead introduced by the search process. It can be noted that the partial rearrangement of the DAG may be necessary when the token circulates or when links may be created or failed. We note also that this algorithm uses less message types than in [142].

The algorithm proposed by Baldoni et al. [8] is based on a dynamic logical ring and combines the two methods token–asking and circulating token. The algorithm aims to maintain device power consumption as low as possible by reducing the number of hops traversed per CS execution and by avoiding sending any control message when no process requests the CS. Mobility is addressed by exploiting the information of the routing table in order to send each message to the closest node in terms of number of hops. In this algorithm, authors assume that processes do not crash and the network is reliable. The network is not subject to permanent partitions. If a partition occurs, each pair of processes will be eventually able to communicate, and finally, each process may query at each time the information provided by the routing protocol. And the disconnected processes are considered to be at infinite distance, and they are treated with waits and tries to transmit.

The algorithm continuously executes transitions alternately between two states: Idle and Coordinator–Change and is executed in rounds. For each round, that is materialized by circulating a token over all nodes, one process is designated to be a coordinator to which processes have to send requests. The round is completed when all nodes of the network are visited. The round is aimed to inform each process of the ring on the new coordinator and to allow the receiving and requesting one to enter the CS.

2.10.4. N. Malpani, Y. Chen, N. H. Vaidya, and J. L. Welch Algorithm

N. Malpani et al. [90, 91] proposed a parametric algorithm with many variants. It uses a logical dynamic/variable size ring (on which the token circulates continuously through all the nodes of the network): the size of the ring may vary at every round (in which all nodes must be visited). Main idea of the algorithm resides in some variants that are distinguished by the policies applied to determine the next node to which the token will be sent. They are divided into two classes, those using only local neighborhood information (local algorithms), and those that uses information about all the network nodes (global algorithms). In order to protect against the potential loss of the token, the proposed algorithm uses TCP connection to deliver the token. For mobility, with some variants, nodes use "hello" messages to discover if neighbors remain connected to them. In each variant of the algorithm, the token carries with it some "count" information for each node in the system. The recipient node, before sending the token, uses the carried information to choose the next node to which it has to send the token (most of the variants chooses the next recipient, among those allowed, with the smallest value) and to update its own count information piggybacked by the token. The different methods are:

- Local–Frequency (LF) variant: The token is sent to the least frequently visited neighbor of the token–holder node.
- Local–Recency (LR) variant: The least recently visited neighbor is chosen.
- Global variants (GR and GF): The token is sent to the node that has been visited the least recently (in GR) or least frequently (in GF) among all the nodes of the network.
- Global variants with Next (GRN and GFN): All intermediate nodes in the route chosen for the destination are visited by the token.

2.10.5. Y. Chen and J. L. Welch Algorithm

Y. Chen et al. [28] proposed a self–stabilizing mutual exclusion algorithm for mobile ad hoc networks. It is based on the LRV (least recently visited) algorithm presented in [90, 91], which presents good results in simulation, and on the self–stabilizing concept defined by Dijkstra [38] and on the idea of "counter flushing" of [137]. The algorithm uses dynamic virtual rings to reflect the changing topology formed by

circulating tokens. It requires that the topology to be static while the algorithm is converging.

2.10.6. J. E. Walter, G. Cao, and M. Mohanty Algorithm

A token based k–mutual exclusion (ℓ–*Exclusion*) algorithm has been presented in [140, 141]. The algorithm induces a directed acyclic graph (DAG) on the network, dynamically modifying the logical structure to adapt to the changing physical topology in the ad hoc environment. The algorithm combines ideas from many papers. The partial reversal technique from [49], used to maintain a destination oriented DAG in a packet radio network when the destination is static, is used in this algorithm to maintain a token oriented DAG with k dynamic destinations. Like the algorithms of [27, 36, 120], each processor in this algorithm maintains a request queue containing the identifiers of neighboring processors from which it has received requests for the token. Like [36], this algorithm totally orders nodes. Based on [21, 22], the algorithm maintains k tokens in the system. When $k = 1$, the lowest node is always the current token holder, making it a "sink" toward which all requests are sent. When $k > 1$, there may be multiple sinks in the system. However, we show that all non token holding processors will always have a path to some token holding processor.

2.11. Discussions

Due to the differences on the approaches adopted by the existing protocols for MANET, and the lack of methodology in complexity analysis for some of them, it is difficult to address a simple evaluation on these protocols. So, we limit our discussion to give some remarks about the presented solutions.

In [142], the proposed algorithm considers that node mobility is slow which is not realistic. On the other hand, it does not consider token loss and network partitioning and merging. In [139], the proposed algorithm acts with the same assumptions as [142] and also uses fewer messages which lead to a comprehensive execution. The simulation shows that the performance of the algorithm is comparable with those of Raymond's algorithm. The other algorithms [8, 90, 91] are not aware of the node mobility because of the use of the routing layer and consider that the network is not subject to partitions. Algorithms in [140, 141] require each node to spend a vast data

structures to operate. Moreover, operational complexities are also high in [140, 141]. The algorithm presented in [28] assumes a strongly connected network, but links may change.

Moreover, some of the solutions presented above consider that the links are uniform (bidirectional), but in reality the machines may be from different manufactures and consequently they may have different communication ranges, so the communication channels may be unidirectional. So, the solutions in [139, 142] do not work in this case.

It is important to realize that all of the above algorithms are token–based, and token–based algorithms have poor failure resiliency [2]. Loss of the token (which can be because a node which has the token crashes and does not remember that it had the token when it recovers) is serious. All token–based mutual exclusion algorithms have these problems for the following reasons [129]:

- Declaration of the token loss requires special care because if the declaration is false and an additional token is generated, then the condition of mutual exclusion will be violated.
- When the token is declared to be lost, exactly one node should generate the token.

In MANET, monitoring and regeneration of token may incur further complications. Therefore, these algorithms may induce catastrophic recitals as none of these do consider loss of regeneration of messages or tokens, crash failure or sudden recovery of nodes, and partition of networks [96, 97].

2.12. MANET Simulators

This section gives a brief overview of simulators used for MANET. Its aim is to summarize the different implementation approaches of each simulator.

2.12.1. OPNET Modeler

OPNET Modeler [110] is a powerful network simulator developed by OPNET. It can simulate all kinds of wired networks, and an 802.11 compliant MAC layer

implementation is also provided. Although OPNET is rather intended for companies to diagnose or reorganize their network, it is possible to implement one's own algorithm by reusing a lot of existing components. Most part of the deployment is made through a hierarchical graphic user interface. Basically, the deployment process goes through the following phases. Implementation and simulation under OPNET Modeler consists of the following steps:

Step–1. To create and configure the node models (i.e. types) and, on top of it, an application process user requires to use in the simulations – for example a wireless node, a workstation, a firewall, a router, a web server, etc. A process model is described as a state machine. Each state can have some code that is executed when it gets active. A transition that links two states is followed whenever a certain condition carried by the transition is true.

Step–2. To build and to organize user's network by connecting the different entities.

Step–3. To select the statistics that user requires collecting during the simulations.

The difficulty with OPNET Modeler is to build the state machine for each level of the protocol. It can be pretty difficult to abstract such a state machine starting from a pseudo–coded algorithm. But anyway, state machines are the most practical input for discrete simulators. In summary, it is possible to reuse a lot of existing components (MAC layer, transceivers, links, etc.) improving the deployment process. But on the other hand, any new feature must be described as a finite state machine which can be difficult to debug, extend and validate.

2.12.2. NS–2

NS–2 [108] is a discrete event network simulator that has begun in 1989 as a variant of the REAL network simulator [135]. Initially intended for wired networks, the Monarch Group at CMU have extended NS–2 to support wireless networking such as MANET and wireless LANs as well [133]. Most MANET routing protocols are available for NS–2, as well as an 802.11 MAC layer implementation [56]. NS–2's code source is split between C++ for its core engine and OTcl, an object oriented version of TCL for configuration and simulation scripts. The combination of the two languages offers an interesting compromise between performance and ease of use. Implementation and simulation under NS–2 consists of 4 steps:

Step–1. Implementing the protocol by adding a combination of C++ and OTcl code to NS–2's source base;

Step–2. Describing the simulation in an OTcl script;

Step–3. Running the simulation;

Step–4. Analyzing the generated trace files.

Implementing a new protocol in NS–2 typically requires adding C++ code for the protocol's functionality, as well as updating key NS–2 OTcl configuration files to recognize the new protocol and its default parameters. The C++ code also describes which parameters and methods are to be made available for OTcl scripting. The NS–2 architecture follows the OSI model closely. An agent in NS–2 terminology represents an endpoint where network packets are constructed, processed or consumed.

Some disadvantages of NS–2 stem from its open source nature. First, documentation is often limited and out of date with the current release of the simulator. Fortunately most problems may be solved by consulting the highly dynamic newsgroups and browsing the source code. Then code consistency is lacking at times in the code base and across releases. Finally, there is a lack of tools to describe simulation scenarios and analyze or visualize simulation trace files. These tools are often written with scripting languages. The lack of generalized analysis tools may lead to different people measuring different values for the same metric names. The learning curve for NS–2 is steep and debugging is difficult due to the dual C++/OTcl nature of the simulator. A more troublesome limitation of NS–2 is its large memory footprint and its lack of scalability as soon as simulations of a few hundred to a few thousand of nodes are undertaken.

2.12.3. PARSEC

PARSEC [5, 101] (for PARallel Simulation Environment for Complex systems) is a C–based discrete–event simulation language. It adopts the process interaction approach to discrete–event simulation. An object (also referred to as a physical process) or set of objects in the physical system is represented by a logical process. Interactions among physical processes (events) are modeled by timestamped message exchanges among the corresponding logical processes. One of the important

distinguishing features of PARSEC is its ability to execute a discrete–event simulation model using several different asynchronous parallel simulation protocols on a variety of parallel architectures. PARSEC is designed to cleanly separate the description of a simulation model from the underlying simulation protocol, sequential or parallel, used to execute it. Thus, with few modifications, a PARSEC program may be executed using the traditional sequential (Global Event List) simulation protocol or one of many parallel optimistic or conservative protocols. In addition, PARSEC provides powerful message receiving constructs that result in shorter and more natural simulation programs. Useful debugging facilities are available. The PARSEC language is derived from Maisie [6], but with several improvements, both in the syntax of the language and in its execution environment.

2.12.4. GloMoSim

GloMoSim [7, 145] is a scalable simulation environment for wireless and wired networks systems developed initially at UCLA Computing Laboratory. It is designed using the parallel discrete–event simulation capability provided by PARSEC [5, 101]. GloMoSim currently supports protocols for purely wireless networks. It is build using a layered approach. Standard APIs are used between the different layers. This allows the rapid integration of models developed at different layers by users. To specify the network characteristics, the user has to define specific scenarios in text configuration files: app.conf and Config.in. The first contains the description of the traffic to generate (application type, bit rate, etc.) and the second contains the description of the remainder parameters. The statistics collected can be either textual or graphical. In addition, GloMoSim provides various applications (CBR, ftp, telnet), transport protocols (TCP, UDP), routing protocols (AODV, flooding) and mobility schemes (random waypoint, random drunken).

With GloMoSim, the difficulty is to describe a simple application that bypasses most OSI layers. The bypass of the protocol stack is not obvious to achieve as most applications usually lie on top of it. Compared to OPNET, for example, GloMoSim architecture is much less flexible.

2.13. Chapter Summary

In this chapter we have briefly explored mobile ad hoc networks, MAC layer that can be used for constructing a mobile ad hoc network, with an outline of routing, addressing, multicasting and QoS in mobile ad hoc network. We have also presented a literature review on solutions to distributed mutual exclusion problem for both static and mobile ad hoc networks. This chapter has been concluded by a brief discussion on MANET simulators.

CHAPTER 3

THE *ℓ–EXCLUSION* ALGORITHM

3.1. Introduction

This chapter explores the system model, assumptions on system, mobile nodes and networks, to design the proposed *ℓ–Exclusion* Algorithm. We present a brief outline of the proposed algorithm, followed by detailed description along with required data structures. A state diagram of the proposed algorithm followed by operational examples of the algorithm has also been demonstrated in this chapter. And the chapter ends with proofs of correctness to establish the validity of the proposed algorithm.

3.2. System Model and Assumptions

We consider a distributed system composed by a set of $n \geq 1$ processes p_i (where $1 \leq i \leq n$) running each on different mobile nodes m_i (where $1 \leq i \leq n$) that follows an arbitrary mobility pattern, and exists in the network for a sufficiently long period. We represent the system as a directed graph G, in which every vertex v_i represents a mobile node m_i and there is a directed link from v_i to v_j if and only if mobile node m_j is within transmission range of m_i. Our *ℓ–Exclusion* Algorithm borrows some ideas for assumptions from [8, 28, 101, 129, 138–142]. Assumptions on system, mobile nodes and networks are:

- G is a message–passing asynchronous system with an upper bound d on the point–to–point message delay.
- Each node has unique integer identifier that is fixed throughout the node's lifetime. Identifiers are used to identify participants during the *ℓ–Exclusion* process. They are also used to break ties among nodes which have the same requesting timestamp.

- Each node stays in the network for a sufficiently long period, i.e. each node has a minimum lifetime. Because, no distributed algorithm for mobile ad hoc networks can succeed, if node failures occur very repeatedly.
- Each node is aware of the (possibly changing) set of neighbors with which it can currently communicate directly by providing indications of link formation and failures.
- Communication links are reliable and FIFO.
- Message propagation delay is finite but unpredictable.
- Message delivery is guaranteed only when the sender and the receiver remain connected (not partitioned) for the entire duration of message transfer.
- Messages are received by an entity in their timestamp order. Messages of a single type with the same timestamp from a common source are received in the order they are sent; however no a priori ordering can be assumed for messages with the same timestamp received from multiple sources.
- We assume that an underlying routing protocol exists to deliver messages between two nodes via a separate secure channel, so that other nodes don't even detect a packet transmission not intended for them. The reason is that transmissions and receptions consume energy, and if it requires many transmissions and receptions, it may not be practical for energy–limited networks.
- Nodes can crash/recover arbitrarily at any time. Moreover, nodes holding message of other node or originator (node) of message may fail. Mobile nodes use timeout (a designated time threshold, or an upper bound on message expiry) to detect message losses. While a node holding request message of other node fails, after timeout, originator (node) of message presumes that message expires and it re–originates that message. And when originator (node) of message fails, instead returning back to the originator, message expires after the designated time threshold (timeout).
- One or multiple concurrent changes (either a link failure or a link formation) may occur at a time, i.e. we do not require the next change (after one link change) to occur after the entire network has recovered from the previous change.
- After a single or multiple concurrent link changes, topology remains static for a sufficiently long period. A substantial *ℓ–Exclusion* algorithm requires this

assumption as no distributed algorithm for mobile ad hoc networks can succeed, if topology changes very frequently.

- Any node not in CS may have at most one outstanding request for CS. We consider that a requesting node waits until its last pending request succeeds or expires.
- Each node, executing CS, eventually releases CS. We also presume that no slow or crashed node stays forever in CS. We consider a node recovered from crash failure resets, i.e. a crashed node never exists in CS.
- Partitions of the network do not occur. We consider link failures as long as the link failures do not partition the communication networks.
- Each node has a sufficiently large receive buffer to avoid buffer overflow at any point in its lifetime.
- We impose neither restrictions on the speed and direction of node movements, nor on the number of participating nodes.

3.3. The *ℓ–EXCLUSION* (LE) Algorithm

We now present the brief outline of the proposed algorithm followed by a detailed description of the pseudocode along with a state diagram of the proposed algorithm.

3.3.1. Brief Outline

The algorithm employs diffusing computations [37] to initialize a mechanism to collect enough consensuses for a mobile node intending to enter CS. We refer to this computations–initiating node as the requesting node. Each requesting node requires only one message (viz. *Request*) to enter CS. The algorithm defines a mechanism for a requesting node by circulating its *Request* message among other nodes of the MANET. Final decision is based on the consensus of these nodes. Elaborately, the algorithm considers two stages of *Request* message, i.e. *Request* message can be originated from a requesting node, or *Request* message of one requesting node may arrive to another mobile node. When a node requires entering CS, it generates a *Request* message and forwards to its nearest neighbor. On receipt of a *Request* message, a node forwards the message to its least recently visited (LRV) neighbor. This forwarding mechanism is repeated in a recursive fashion. However, *Request* is eventually forwarded to its originator node. On receipt of own *Request*, the originator node takes one of the following actions –

- To enter CS (if one or more entries is/are free), or
- To discard *Request* and originate *Request* again (if no entry to CS is available and all nodes are visited by *Request*), or
- To forward *Request* to the unvisited alive nodes (if no entry to CS is available and there are still unvisited nodes). Note that this forwarding aid *Request* to collect more permission from unvisited nodes, as entry to CS is not yet available.

However, complications arise when two or more requesting nodes originate their own *Request*s concurrently to enter CS. Complications are extrapolated when the number of *Request*s is larger than that of available ℓ identical copies of resource. Moreover, communication links may change due to unpredictable mobility pattern of nodes, and/or nodes can crash or merge or recover from failure during the *ℓ–Exclusion* process. Therefore, we extend our *ℓ–Exclusion* protocol to serve several simultaneous *Request*s, to tolerate dynamic change of links, and to allow at most ℓ–1 nodes to fail concurrently.

3.3.2. Data Structures

We now sketch the data structures required for mobile nodes and message employed by our algorithm. Each node m_i maintains the following data structures:

- *LastRequestID*: *integer*; indicates the identifier of the last request (for CS) sent by the node.
- *LastRequestTS*: *time*; indicates the local timestamp of the last request (for CS) sent by the node.
- *RequestQueue*: *FIFO queue*; holds the incoming messages.
- *TS_of_Racer*: *array* [*1...n*] *of time*; holds timestamp of request arrival from *n_node*s. Detail of *n_node* will be explored in the next section.

Each node m_i sends the following message(s) to adjacent node(s):

- *Request* (*ProcessID*, *RequestID*, *RequestTS*, *Time*, *Visited*, *Next*, *ℓ_Count*): sent by a node m_i for requesting CS and forwarded by nodes m_j's ($i \neq j$) to their neighbor; where,
  - *ProcessID*: *integer*; indicates the identifier of the node requesting for CS.

- *RequestID*: *integer*; indicates the identifier of the request message.
- *RequestTS*: *time*; indicates the local timestamp when the request message is originated.
- *RecipientID*: *integer*; indicates the identifier of the recipient node.
- *Time*: *integer*; indicates a counter incremented by m_i on receipt of the request message.
- *Visited*: *array* [*1...n*] *of integer*; *Visited*[*i*] is updated by node m_i using *Time* variable of the request message. Any node m_k uses *Visited*[ ], hold by its *Request* to select its least recently visited (LRV) neighbor, and to recognize μ_*node*s. Detail of μ_*node* will be explored in the next section.
- *Next*: *array* [*1...n*] *of boolean*; *Next*[*j*] ← TRUE, set by m_i, if m_j's are μ_*node*s. Any node m_k uses *Next*[ ], hold by its *Request* to identify μ_*node*s.
- ℓ_*Count*: *integer*; a counter incremented by each m_i executing CS. Any node m_k uses this counter hold by its own *Request* to enter CS or not.

3.3.3. Pseudocode

This section presents the formal pseudocode for *ℓ–Exclusion* algorithm (see Figure 3.1, and Figure 3.2) followed by detailed explanation. Throughout this section, subscripts on data structures (to indicate a particular node) are only included when needed. For example, we denote $Request_{index}$ as a request message of any mobile node m_{index} to enter CS; and $Request_{index}'$ as a request message (of any mobile node m_{index}) that has been already served in the earlier steps; and ℓ_Count_{index} as the counter associated with $Request_{index}$.

We split the operation of this algorithm into two procedures. We present a brief outline of those procedures in the following:

Forwarding Request for CS: When a node requires entering CS, it calls the procedure in Fig. 1. The node initializes data structures associated with *Request* (in line 1–11). Then the node recognizes the possible candidates for μ_*node*s. Here, we define μ_*node* (for a particular $Request_k$) as the unvisited node m_j that is directly reachable from a node m_i holding $Request_k$ (where $i \neq j$), but is not selected as immediate

neighbor of m_i to send/forward $Request_k$ from m_i. Now, *Next*[] is set to TRUE for all μ_nodes (in line 12), and sends *Request* to selected immediate neighbor (in line 14).

Procedure *Request_for_CS* ( ):

1. *Request.ProcessID* ← *Current.ProcessID*;
2. **Initialize** *Request.RequestID*;
3. *Current.LastRequestID* ← *Request.RequestID*;
4. *Current.TS_of_Racer*[*i*] ← +∞, where $1 \le i \le n$;
5. **Set** *Request.RequestTS*;
6. *Current.LastRequestTS* ← *Request.RequestTS*;
7. *Request.Time* ← 1;
8. *Request.Visted*[*i*] ← 0, where $1 \le i \le n$;
9. *Request.Next*[*i*] ← FALSE, where $1 \le i \le n$;
10. *Request.Visted*[*Current.ProcessID*] ← *Request.Time*;
11. *Request.*ℓ*_Count* ← 0;
12. *Request.Next*[*j*] ← TRUE, where m_j's are μ_nodes;
13. **Select** *immediate neighbor* m_n;
14. **Send** *Request* **to** m_n;

Figure 3.1. Pseudocode for requesting CS.

Handling Request: Nodes enqueue the received *Request*(s) in the *RequestQueue*, and calls the procedure in Fig. 2. Several cases for handling *Request*(s) are in the following:

Case I: When node m_i receives a $Request_k$, where $i \neq k$, i.e. m_i isnot the originator of $Request_k$. We consider two cases:

Case Ia: If the receiving node m_i is in CS, or if it is a requesting node and timestamp of its $Request_i$ is less than that of incoming $Request_k$, it increments ℓ_Count_k. Therefore, originator of $Request_k$ becomes π_node of m_i for $Request_i$. We define π_node as a requesting node m_j, if ℓ_Count_j is incremented by a requesting node m_i, or by a node m_i executing CS, when $Request_j$ visits m_i. (Note that the algorithm requires π_node for each *Request* to mark the other requesting nodes that are competing to enter CS concurrently with the originator of the *Request*.) If receiving node is a requesting node, and timestamp of its $Request_i$ and that of incoming $Request_k$ are equal, then it increments ℓ_Count_k, if its identifier is smaller than that of incoming *Request* originator.

Case Ib: If originator of $Request_k$ is a μ_node of m_i for $Request_i'$, and m_i is not in CS, ℓ_Count_k is decremented by one.

```
Procedure Handle_Request( ):
15. if (Request.ProcessID ≠ Current.ProcessID) then
16.   if ((Request.Visted[Current.ProcessID] = 0) and
        (InCS(Current.ProcessID) or CompareTimeStamp(Current, Request))) then
17.     Request.ℓ_Count ++;
18.     Current.TS_of_Racer[Request.ProcessID] ← Request.RequestTS;
19.   elseif ((Request.RequestTS < Current.LastRequestTS) and
      (Current.TS_of_Racer[Request.ProcessID]=Request.RequestTS)) then
20.     Request.ℓ_Count —;
21.     Current.TS_of_Racer[Request.ProcessID] ← +∞;
22.   endif;
23.   Request.Next[Current.ProcessID] ← FALSE;
24.   Request.Time ++;
25.   Request.Visted[Current.ProcessID] ← Request.Time;
26.   Select immediate LRV neighbor m_n;
27.   Request.Next[j] ← TRUE, where m_j's are μ_nodes;
28.   Forward Request to m_n;
29. else
30.   if ((Current.LastRequestID = Request.RequestID) and
        ValidTimeStamp(Request.RequestTS)) then
31.     if (Request.Next[ ] ≠ FALSE) then
32.       if ((Request.ℓ_Count + (n – |visited nodes|)) < ℓ) then  //LEConditionI
33.         Enter CS; /*Entering CS*/
34.       else
35.         Select immediate LRV neighbor m_n;
36.         Forward Request to m_n;
37.       endif;
38.     else
39.       Current.LastRequestID ← NULL;
40.       Current.LastRequestTS ← +∞;
41.       if (Request.ℓ_Count < ℓ) then                    //LEConditionII
42.         Enter CS; /*Entering CS*/
43.       else                                             //LEConditionIII
44.         Request_for_CS ( ); /*Requesting for CS again*/
45.       endif;
46.     endif;
47.   else
48.     Do nothing; /*Discarding duplicate permissions*/
49.   endif;
50. endif;
```

Figure 3.2. Pseudocode for handling incoming message.

Next node m_i increments *Time* variable, and updates *Visited*[i] (in line 23–25). It also sets *Next*[i] to FALSE, if it were a μ_*node*. It selects immediate least recently visited (LRV) neighbor m_n, updates *Next*[ ] for its μ_*node*s, and forwards *Request* to m_n (in line 26–28).

Case II: When a node m_i receives its own $Request_i$, (Note that $Request_i$ must have a valid timestamp, because if $Request_i$ holds a timestamp higher than a designated time threshold (timeout), $Request_i$ expires, and m_i discards $Request_i$), node m_i takes action to enter CS only under two cases:

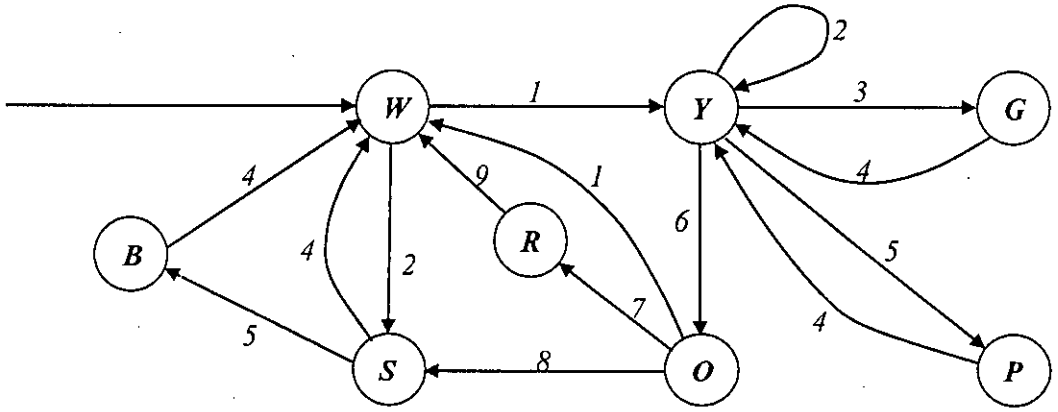
Case IIa: If $Request_i$ has unvisited nodes (i.e. μ_*node*s), and (ℓ_$Count_i$ + number of unvisited nodes) is less than ℓ (i.e. LEConditionI holds in line 32). Otherwise, m_i forwards $Request_i$ to a least recently visited (LRV) neighbor again.

Case IIb: If $Request_i$ has no unvisited nodes (i.e. μ_*node*s), and ℓ_$Count_i$ is less than ℓ (i.e. LEConditionII holds in line 41). Otherwise, m_i discards $Request_i$, regenerates new $Request_i$, and sends this $Request_i$ to its least recently visited (LRV) neighbors (i.e. LEConditionIII holds in line 43).

3.3.4. State Diagram

We now present our protocol via a state diagram for any mobile node m_i. Node m_i operates in one of eight states: *WHITE*, *BLUE*, *YELLOW*, *RED*, *SKY*, *ORANGE*, *GREEN*, or *PURPLE* at any stage (see Figure 3.3). Similar to Section 3.3.3, we denote $Request_{index}$ as a request message of any mobile node m_{index} to enter CS; and $Request_{index}'$ as a request message (of any mobile node m_{index}) that has been already served in the earlier steps; and ℓ_$Count_{index}$ as the counter associated with $Request_{index}$.

Initially current state of node m_i is in *WHITE* state, i.e. node m_i is idle in *WHITE* state. For simplicity, we divide the action of node m_i into two cycles: one is cycle of requesting for CS, and another is cycle of receiving request.



States:

- *WHITE*(W) = Node m_i is idle
- *YELLOW*(Y) = Node m_i is a Requesting node
- *GREEN*(G) = $Request_i.\ell\_Count$ is incremented, where $i \neq j$
- *PURPLE*(P) = $Request_i.\ell\_Count$ is decremented,where $i \neq j$, i is the requesting node
- *ORANGE*(O) = Node m_i receives $Request_i$
- *RED*(R) = Node m_i is in CS
- *SKY*(S) = Node m_i receives $Request_j$ or, $Request_i$ for re–circulating, where $i \neq j$
- *BLUE*(B) = $Request_j.\ell\_Count$ is decremented, where $i \neq j$

Inputs:

- *1* = Sending $Request_i$ to least recently visited (LRV) neighbor
- *2* = Received $Request_j$ is older than $Request_i$, and forwarding $Request_j$ to least recently visited (LRV) neighbor, where $i \neq j$
- *3* = Received $Request_j$ is younger than $Request_i$, where $i \neq j$
- *4* = Forwarding $Request_j$ to least recently visited (LRV) neighbor, where $i \neq j$
- *5* = Originator of $Request_j$ is μ_node of node m_i for $Request_i'$, and $Request_j$ revisits node m_i, where $i \neq j$
- *6* = Receiving $Request_i$ from neighbor
- *7* = $(Request_i.\ell\_Count + n - |\text{visited nodes}|) < \ell$
- *8* = $(Request_i.\ell\_Count + n - |\text{visited nodes}|) \geq \ell$
- *9* = Exiting CS
- *10* = Discarding $Request_i$ if $Request_i$ expires or, if no entries available

Figure 3.3. State Diagram for Mobile Node m_i.

We first explore the node m_i's cycle of requesting for CS. Our protocol allows only an idle node to request for CS, i.e. node m_i only in *WHITE* state can request for CS. To initiate requesting for CS, node m_i executes the following steps:

- Sends *Request* to its nearest neighbor, putting the other immediate neighbors as μ_node(s),

- Switches to *YELLOW* state, and turns into a requesting node (as node m_i has an outstanding *Request*).

In *YELLOW* state, requesting node m_i can receive *Request*(s) from its neighbor(s). Originator of this *Request*(s) may be either m_i, or any other node. Node m_i's receipt of other nodes' *Request* can be classified into four categories:

Case 1: Let node m_i receives other node m_j's unvisited $Request_j$ (i.e. *Request* never visits node m_i before). If $Request_j$ is older than node m_i's own *Request* ($Request_j$ holds a timestamp earlier than that of node m_i's outstanding $Request_i$), node m_i forwards that $Request_j$ to its least recently visited (LRV) neighbor, putting the immediate neighbors that is/are not visited by $Request_j$ yet, as μ_node(s).

Case 2: Suppose node m_i receives other node m_j's visited $Request_j$ (i.e. $Request_j$ revisits node m_i). If originator of $Request_j$ is Λ_node of node m_i for $Request_i'$, node m_i carries out the following operations:

- Switches to *PURPLE* state,
- Decrements ℓ_Count_j (hold by that $Request_j$) by one, (As ℓ_Count_j was incremented by node m_i, when m_i was a requesting node and m_j was a Λ_node of m_i's $Request_i'$. In this state, m_i is done with its CS, and it must allow other to enter into CS by decrementing ℓ_Count_j, which was previously incremented by m_i),
- Forwards that $Request_j$ to its least recently visited (LRV) neighbor, putting the immediate neighbors that is/are not visited by $Request_j$ yet, as μ_node(s),
- Switches back to *YELLOW* state.

Case 3: When node m_i receives other node's $Request_j$ younger than its own (i.e. $Request_j$ holds a timestamp later than that of node m_i's outstanding $Request_i$), node m_i executes the following steps:

- Switches to *GREEN* state,
- Increments ℓ_Count_j (hold by that $Request_j$) by one,
- Forwards that $Request_j$ to its least recently visited (LRV) neighbor, putting the immediate neighbors that is/are not visited by $Request_j$ yet, as μ_node(s),
- Switches back to *YELLOW* state.

Case 4: When node m_i receives its own outstanding $Request_i$, node m_i switches to *ORANGE* state. Now if $Request_i$ has a invalid timestamp, i.e. if $Request_i$ holds a timestamp higher than a designated time threshold (timeout), $Request_i$ expires by the code; Therefore m_i discards $Request_i$, switches back to WHITE state and becomes idle. Other actions of node m_i, in *ORANGE* state, can be divided into three cases:

Case 4a: When entry to CS is available, i.e. LEConditionI holds as sum of ℓ_Count (hold by $Request_i$) and total unvisited nodes (hold by *visited*[ ] of $Request_i$) is less than ℓ, or LEConditionII holds as there are no unvisited nodes ($Request_i$ visits the entire network), and ℓ_Count (hold by $Request_i$) is less than ℓ, node m_i switches to *RED* state (i.e. enters CS). On exiting CS, node m_i (in *RED* state) switches back to *WHITE* state, and becomes idle.

Case 4b: When $Request_i$ visits the entire network (all node), but entry to CS is unavailable, node m_i discards its $Request_i$, switches back to *WHITE* state, and turns into idle.

Case 4c: When entry to CS is unavailable, and $Request_i$ visits a subset of the entire network (i.e. some nodes are not visited), node m_i switches to *SKY* state. We explicate the operations of m_i node a bit later.

In *WHITE* state, when node m_i receives other nodes' $Request_j$, it switches to *SKY* state, and starts the receiving request cycle. Note that node m_i is not a requesting node at this cycle.

To explore the operations node m_i takes in *SKY* state, we consider some cases:

Case 1: When node m_i receives a visited $Request_j$ older than its own last $Request_i'$, such that, most recent visit of received *Request* to node m_i took place, when node m_i was requesting node, and timestamp of $Request_j$ is older than that of its own last $Request_i'$, i.e. originator of $Request_j$ is π_node of m_i for $Request_i'$, node m_i performs the following actions:

- Switches to *BLUE* state,

- Decrements ℓ_Count_j (hold by received $Request_j$) by one, (As ℓ_Count_j was incremented by node m_i, when m_i was a requesting node and m_j was a π_node of m_i's $Request_i$'),
- Forwards that $Request_j$ to its least recently visited (LRV) neighbor, putting the immediate neighbors that is/are not visited by $Request_j$ yet, as μ_node(s),
- Switches back to *WHITE* state.

Case 2: When node m_i receives a visited $Request_j$ younger than its own last $Request_i$, or receives an unvisited $Request_j$ ($Request_j$ never visits node m_i before), node m_i takes the following steps:

- Forwards that *Request* to its least recently visited (LRV) neighbor, putting the immediate neighbors that is/are not visited by $Request_j$ yet, as μ_node(s),
- Switches back to *WHITE* state to become idle.

3.3.5. Operations

This section illustrates a sample execution of the algorithm. For simplicity, we describe the protocol in a somewhat synchronous manner, even though all the activities are in asynchronous manner. Moreover, we assume that the *ℓ–Exclusion* problem involves six mobile nodes *A*, *B*, *C*, *D*, *E*, which want to coordinate themselves to access and to control intermittently two single, non–sharable, and reusable designated piece of code called critical section CS (i.e. $\ell = 2$). Throughout this section, subscripts on data structures to indicate a particular node are only included when needed. For example, we denote $Request_A$ as a request message of any mobile node *index* to enter CS; $Request_A'$ as a request message (of any mobile node *A*) that has been served in finite number of steps; ℓ_Count_A as the counter associated with $Request_A$; $Visited_A$ as *Visited*[ ] associated with $Request_A$; and $Next_A$ as *Next*[ ] associated with $Request_A$;

Let consider an illustration of the algorithm on a dynamic network (in which node moves, but links do not change) in Figure 3.4. Snapshots of the system configuration during algorithm execution are shown, with time increasing from Figure 3.4(a) to Figure 3.4(l). In this figure and for the other following figures in this section, each dashed line indicates the direct physical communication link connecting two

circulating nodes, each thin line represents the direct physical communication link over which messages flow for once, each thin line (thin communication link) becomes thicker when more messages flow over that link. Solid and dashed arrows indicate *Request*, the number adjacent to *Request* represents *ℓ_Count*. A requesting node (originator of *Request*) is represented by an intensely shaded node, while a slightly shaded node stands for a node executing CS.

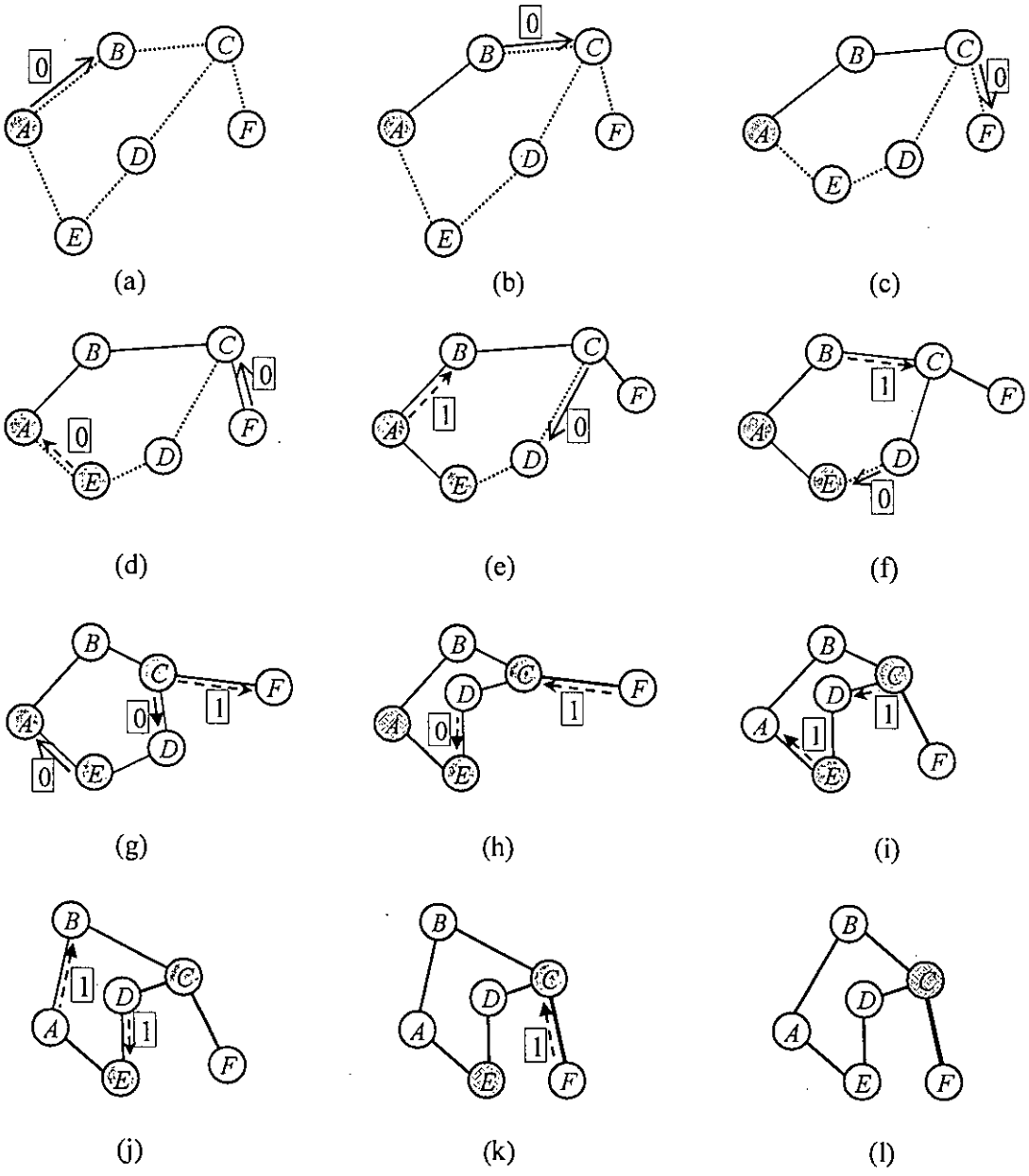


Figure 3.4. Operation of the *ℓ–Exclusion* algorithm on dynamic network (where node moves, but links do not change).

In Figure 3.4(a), node A intends to enter CS, and sends $Request_A$ to its nearest neighbor B. Note that A is now requesting node, and $\ell\_Count_A = 0$. In part (b), node D moves, node B, on receipt of $Request_A$, forwards $Request_A$ to its neighbor C, as node C is the B's one of two neighbors that is least recently visited (LRV) by $Request_A$. Now node E moves, and node C, receiving $Request_A$, sets node D as μ_node for $Request_A$, and forwards $Request_A$ to node F, as shown in part (c). Part (d) depicts the system configuration where node F, after receiving $Request_A$, forwards it to node C, and node E, to enter CS, sends $Request_E$ to its nearest neighbor A. Note that both A and E are now requesting node, and $\ell\_Count_A = \ell\_Count_E = 0$.

In part (e), node F moves, node C forwards $Request_A$ to node D, and node A forwards $Request_E$ to node B accordingly. Observe that node D is no longer μ_node for $Request_A$, and $\ell\_Count_E = 1$, as node E is now μ_node of node A for $Request_A$. $Request_E$, after arriving node B, visits node C and F, and reaches node C, while $Request_A$, after reaching node E, visits node E, and returns back to the originator node A, as depicted in part (f)–(h). At this point in the execution, node C and D are in their new positions, and node A, on receipt of $Request_A$, enters CS, as LEConditionII holds. However, node F and B moves from their positions, and $Request_E$, after visiting node C and D, returns back to the originator node E, as in part (i)–(j). Part (k)–(l) shows that node A releases CS, and node C enters CS, as LEConditionI holds. At this time, there are no pending *Request*s, no requesting nodes.

In Figure 3.5, we now explore the execution of the *ℓ–Exclusion* algorithm on a dynamic network (in which nodes moves, and links change). Figure 3.5(a) depicts the same snapshot of the system execution as shown in Figure 3.4(a), with time increasing for Figure 3.5 (a) to Figure 3.5(l). Figure 3.5(b)–(c) depicts system configuration where node E, after dropping its link with node A, moves to a new position, resulting in a link formation between node E and F. Meanwhile, $Request_A$ visits node B and C, and is forwarded to node F. Note that node D is now a μ_node for $Request_A$. In part (d), node C intends to enter CS, and forwards $Request_C$ to node B. Now node A and C both are requesting node. A link is formed between node B and D, due to movement of node D, following a link removal between node C and F, as shown in Figure 3.5(e)–(f). By this time, $Request_C$ visits node A, and is forwarded to

node D, while $Request_A$ is forwarded to the originator node A by node D. At this moment, $\ell\_Count_C = 1$, as node C is a *n_node* of node A for $Request_A$. Figure 3.5(g) demonstrate a system configuration in which node A enters CS, as LEConditionII holds, and node F disappears, due to its crash–failure, followed by a link formation between node E and C, due to movement of node E. $Request_C$ is now forwarded to node E by node D. Node B moves resulting in two link removals between node B and D, and between node B and C, while node C receives its own $Request_C$, as can be seen in part (h). As LEConditionI or LEConditionII or LEConditionIII does not hold, Node C forwards $Request_C$ to its least recently visited (LRV) neighbor node E once again. Figure 3.5(i)–(k) demonstrate the movement of node A, resulting in a link formation between node A and C. This link formation is followed by the recovery of node F from failure, with two link formations between node F and D, and between node F and D accordingly. In the mean time, link between node D and E breaks, and $Request_C$ return to its originator node C. Part (l) shows node C in CS, node E in a changed position.

3.3.6. Correctness

To establish the correctness of the *ℓ–Exclusion* algorithm, we now demonstrate that the algorithm guarantees the *ℓ–Exclusion*, ℓ–Lockout Avoidance and First–Come–First–Guaranteed properties defined in Section 1.2.

For this section, we presume that the *ℓ–Exclusion* problem involves several mobile nodes to control access to ℓ identical copies of an indivisible, non–sharable, designated piece of code called critical section (CS), and in the combination of our *ℓ–Exclusion algorithm* and any collection of mobile nodes, there is no reachable system state in which more than ℓ mobile nodes are in CS. We also recall the assumptions mentioned in Section 3.2. In addition, we assume that no *Request* expires, i.e. *Request* never holds a timestamp higher than a designated time threshold (timeout), and originator of *Request* never fails. We view a node failure as the failure of links between the nodes and each of its neighbors, and similarly, a node recovery is viewed as the recovery of links between the recovering nodes and its neighbors, i.e. we assume failure or recovery of nodes as link change(s). We don't consider the failures of node holding message of other, or originator (node) of message.

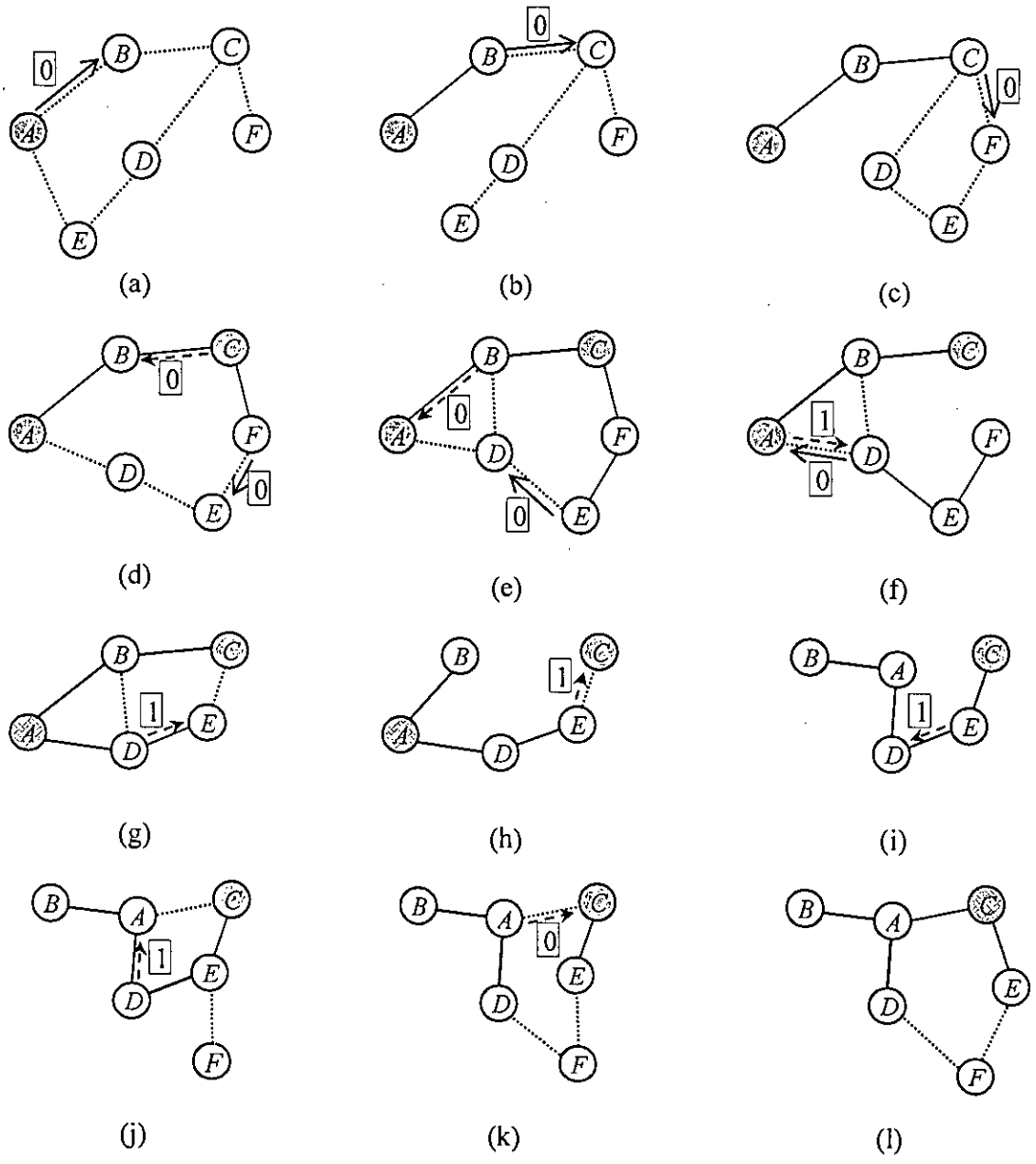


Figure 3.5. Operation of the *ℓ–Exclusion* algorithm on dynamic network (where node moves, and links change).

Throughout this section, subscripts on data structures to indicate a particular node are only included when needed. For example, we denote m_{index} as a mobile node; $Request_{index}$ as a request message of any mobile node m_{index} to enter CS; $Request_{index}'$ as a request message (of any mobile node m_{index}) that has been served in finite number of steps; ℓ_Count_{index} as the counter associated with $Request_{index}$; $Visited_{index}$

as *Visited*[ ] associated with $Request_{index}$; and $Next_{index}$ as *Next*[ ] associated with $Request_{index}$;

Lemma 3.1. *Each and every Request eventually returns back to its originator node.*

Proof. We show by the way of contradiction that each and every *Request* visits the active nodes (i.e. alive nodes) until LEConditionI or LEConditionII or LEConditionIII holds, and eventually returns back to its originator node. We now consider some cases:

Case 1 (*Nodes move, but no link changes*): Let n nodes are active. Assume that $Request_i$ arrives node m_k in some reachable system state, where $i \neq k$, and m_k forwards $Request_i$ to its least recently visited (LRV) neighbor, say m_g, putting the immediate neighbors that is/are not visited by $Request_i$ yet, as μ_node(s). By the code, m_g may be a visited node, or an unvisited node, or a μ_node for $Request_i$, and m_g does the same as m_k did. However, as each μ_node or unvisited node is in the neighborhood of other visited node(s) or unvisited node(s) or μ_node(s), each one eventually becomes the least recently visited (LRV) neighbor of other nodes. Now by contradiction, assume that $Request_i$ may visit the originator m_i before LEConditionI or LEConditionII or LEConditionIII holds, but $Request_i$ does never eventually returns back to its originator node m_i. We now consider two cases:

Case 1a: Consider an execution that leads to a system state where $Request_i$ arrives m_s after visiting n–1 nodes except the originator m_i. By the code, m_s forwards $Request_i$ to its least recently visited (LRV) neighbor. As m_i is in the neighborhood of one or more visited nodes, m_i eventually becomes the least recently visited neighbor of any other visited node, and $Request_i$ reaches m_i, a contradiction.

Case 1b: Consider an execution that leads to a system state where $Request_i$ arrives m_s, after visiting n active nodes including its originator m_i. By the code, m_s forwards $Request_i$ to its least recently visited (LRV) neighbor. As m_i is in the neighborhood of one or more visited nodes, m_i eventually becomes the least recently visited neighbor of any other visited node, and $Request_i$ reaches m_i, a contradiction.

Case 2 (*Nodes move and links change*): Assume that n nodes are active. Assume that $Request_i$ arrives node m_k in some reachable system state, where $i \neq k$, and m_k forwards $Request_i$ to its least recently visited (LRV) neighbor (chosen dynamically), say m_g, putting the immediate neighbors that is/are not visited by $Request_i$ yet, as μ_node(s). By the code, m_g may be a visited node, or an unvisited node, or a μ_node for $Request_i$, and m_g does the same as m_k did. However, as each μ_node or unvisited node remains static in the neighborhood of other visited node(s) or unvisited node(s) or μ_node(s) for a sufficiently long time, each one eventually becomes the least recently visited (LRV) neighbor of other nodes, and is visited by $Request_i$. In case of failure, any μ_node or unvisited node becomes inactive (i.e. not alive). On recovery from failure, any unvisited node eventually becomes active, and eventually becomes μ_node and/or the least recently visited neighbor of other visited node, as each recovered node remains static in the neighborhood of other active visited node(s) or unvisited node(s) or μ_node(s) for a sufficiently long time. Now by contradiction, assume that $Request_i$ may visit the originator m_i before LEConditionI or LEConditionII or LEConditionIII holds, but $Request_i$ does never eventually returns back to its originator node m_i. We now consider two cases:

Case 2a: Consider an execution that leads to a system state where $Request_i$ arrives m_s after visiting n–1 nodes except the originator m_i. Rest of this case is similar to *Case 1a*.

Case 2b: Consider an execution that leads to a system state where $Request_i$ arrives m_s, after visiting n active nodes including its originator m_i. Rest of this case is similar to *Case 1a*. □

Lemma 3.2. *The ℓ–Exclusion algorithm allows ℓ–1 mobile nodes to fail or crash.*

Proof. We show by the way of contradiction that the *ℓ–Exclusion algorithm* allows at least one requesting node to enter CS, when ℓ–1 mobile nodes are inactive (i.e. failed, or crashed). By contradiction, assume that the *ℓ–Exclusion algorithm* allows one requesting node m_i to enter CS in some reachable system state, when ℓ mobile nodes are inactive. Consider an execution that leads to this system state. Let m_i sends

Request$_i$ to its nearest neighbor, putting the other immediate neighbors as μ_*node*(s), and switch from *WHITE* to *YELLOW* state. Imagine that *Request*$_i$ visits all active nodes, and eventually returns back to m_i, by Lemma 3.1. However, by the code, LEConditionI or LEConditionII don't hold for m_i. Therefore, m_j discards *Request*$_j$, switches to *WHITE* state and becomes idle, a contradiction. □

Lemma 3.3. *The ℓ–Exclusion algorithm satisfies the ℓ–Exclusion property for ℓ = 1.*

Proof. By Lemma 3.2, the *ℓ–Exclusion algorithm* allows no node to fail or crash when ℓ = 1. Hence, we only consider the case where nodes move, but no link changes. By contradiction, assume that mobile nodes m_i and m_j, $i \neq j$, are simultaneously in CS in some reachable system state. Consider an execution that leads to this system state. Let both m_i and m_j send *Request*$_i$ and *Request*$_j$ accordingly to their nearest neighbors, putting the other immediate neighbors as μ_*node*(s), and switch from *WHITE* to *YELLOW* state before entering CS. By the code, both *Request*$_i$ and *Request*$_j$ require to visit the entire network. We now consider four cases:

Case A: Let assume that *Request*$_i$ has earlier timestamp than *Request*$_j$. So, on receipt of *Request*$_i$, m_j forwards *Request*$_i$ to m_j's least recently visited (LRV) neighbor, putting the immediate neighbors that is/are not visited by *Request*$_i$ yet, as μ_*node*(s) for *Request*$_i$, and hangs on *YELLOW* state. On the other hand, on receipt of *Request*$_j$, m_i switches to *GREEN* state, increments ℓ_*Count*$_j$ (hold by that *Request*$_j$) by one, and forwards *Request*$_j$ to m_i's least recently visited (LRV) neighbor, putting the immediate neighbors that is/are not visited by *Request*$_j$ yet, as μ_*node*(s) for *Request*$_j$, and switches back to *YELLOW* state. By Lemma 3.1, *Request*$_i$ and *Request*$_j$ both eventually return back to m_i and m_j accordingly, and m_i and m_j both switch to *ORANGE* state. On receipt of *Request*$_i$, m_i switches to *RED* state as LEConditionII holds, and enters CS. Similarly, on receiving *Request*$_j$, m_j intends to switch to *RED* state. However, by the code, it is impossible as LEConditionII holds only for m_i. Therefore, m_j discards *Request*$_j$, switches to *WHITE* state and becomes idle, a contradiction.

Case B: Let assume that $Request_j$ has earlier timestamp than $Request_i$. Rest of this case is similar to *Case A*.

Case C: Let $Request_i$ has same timestamp than $Request_j$, such that $i > j$, i.e. identifier of node m_i is larger than that of m_j. So, on receipt of $Request_i$, m_j forwards $Request_i$ to m_j's least recently visited (LRV) neighbor, putting the immediate neighbors that is/are not visited by $Request_i$ yet, as μ_node(s) for $Request_i$, and hangs on *YELLOW* state. On the other hand, on receipt of $Request_j$, m_i switches to *GREEN* state, increments ℓ_Count_j (hold by that $Request_j$) by one, and forwards $Request_j$ to m_i's least recently visited (LRV) neighbor, putting the immediate neighbors that is/are not visited by $Request_j$ yet, as μ_node(s) for $Request_j$, and switches back to *YELLOW* state. By Lemma 3.1, $Request_i$ and $Request_j$ both eventually return back to m_i and m_j accordingly, and m_i and m_j both switch to *ORANGE* state. On receipt of $Request_i$, m_i switches to *RED* state as LEConditionII holds, and enters CS. Similarly, on receiving $Request_j$, m_j intends to switch to *RED* state. However, by the code, it is impossible as LEConditionII holds only for m_i. Therefore, m_j discards $Request_j$, switches to *WHITE* state and becomes idle, a contradiction.

Case D: Let $Request_j$ has same timestamp than $Request_i$, such that $j > i$, i.e. identifier of node m_j is larger than that of m_i. Rest of this case is similar to *Case C*. □

Theorem 3.1. *The ℓ–Exclusion algorithm satisfies the ℓ–Exclusion property.*

Proof. We show by induction on ℓ that no more than ℓ mobile nodes are simultaneously in their CS.

Basis: Let ℓ=1, the statement holds by Lemma 3.2.

Induction: By inductive hypothesis, assume that the statement holds for ℓ–1 mobile nodes in a system having ℓ identical copies of CS, i.e. system reaches a system state where (ℓ+1) mobile nodes intending to enter CS, send their *Request* to their nearest neighbors, and switches to *YELLOW* state; in next system state, ℓ–1 mobile nodes, m_1, m_2, …, m_i, …, m_ℓ are now simultaneously in their CS, i.e. ℓ–1 nodes simultaneously hang on *RED* state. Now other two requesting nodes m_i and m_j hang on *YELLOW*

state. By Lemma 3.3, one of m_i and m_j switches to *RED* state, and enters CS. And the other one discards its *Request*, switches to *WHITE* state and becomes idle. □

Theorem 3.2. *The ℓ–Exclusion algorithm satisfies the ℓ–Lockout Avoidance property.*

Proof. We show by the way of contradiction that no requesting node waits forever to enter CS, i.e. each and every requesting node eventually succeeds to enter CS. Imagine a system that consists of n mobile nodes, and a system state where (ℓ+1) nodes intend to enter CS (where $\ell < n$), switch to *YELLOW* state, and send $Request_1$, $Request_2$, ..., $Request_i$, ..., $Request_\ell$, $Request_{\ell+1}$ to their nearest neighbors accordingly. Suppose that $Request_h$ has the youngest timestamp among these (ℓ+1) *Requests*. Now by contradiction, assume that originator of $Request_h$ never succeeds, i.e. m_h waits forever to enter CS. However, by Lemma 3.1, $Request_1$, $Request_2$, ..., $Request_i$, ..., $Request_\ell$, $Request_{\ell+1}$ eventually return back to their originator node m_1, m_2, ..., m_i, ..., m_ℓ, $m_{\ell+1}$ accordingly, and m_1, m_2, ..., m_i, ..., m_ℓ, and $m_{\ell+1}$ switch to *ORANGE* state. We now consider two cases:

Case A (*Nodes move, but no link changes*): By Theorem 3.1, ℓ mobile nodes switch to *RED* state, and enter CS, as LEConditionI or LEConditionII holds for those ℓ nodes. However, on receipt of $Request_h$, m_h discards $Request_h$, and switches back to *WHITE* state. As LEConditionIII holds for m_h, m_h regenerates $Request_h$, switches to *YELLOW* state, and sends $Request_h$ to m_h's nearest neighbor. Under our assumptions, those ℓ mobile nodes release CS in the mean time, and switch back to *WHITE* state. By Lemma 3.1, $Request_h$ eventually returns back to their originator node m_h, and switches to *ORANGE* state. This time, on receiving $Request_h$, m_h, as LEConditionI or LEConditionII holds, switches to *RED* state, and enters CS, a contradiction.

Case B (*Nodes move and links change*): By Lemma 3.2 and Theorem 3.1, each of $Request_1$, $Request_2$, ..., $Request_i$, ..., $Request_\ell$, $Request_{\ell+1}$ visits at least (n–ℓ+1) mobile nodes, switches to *RED* state, and at least one of those nodes enters CS, as LEConditionI or LEConditionII holds for the node(s). However, on receipt of $Request_h$, m_h discards $Request_h$, and switches back to *WHITE* state. As LEConditionIII holds for m_h, m_h regenerates $Request_h$, switches to *YELLOW* state, and sends $Request_h$ to m_h's nearest neighbors. Under our assumptions, node(s) in CS release(s) CS in the

mean time, and switch back to *WHITE* state. By Lemma 3.1, $Request_h$ eventually returns back to their originator node m_h, and switches to *ORANGE* state. This time, on receipt of $Request_h$, LEConditionI or LEConditionII holds for m_h by Lemma 3.2. Therefore, m_h switches to *RED* state, and enters CS, a contradiction. □

Theorem 3.3. *The ℓ–Exclusion algorithm ensures the First–Come–First–Guaranteed property.*

Proof. We show by the way of contradiction that no node requesting later than node m_i can enter CS before m_i, when only one entry is available. By Lemma 3.2, the *ℓ–Exclusion algorithm* allows no node to fail or crash when $\ell = 1$. Hence, we only consider the case where nodes move, but no link changes. Let assume a system where one entry to CS is available. Consider a system state where each of mobile nodes m_i and m_j switches to *YELLOW* state, and send $Request_i$ and $Request_j$ accordingly to their nearest neighbors, such that $Request_i$ has an earlier timestamp than that of $Request_j$. By contradiction, presume that in some reachable system state $Request_j$ succeeds to enter CS before $Request_i$ does. Consider an execution that leads to this system state. By Lemma 3.1, $Request_i$ and $Request_j$ eventually return back to their originator node m_i and m_j accordingly, and both switch to *ORANGE* state. On receipt of $Request_i$, LEConditionI or LEConditionII holds for m_i by the code. Therefore, m_h switches to *RED* state, and enters CS. However, on receipt of $Request_j$, LEConditionI or LEConditionII doesn't hold for m_j by the code, and m_h discards $Request_j$, and switches back to *WHITE* state, a contradiction. □

3.4. Chapter Summary

In this chapter, we have sketched the system model, assumptions on system, mobile nodes and networks, to design the proposed *ℓ–Exclusion* Algorithm. We have also presented a brief outline followed by detailed description along with required data structures, a state diagram of the proposed algorithm, and operational examples of the algorithm in this chapter. The chapter has been ended with proofs of correctness to establish the proposed algorithm.

CHAPTER 4

SIMULATION AND PERFORMANCE EVALUATION

4.1. Introduction

In this chapter, we study the performance of the proposed algorithm in a mobile ad hoc setting by a simulation technique since the operations of mutual exclusion algorithms are very complex and quite difficult to analyze mathematically [55]. Through our simulation, we consider several performance metrics that can significantly impact the behavior of such an algorithm in different ad hoc settings. We also compared the performance of the proposed algorithm to the *k–Reverse Link* (KRL) algorithm presented in [140, 141].

4.2. Simulation Setting

We simulate our algorithm using PARSEC [5, 101], a parallel C–based discrete–event simulation language, developed in UCLA Parallel Computing Library. The main objective of our simulations is to gain a better understanding of how to learn in detail how various simulation parameters impact the performance of the *ℓ–Exclusion* algorithm, and also to observe different performance metrics when mobile node crashes or merges.

4.2.1. Performance Metrics

The performance metrics that we consider in our simulations are the *Message Overhead* (*M*), *Average Waiting Time per CS Entry* (*W*), and *Success Rate* (*S*).

Message Overhead (*M*) is the average communication Complexity (per request) to enter CS, i.e. the average number of messages transmitted or forwarded against a single request of a node that intends to enter CS. Suppose *Request* messages of *r*

originator nodes traverse $x_1, x_2, x_3, \ldots x_r$ number of nodes accordingly. So, *Message Overhead*, $M = \sum_{p=1}^{r} \frac{(x_p - 1)}{r}$.

Average Waiting Time per CS Entry (W) is defined as the average fraction of time units that a node spends in waiting for CS after its request. Let r number of nodes forwards *Request*s for CS, and q number of nodes succeeds to enter CS after delay of $t_1, t_2, t_3, \ldots t_q$ time units accordingly, where $q \leq r$. *Request*s of the rest of the nodes, i.e. *Request*s of $(r - q)$ expires after timeout. So, *Average Waiting Time per CS Entry*, $W = \sum_{p=1}^{q} \frac{t_q}{r}$.

We define *Success Rate* (S) as the percentage of the successful requests, i.e. the percentage of the requesting nodes succeeding to enter CS. Imagine that r number of nodes forwards *Request*s for CS, and q number of nodes succeeds to enter CS, where $q \leq r$. So, *Success Rate*, $S = \frac{q}{r} \times 100\%$.

4.2.2. Simulation Environment and Parameters

The simulation environment is composed by 30 mobile nodes randomly spread over an obstacle–free terrain of 1000×1000 meters. The transmission radius of each node is 250 meters. The dimension of the mobility area and the transmission radius of each node have been chosen after several trials as a tradeoff between node mobility and *Message Overhead* to enter CS. A 30 node system has been chosen, in part, because for networks larger than 30 nodes the time needed for simulation was very high. Also, ad hoc networks are generally envisioned to be much smaller scale than wired networks like the Internet. Typical numbers of nodes used for simulations of ad hoc networks range from 10 to 50 [10, 19, 26, 69, 76, 77, 125].

In our simulation, each run of the simulation has been triggered for 5000 time units, and each request message has expired after its traversal for 600 time units. Each CS execution has taken one time unit, and each message transmission time was one time unit. Requests for the CS have been modeled as a Poisson process with arrival

rate λ_{req}, which represents the number of requests (per second) generated by a single node. Thus the time delay between when a node left the CS, and another node made its next request to enter the CS is an exponential random variable with mean $1/\lambda_{req}$ time units.

Mobility of node is modeled with a random–waypoint behavior [70], i.e. a node moves towards a randomly selected point inside the area and then pauses for a selected amount of time (pause time) before moving again. Nodes move at various speed inside the 1000×1000 meters area. Each link change associated with each node movement has been considered as the deletion of a link chosen at random (whose loss did not disconnect the graph) and the formation of a link chosen at random. The simulation has been carried out under three different mobility settings, chosen by adjusting the pause time so that the percentage of the total simulation time the node does move and link does change (calculated considering the average time required for a movement) is 100% (high mobility), 10% (low mobility), and 0% (zero mobility) respectively. Rationally, our choice for the value of the low mobility parameter corresponds to the situation where nodes remain stationery for up to a few of ten seconds after moving and prior to making another movement, whereas our choice for the value of high mobility parameter represents the most volatile network, where nodes never remain static between moves. During periods of mobility, node is allowed to move into a new point inside the 1000×1000 meters area, only when this new point is in the transmission radius of any other node, and thus nodes are never allowed to disconnect the entire network due to mobility. Node failures and node recoveries from failures modeled as a Poisson process with crash–merge rate λ_{crash_merge}, which represents the number of nodes (per second) failed or recovered from failures. Thus the time delay between each node failure and/or node recovery from failure is an exponential random variable with mean $1/\lambda_{crash\_merge}$ time units. Each link change associated with each node failure has been considered as the deletion of the link of failed node with other node(s), and merge as the formation of a link between merged node and other node(s) chosen at random.

In each run of the simulations, we have considered the number of available copies of identical resources (ℓ) to be lesser than the number of mobile nodes (n) to demonstrate mutual exclusive access of nodes to CS, as system never requires mutual exclusion if it has at least one resource for each node (i.e. $\ell \geq n$). We denote x% available copies of identical resources as $\ell = nx/100$, i.e. at most $nx/100$ nodes can concurrently execute their CS, when x% resources are available. For example, if a homogeneous system consists of 50 nodes with 30% available identical resources, at most 15 nodes can enter CS simultaneously.

In our simulations, collected performance measures (viz. *Message Overhead*, *Average Waiting Time per CS Entry*, and *Success Rate*) are probabilistic in nature. To increase the accuracy of results, we have taken each of the plotted point of the graphs from the average of five to ten repetitions of the simulation.

Our *ℓ–Exclusion* (LE) simulation starts with initializing a connected graph whose vertices indicate mobile nodes, and edges represent initial communication links among nodes. Based on the transmission range of nodes, initial links among nodes have been chosen randomly, such that no node has been initialized at a position outside the 1000×1000 meters area, or out of transmission radius of all other nodes.

4.3. Sensitivity Analyses: Results versus Performance

In simulations, we have varied the request load on the system (λ_{req}), the number of identical resources (ℓ), the degree of crash–merge rate (λ_{crash_merge}) under different mobility settings (zero, low, and high mobility) to observe the behavior of our predefined performance metrics (i.e., *Message Overhead* (M), and *Average Waiting Time* (T)) as the function of request load (λ_{req}), or crash–merge rate (λ_{crash_merge}), or the number of identical resources (ℓ). To limit the illustrations, we now depict only some particular simulation–based scenarios, where subtle changes in various parameters (e.g., request load, number of identical resources, crash–merge rate etc.) result in salient differences in the behavior of our performance metrics.

4.3.1. Message Overhead

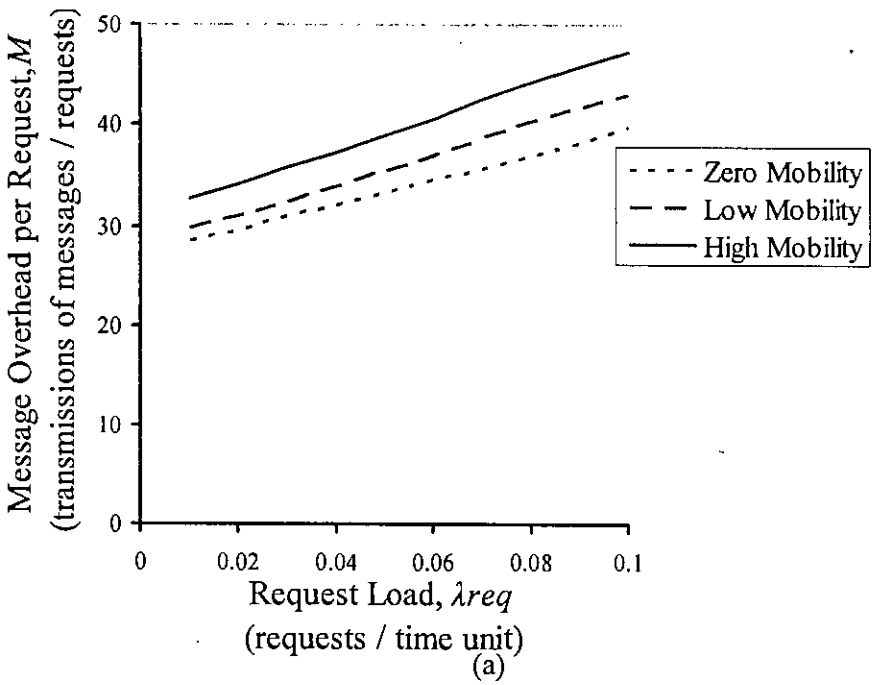
We first learn the impact of the request load (λ_{req}), and the number of identical resources (ℓ) on the *Message Overhead* (M) under an ad hoc environment where no node failures or merges occur. The impact of the crash–merge rate (λ_{crash_merge}), and the number of identical resources (ℓ) on the *Message Overhead* (M) are also investigated for an ad hoc environment where at ℓ–1 nodes can fail and failed nodes may merge.

4.3.1.1. Impact of request load

Figure 4.1(a) and (b) plot *Message Overhead* (M) against request load (λ_{req}) for three mobility settings (high, low, and zero). We varied the values of λ_{req} increasing from 10^{-2} (the time units between requests is 10^{2}) to 10^{-1} (the mean time units between requests is 10). We chose 0.1 for the high load value of λ_{req}, because at this rate each node would have a request pending almost all the time, whereas the low load value of $\lambda_{req} = 10^{-2}$ represents a much less busy network, with requests rarely pending at all nodes at the same time.

Plots are shown for runs under two variants of number of identical resources (ℓ), i.e. in part (a), $\ell = 6$, and $\ell = 9$, in part (b). We chose the lower ($\ell = 6$) value of ℓ to observe the effect of λ_{req} on M, when system has 20% identical copies of resources to serve. Similarly, we chose the higher value ($\ell = 9$) of ℓ to observe the same, when system has 30% identical copies of resources than it has earlier.

The plots show that *Message Overhead* (against increasing request load) increases linearly under various mobility settings, when resources are limited (20%). As request load increases, request messages require traversing more unvisited nodes, causing *Message Overhead* to increase proportionally. Due to mobility, request messages traverse some nodes more than once, while the rest of the nodes remain unvisited for long time. As a result, *Message Overhead* increases. However, the impact of request load on *Message Overhead* under zero mobility is closer to that under high mobility.



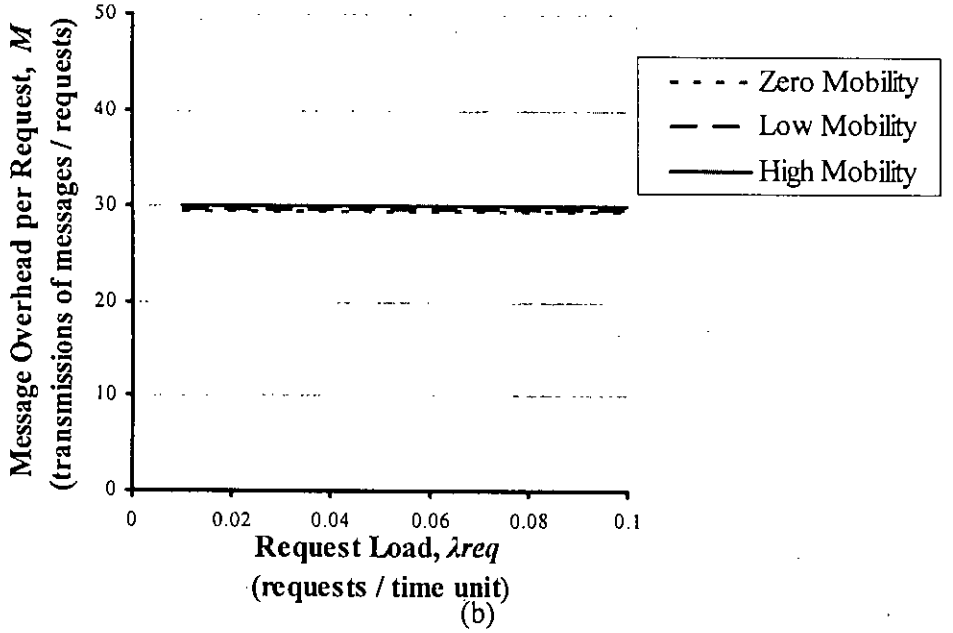


Figure 4.1. Effect of request load (λ_{req}) on *Message Overhead* (*M*), for (a) $\ell = 6$, (b) $\ell = 9$, when no node failure or merge occurs.

Observe that increasing rate of *Message Overhead* (against increasing request load) reduces to nil under all mobility settings, when there are more resources (30%). This

is due to the fact that, at this time, request messages require visiting less number of unvisited nodes to collect votes. As a less number of request messages remain pending, *Message Overhead* remains constant when request load increases. For the same reason, the impact of mobility on *Message Overhead* is almost nothing, if 30% resources are avaiable. Hence, when 30% or more resources are avaiable, there is absolutely no impact of request load on *Message Overhead* even in a highly mobile environment.

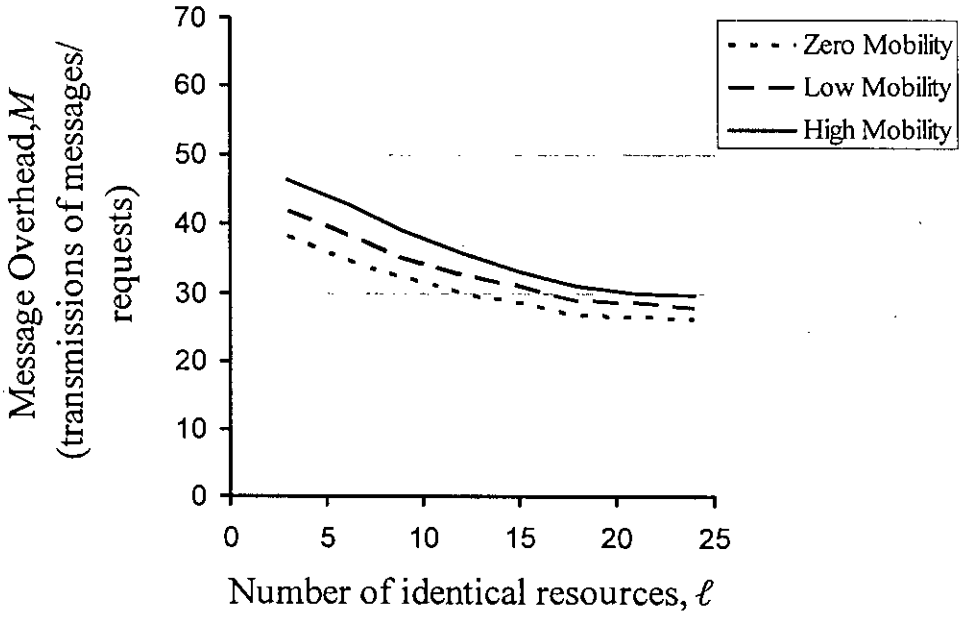
4.3.1.2. Impact of the number of identical resources

Figure 4.2(a) and (b) plot *Message Overhead* (*M*) against the number of identical resources (ℓ). We varied the values of ℓ increasing from 3 (system has 10% resources to serve) to 24 (system has 80% resources to serve).

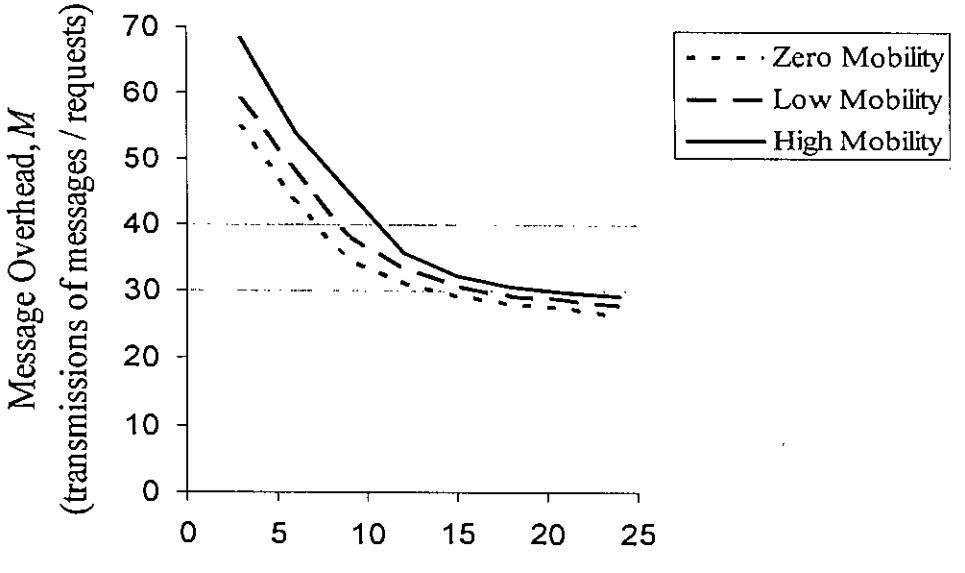
We chose 24 for the high resource value of ℓ, because at this time each node would access CS very promptly, because a request would require fewer votes to succeed. The low load value ($\ell = 3$) of ℓ represents a much busy network with requests pending at all nodes at the same time, because each node would require its request to traverse the entire network to fulfill its demand. Plots are shown for runs under two variants of request load (λ_{req}), i.e. in part (a), $\lambda_{req} = 0.04$, and $\lambda_{req} = 0.08$, in part (b).We chose the lower value ($\lambda_{req} = 0.04$) of λ_{req} to observe the effect of ℓ on *M*, when system may have a small number of pending requests. Similarly, we chose the higher value ($\lambda_{req} = 0.08$) of λ_{req} to look the effect of ℓ on *M* closely, when system may have a large number of pending requests.

The plots focus that *Message Overhead* reduces considerably against increasing number of identical resources. Because, the more available resources increase, the lesser number of unvisited nodes request messages require visiting. In case of high request load in part (b) of Figure 4.2, the reduction in *Message Overhead* is more radical than that in case of lower request load in part (a) of Figure 4.2. This is due to the fact that request messages require visiting more unvisited nodes under high request load than under low request load, when limited resources are available. As available resources increase, *Message Overhead* decreases due to visiting lesser

number of nodes by request messages. However, when enough resources are available, the decreasing rate of *Message Overhead* is reduced as the number of available reaches to saturation against the request load.



(a)



(b)

Figure 4.2. Effect of the number of identical resources (ℓ) on *Message Overhead* (M), for (a) $\lambda_{req} = 0.04$, (b) $\lambda_{req} = 0.08$, when no node failure or merge occurs.

Observe that the impact of the increasing resources on *Message Overhead* under zero mobility is always little lower than that under high mobility, i.e. scarce resource has small additional impact on *Message Overhead* due to high mobility. This is because, request messages require visiting some nodes more than once due to mobility, causing increase in *Message Overhead.* But, the rate of increase in *Message Overhead* increases proportionally.

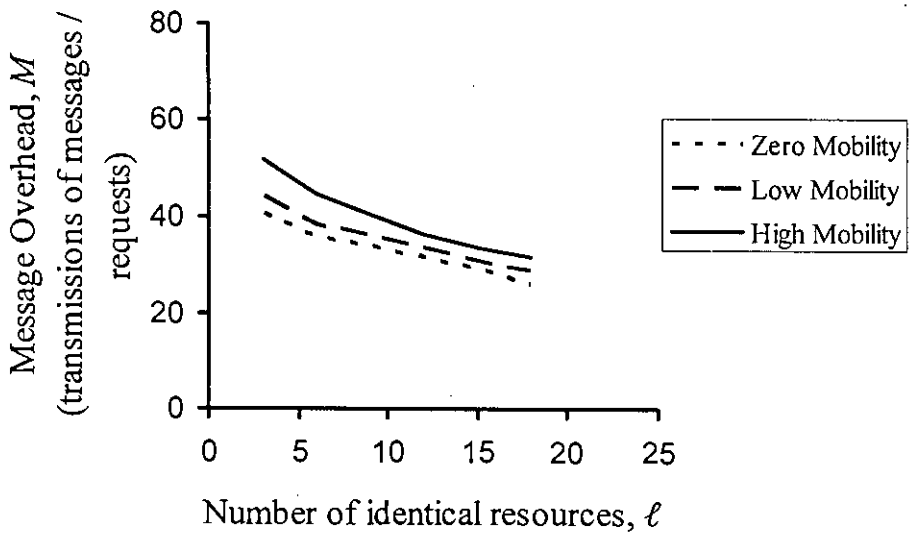
So far, we have not tolerated node failures or merges during simulation. Figure 4.3(a) and (b), demonstrate the effect of the number of identical resources (ℓ) on *Message Overhead* (M), where we allow at most ℓ–1 node(s) to fail individually or simultaneously and failed node(s) to merge (at any instance) during simulation.

Plots are shown for runs under two variants of crash–merge rate (λ_{crash_merge}), i.e. in part (a), λ_{crash_merge} = 0.04, and λ_{crash_merge} =0.1 in part (b). We chose 0.1 as the higher value of crash–merge rate (λ_{crash_merge}), because at this rate, the network is much vulnerable. Similarly, we chose 0.04 as the lower value of crash–merge rate (λ_{crash_merge}), because at this rate, the network is less vulnerable.

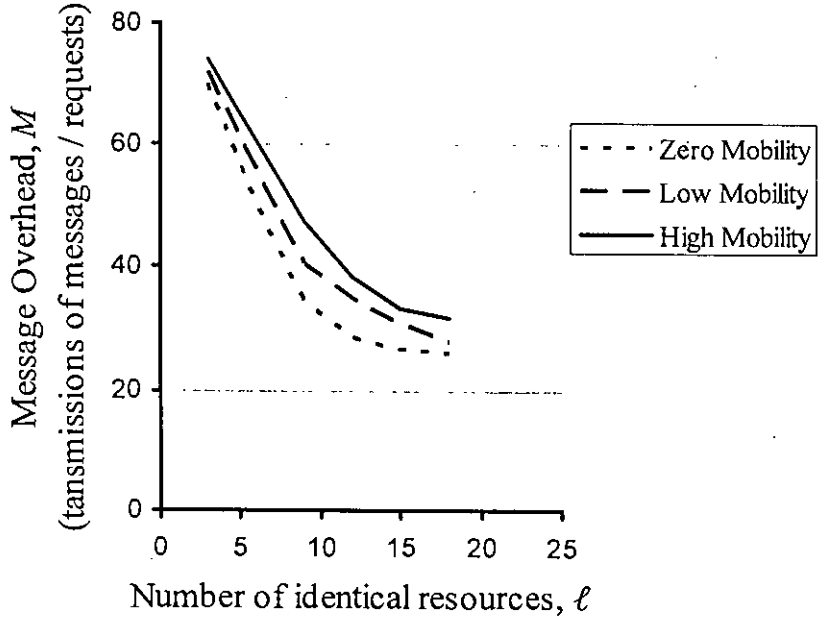
Plots in Figure 4.3 point to akin effect of the number of identical resources (ℓ) on *Message Overhead* (M) as shown in Figure 4.2. In addition, due to high rate of crash_merge in part (b) of Figure 4.3, the reduction in *Message Overhead* is more radical than that in part (a) of Figure 4.3. This is due to the fact that request messages require visiting more nodes under high crash_merge rate, when limited resources are available. As available resources increase, *Message Overhead* decreases due to visiting lesser number of nodes by request messages. However, when enough resources are available, the decreasing rate of *Message Overhead* is reduced as the number of available resources reaches to saturation against the crash_merge rate.

4.3.2. Average Waiting Time per CS Entry

We now study the impact of the request load (λ_{req}) on the *Average Waiting Time per CS Entry* (W) under various vulnerable ad hoc environments.



(a)



(b)

Figure 4.3. Effect of the number of identical resources (ℓ) on *Message Overhead* (M), for (a) λ_{crash_merge}=0.04, (b) λ_{crash_merge}=0.1, when node fails/merges, and λ_{req}=0.04.

4.3.2.1. Impact of request load

Figure 4.4(a) and (b) illustrate *Average Waiting Time per CS Entry* (W) against varying request load (λ_{req}) under ad hoc environment. The plots show that *Average Waiting Time per CS Entry* (against increasing request load) increases proportionally

under various mobility settings, when resources are limited (20%). Observe that the impact of request load on *Average Waiting Time per CS Entry* under zero mobility is closer to that that under high mobility, i.e. request load has small impact on the behavior of *Average Waiting Time per CS Entry* under high mobility. This is due to the fact that, as request load increases, the number of pending requests increases as well as requests messages require more traversals resulting in increase in delay in granting CS.

When resources increase to more (30%) than earlier (20%), increasing rate of *Average Waiting Time per CS Entry* (against increasing request load) reduces to nil under all mobility settings. As resources increases, request messages require less number of traversals in granting CS. During mobility, a slight increase in *Average Waiting Time per CS Entry* is realized, but increasing rate of *Average Waiting Time per CS Entry* remains constant as system has less number of pending requests due to availability of resources. Hence, when resources are 30% (of users) or more, *Average Waiting Time per CS Entry* is independent of request load.

4.3.2.2. Impact of crash–merge rate

Figure 4.5(a) and (b) illustrate *Average Waiting Time per CS Entry* (W) against crash–merge rate (λ_{crash_merge}) for a vulnerable ad hoc environment where at ℓ–1 nodes can fail or failed nodes may merge. We varied the values of λ_{crash_merge} increasing from 0.02 to 0.1.

We chose 0.1 for the high load value of λ_{crash_merge}, because at this rate, the network is much vulnerable. Similarly, we chose 0.02 as the lower value of λ_{crash_merge}, because at this rate, the network is much less vulnerable. Plots in Figure 4.5 have been shown for runs under two variants of the number of identical resources (ℓ), and request load (λ_{req}), i.e. in part (a), $\ell = 3$, $\lambda_{req} = 0.1$, and $\ell = 9$, $\lambda_{req} = 0.02$, in part (b). In part (a), we look the effect of λ_{crash_merge} on W closely, when system may have a large number of pending requests against the 10% available identical resources. In part (b), we observe the effect of λ_{crash_merge} on W, when system may have a small number of pending requests against the 30% available identical resources.

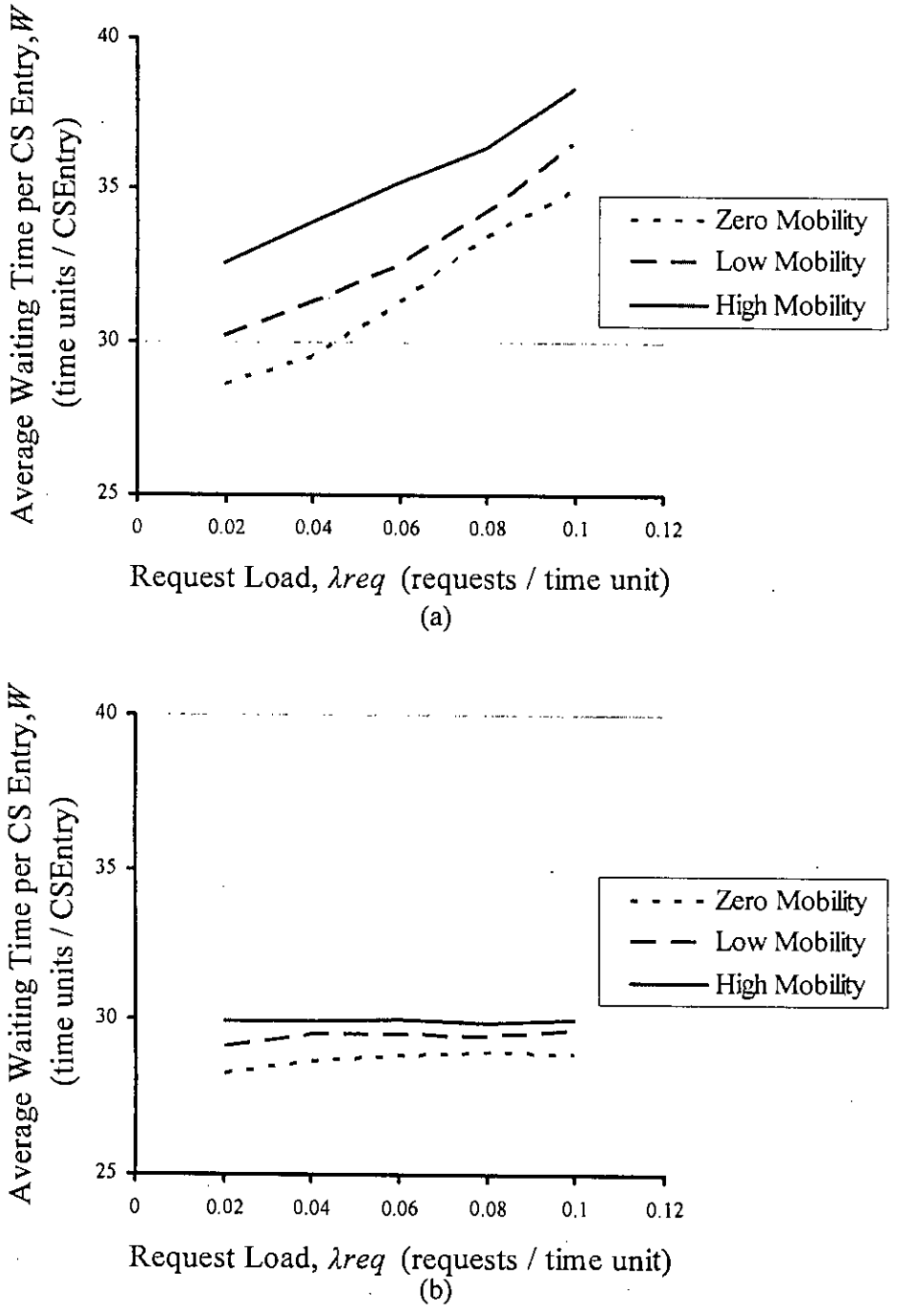
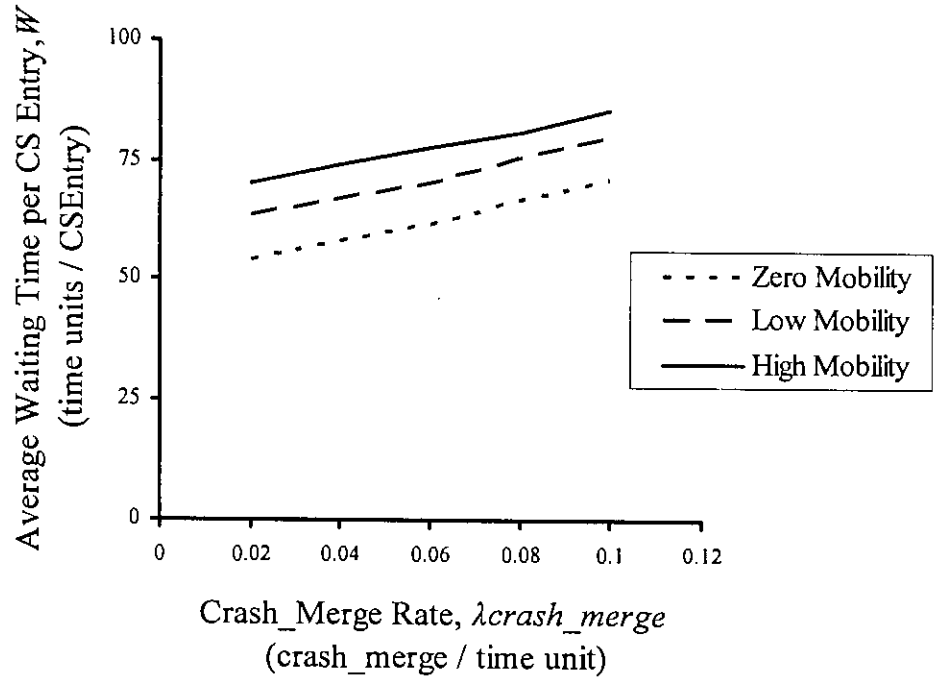
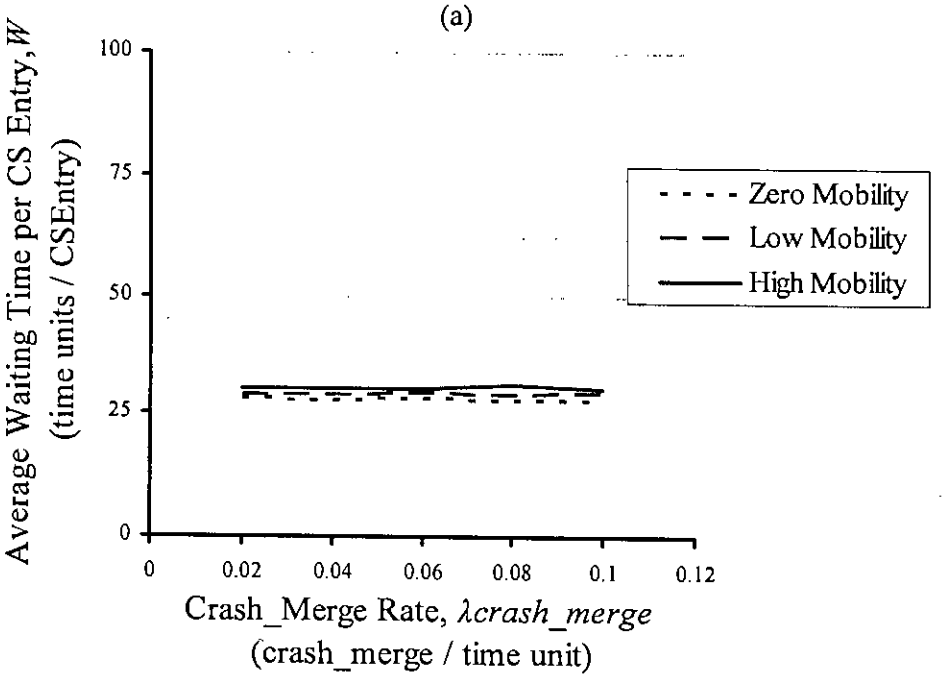


Figure 4.4. Effect of request load (λ_{req}) on *Average Waiting Time* (W), for (a) $\ell = 6$, (b) $\ell = 9$, when no failures or merges occur.



(a)



(b)

Figure 4.5. Effect of crash–merge rate (λ_{crash_merge}) on *Average Waiting Time* (*W*), for (a) $\ell = 3$, $\lambda_{req} = 0.1$, (b) $\ell = 9$, $\lambda_{req} = 0.02$.

From these plots, we can view that the impact of crash–merge rate on *Average Waiting Time per CS Entry* is quite similar to effect of request load on *Average Waiting Time per CS Entry* presented in Figure 4.4. This is due to the fact that request messages need to visit some nodes frequently, while some nodes remain unvisited with the increasing crash_merge rate, when system has limited (10%) resources. Therefore, some requests traverse for long time, increasing the average delay in granting CS. Mobility causes these requests to wait more to reach unvisited nodes, causing further increase in waiting for granting CS. While resources increase to 30%, request messages need to traverse lesser number of nodes. Hence, increasing rate of average delay in granting CS remain constant despite high mobility and high vulnerability.

4.4. Performance Comparison with KRL Algorithm

This section discusses the static and dynamic performance of the *ℓ–Exclusion* (LE) algorithm compared to the token–based *k–Reverse Link* (KRL) algorithm [140, 141] designed to operate on mobile ad hoc networks. To limit the illustrations, we depict some particular simulation–based scenarios for both algoithms under different mobility settings in a vulnerable ad hoc environment, where nodes can fail or failed nodes may merge.

The KRL simulation starts with nodes with indentifiers ranging from 0 to k–1 holding tokens. We initially adjusted the the height (a three–tuple representing the height of a node, such that links are directed from nodes with higher height towards nodes with lower heights) of each token holder to ensure that it had one incoming link. A connected graph whose initial edges were chosen at random, node heights and link directions were then initialized. However, all other parameters are identical to those described in the LE simulations.

4.4.1. Message Overhead

Figure 4.6 plots the average number of messages required to enter CS against values of crash–merge rate (λ_{crash_merge}) increasing from 0.01 to 0.05. Plots are shown for runs of LE simulation with number of identical resources, $\ell = 6$, and request load,

$\lambda_{req} = 0.04$, and for runs of KRL simulation with number of tokens, $k = 6$, and request load, $\lambda_{req} = 0.04$.

Figure 4.6 illustrates that *Message Overhead* increases as crash_merge rate increases in all simulations. But, in dynamic networks, the increasing rate of *Message Overhead* of KRL simulation is steeper than that of LE simulation. In static environment, KRL algorithm shows enhanced performance than LE algorithm at all rate of crash_merge. During low mobility, performance improvement of LE algorithm is realized at high crash_merge rate. As mobility increases, LE algorithm shows significant performance gain over KRL algorithm at medium to high crash_merge rate.

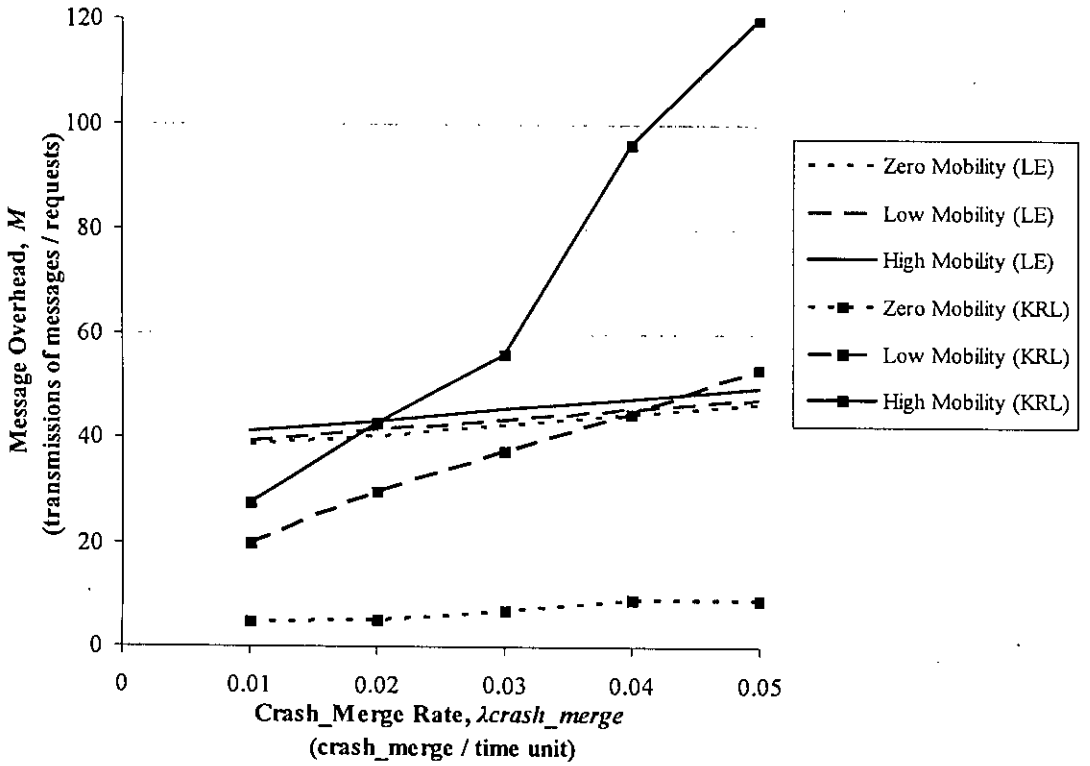


Figure 4.6. Crash–merge rate (λ_{crash_merge}) versus *Message Overhead* (M), when $\ell = 6$ (LE), or $k = 6$ (KRL), and $\lambda_{req} = 0.04$.

In static networks, KRL algorithm requires mobile nodes to transmit messages only to collect token by following directed links towards token holder nodes. Moreover, due to unfair pattern of token use, KRL algorithm can serve requesting nodes without following any order. For example, in a particular execution of KRL with a single

token, when two nodes are competing to enter CS, node that requested later than other can get the token along the token–returning path from token holder, and can enter CS earlier. Whereas, in similar execution, *Message Overhead* is greater in LE algorithm than it is in KRL algorithm, as LE algorithm ensures fairness in granting CS (see Theorem 3.3), it requires the potentially larger number of hops to collect enough votes to enter CS.

As mobility increases, KRL algorithm requires mobile nodes to transmit messages to maintain a DAG, and to adjust the direction of links among nodes towards token holder nodes. Moreover, due to unfair pattern of token use and high mobility, request paths may not be updated as execution continues. As a result, some tokens are used by a few nodes very frequently, causing few or no messages to be sent for these tokens, and allowing a few nodes to make tens of thousands of CS entries while the rest of the nodes make only thousands of CS requests. In addition, as KRL algorithm may lose some or all tokens due to sudden crash of nodes, the overall average is increased. On the contrary, during high mobility and high vulnerability, as LE algorithm requires no tokens and no directed links among nodes, it requires less number of messages to find unvisited nodes and to collect votes against any request. Hence, LE algorithm suits the spontaneous nature of the ad hoc environment well than KRL algorithm in terms of *Message Overhead.*

4.4.2. Average Waiting Time per CS Entry

Figure 4.7 shows the average delay in granting CS (after request) against values of crash–merge rate (λ_{crash_merge}). Plots are shown for runs of LE simulation and KRL simulation with parameters similar to those mentioned in Section 4.4.1.

From Figure 4.7, we perceive that *Average Waiting Time per CS Entry* increases roughly as crash_merge rate increases in all simulations. With the increasing rate of crash_merge, KRL algorithm performs better, in terms of *Average Waiting Time per CS Entry*, than LE algorithm in static environment. However, during mobility, LE algorithm attains significant performance improvement over KRL algorithm at all crash_merge rate.

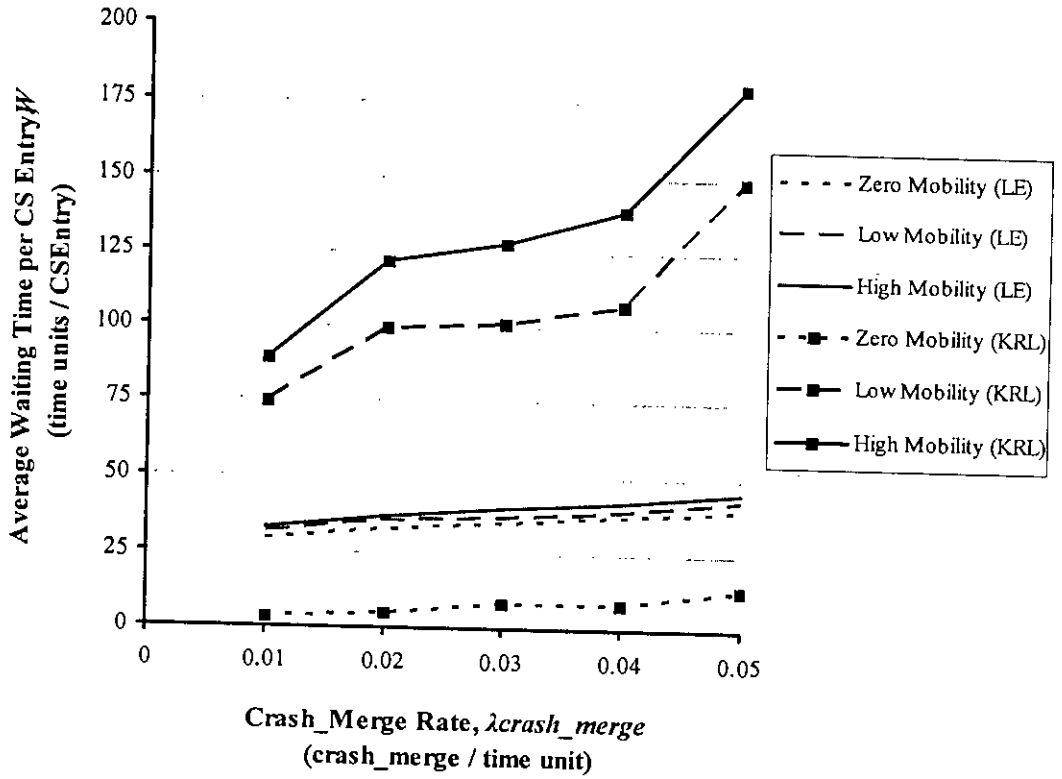


Figure 4.7. Crash–merge rate (λ_{crash_merge}) versus *Average Waiting Time per CS Entry* (W), when $\ell = 6$ (LE), or $k = 6$ (KRL), and $\lambda_{req} = 0.04$.

In static environment, the decrease in *Average Waiting Time per CS Entry* at all crash_merge rates for KRL simulation is caused by the unfair pattern of token used mentioned earlier. This skewed pattern of token–use results in zero waiting time for some CS entries bringing down the *Average Waiting Time per CS Entry* in static networks. In dynamic networks, when crash_merge rate increases, KRL simulation requires some of the requesting nodes to wait long in granting CS due to loss of directed links, loss of tokens, and sudden crash or recovery of nodes. On the other hand, as LE algorithm has no directed links or tokens, it performs significantly better, in terms of *Average Waiting Time per CS Entry*, than KRL algorithm under high mobility and high vulnerability.

4.4.3. Success Rate

This section show the impact of the crash–merge rate (λ_{crash_merge}) on the *Success Rate* (S) in a vulnerable ad hoc environment. Plots are shown for runs of LE simulation and

KRL simulation with parameters similar to those mentioned in Section 4.4.1. Plots in Figure 4.8 show that, in LE simulations, more than 90% of requesting nodes succeed to enter CS under variety of ad hoc environment. Whereas *Success Rate* of KRL algorithm reduces radically with the increasing crash_merge rate. Hence, increasing vulnerability of ad hoc network has no very small impact on *Success Rate* of LE algorithm, as compared to KRL algorithm. Note that with the increasing rate of crash_merge rate *Success Rate* of KRL algorithm degrades due to loss of tokens and directed links under high mobility and high vulnerability.

4.5. Chapter Summary

This chapter presented the simulation of the proposed algorithm. Description of simulation settings, parameters and mobility model has been followed by an intense investigation through an extensive simulation study. Simulation results confirmed that the proposed algorithm is quite effective to variety of operating conditions, and is highly adaptive to frequent and unpredictable topology changes due to loss of messages, link changes or failures or formations, sudden crashes or recoveries of at most ℓ–1 mobile nodes, under different mobility settings. In addition, performance of our algorithm has been compared to KRL algorithm. Our algorithm performs favorably at high crash/merge rate in terms of *Message Overhead*, particularly when nodes are mobile. The performance of our algorithm in terms of *Average Waiting Time per CS Entry* is remarkable, particularly under mobility and vulnerability.

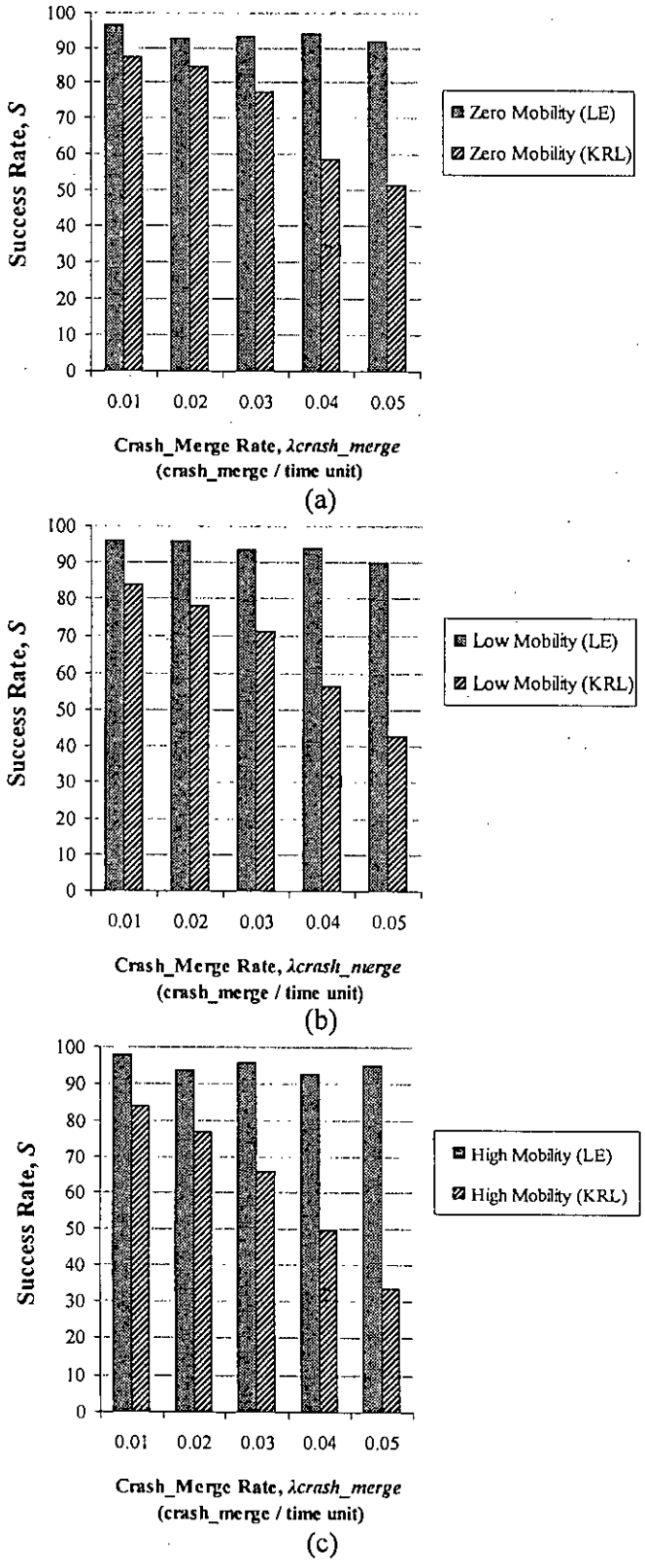


Figure 4.8. Crash–merge rate (λ_{crash_merge}) versus *Success Rate* (*S*), when $\ell = 6$ (LE), or $k = 6$ (KRL), and $\lambda_{req} = 0.04$, for (a) Zero, (b) Low, (c) High Mobility.

CHAPTER 5

CONCLUDING REMARKS

5.1. Major Contributions

In this research work, we present a consensus–based mobility–aware ℓ–exclusion algorithm that explicitly copes with arbitrary (possibly concurrent) topology changes associated with such networks. The algorithm is based on collection enough consensuses for a mobile node intending to enter CS, and uses diffusing computations for this purpose. Contributions of this thesis are summarized in the following:

- We devise a consensus–based mobility–aware *ℓ–Exclusion* algorithm that operates asynchronously, and accommodates arbitrary topology changes induced by node mobility. To enter the ℓ–entry CS, the algorithm requires mobile nodes to communicate only with their immediate neighbors chosen dynamically. Thus the algorithm suits well in ad hoc locale.

- The algorithm is fault–resilient in the sense that it can tolerate loss of messages, link changes or failures, sudden crashes or recoveries of at most ℓ–1 mobile nodes, and lossy communication links, as long as the link failures do not partition the communication networks.

- We develop an improved understanding of how to design and implement a distributed algorithm, such as *ℓ–Exclusion* algorithm, that accounts for the mobility found in an ad hoc network. In the context of *ℓ–Exclusion*, we observe that making subtle design changes in ad hoc settings can greatly affect the performance of our algorithm. In our context, this results in an algorithm that ensures that more than 90% of requesting nodes succeeds to access and to control CS under a variety of vulnerable operating conditions, where at most ℓ–1 node(s)

can crash independently or concurrently, and/or crashed node(s) may recover from failures.

- We present a formal verification to exhibit the correctness and the fairness of the algorithm, such as *ℓ–Exclusion*, *ℓ*–Lockout Avoidance and First–Come–First–Guaranteed properties, using theory of distributed computing.

- We present a thorough simulation study using PARSEC [5, 101] to analyze and evaluate the performance and the behavior of the algorithm as a function of request load, the number of identical resources, crash–merge rate. During simulation we consider some performance indices, such as *Message overhead per request*, *Average waiting time per request*, *Success rate*. We present the simulation results under following variants –
    - Zero, low, and high mobility without node failures/merges,
    - Zero, low, and high mobility with node failures/merges.

- We also study the performance of the *ℓ–Exclusion* algorithm compared to the *KRL* algorithm. Simulation results significantly demonstrates that, as compared to the *KRL* algorithm, the proposed algorithm is quite effective to variety of operating conditions, and is highly adaptive to frequent and unpredictable topology changes due to loss of messages, link changes or failures or formations, sudden crashes or recoveries of at most *ℓ*–1 mobile nodes, under different mobility settings.

5.2. Scope for Future Investigations

Based on our analyses on current design and the results of simulations presented in this thesis, we are under–way to investigate the extension of our frame works in the following directions:

- When nodes are mobile, frequent transmissions of messages for route discovery to originator node hurt the overall performance since transmissions and receptions of messages consume battery power and result in reduced energy savings, while battery power is an important resource to be managed carefully in ad hoc networks [146]. Hence, we are investigating to optimize the *ℓ–Exclusion*

algorithm with respect to communication complexity per CS (*Message Overhead*) under heavy request load without any compromise with impulsive behaviors of ad hoc networks.

- As the energy cost of a protocol is an important quantity to assess, further study is required to reduce the information size (data structures) associated with *Request* message and mobile nodes, and thus to minimize the energy utilization for handling these data structures.

- Future scopes also include employing some mechanisms to shrink the operational complexity of the *ℓ–Exclusion* algorithm to handle mobility and to tolerate intermittent failures more efficiently under various operational environments.

- To make our *ℓ–Exclusion* protocol more scalable, we are interested to impose upper bounds on *Message Overhead*, *Average Waiting Time* with respect to cardinality of the network (number of nodes), network load (request load), connectivity (transmission range), volume of identical resources, and crash–merge rate.

- The *ℓ–Exclusion* algorithm does not consider the network 2–partition. We hope that this research work will trigger the future investigations on *ℓ–Exclusion* problem for mobile ad hoc networks to a more fault–resilient direction by adapting this algorithm to tolerate network 2–partition.

- Our *ℓ–Exclusion* algorithm may be extended to devise an energy–efficient algorithm for asynchronous group mutual exclusion in mobile ad hoc networks. Note that asynchronous group mutual exclusion [3, 23, 57, 65, 66, 71, 72, 74, 75], a natural generalization of the classical mutual exclusion, was recently identified and solved by Joung [72] for static networks.

- This *ℓ–Exclusion* algorithm may be refined for SMANET [31]. Note that SMANET is a secure MANET system that accepts only those packets whose MAC addresses are in the Linux iptable firewall rules. Detailed of ip table set up and the performance of the firewall are available in [31].

REFERENCES

[1] Y. Afek, D. Dolev, E. Gafni, M. Merrit, and N. Shavit, "A Bounded First–In, First–Enabled Solution to the *l*–exclusion Problem", *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 3, pp. 939–953, May 1994.

[2] D. Agarwal and A. El Abbadi, "An Efficient and Fault–Tolerant Solution for Distributed Mutual Exclusion", *ACM Transactions on Computer Systems*, Vol. 9, No. 1, pp. 1—20, February 1991.

[3] K. Alagarsamy and K. Vidyasankar, "Elegant solutions for group mutual exclusion problem", Unpublished manuscript, 1999.

[4] B. R. Badrinath, A. Acharya, and T. Imielinski, "Structuring Distributed Algorithms for Mobile Hosts", *Proceedings of the 14th International Conference on Distributed Computing Systems*, May 1994, pp. 21–28.

[5] R. Bagrodia, R. Meyerr, et al., "PARSEC: A Parallel Simulation Environment for Complex System", UCLA Technical Report, 1997.

[6] R. Bagrodia and W. Liao, "Maisie: A Language for Design of Efficient Discrete–Event Simulation", *IEEE Transactions on Software Engineering*, April 1994.

[7] L. Bajaj, M. Takai, R. Ahuaja, K. Tang, R. Bagrodia, and M. Gerla, "Glomosim: A scalable network simulation environment", Technical Report 990027, UCLA Computer Science Department, May 1999.

[8] R. Baldoni, A. Virgillito, and R. Petrassi, "A Distributed Mutual Exclusion Algorithm for Mobile Ad–Hoc Networks", *Proceedings of the 7th IEEE International Symposium on Computers & Communications*, pp. 539–544, 2002.

[9] S. Banerjee and P. K. Chrysanthis, "A New Token Passing Distributed Mutual Exclusion Algorithm", *Proceedings of the 16th ICDCS*, pp. 717–724, 1996.

[10] S. Basagni, I. Chlamtac, and V. R. Syrotiuk, "A distance routing effect algorithm for mobility (DREAM)", *Proceedings of the 4th ACM/IEEE International Conference on Mobile Computing and* Networking, pp. 76–84, 1998.

[11] S. Basagni, I. Chlamtac, and A. Farago, "A generalized clustering algorithm for peer–to–peer networks", *Proceedings of the Workshop on Algorithmic Aspects of Communication,* Bologna, Italy, July 1997.

[12] B. Bellur, R. Ogier, and F. Templin, "Topology broadcast based on reverse–path forwarding (TBRPF)", IETF Internet–Draft draft–ietf–manet–tbrpf–00.txt, August, 2000.

[13] M. Benchaiba, A. Bouabdallah, N. Badache, and M. Ahmed–Nacer, "Distributed Mutual Exclusion Algorithms in Mobile Ad Hoc Networks: An Overview", *ACM Operating Systems Review*, Vol. 38, No. 1, pp. 74–89, 2004.

[14] V. Bharghavan, A. Demers, S. Shenker, L. Zhang, "MACAW: a media access protocol for wireless LANs", *Proceedings of ACM SIGCOMM'94*, pp. 212–225, 1994.

[15] A. Bouabdallah, "On mutual exclusion in faulty distributed systems", *ACM Operating Systems Review*, Vol. 28, No. 1, pp. 80–87, 1994.

[16] A. Bouabdallah, J. C. König, and M. B. Yagoubi, "A fault–tolerant algorithm for the mutual exclusion in real–time distributed systems", *Journal of computing and Information*, Vol. 1, No. 1, pp. 438–454, 1994.

[17] A. Bouabdallah and J. C. König, "A distributed algorithm for the mutual exclusion problem", *Parallel and Distributed Computing in engineering systems*, Tzafestas et al. (Editeurs), Elsevier Science Publisher B.V, North–Holland, pp. 285–290, 1992.

[18] J. Broch, D. Johnson, and D. Maltz, "The dynamic source routing protocol for mobile ad hoc networks", IETF Internet–Draft draft–ietf–manet–dsr–03.txt, October, 1999.

[19] 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", *Proceedings of the 4th ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 85–97, 1998.

[20] R. Bruno, M. Conti, and E. Gregori, "Bluetooth: Architecture, protocols and scheduling algorithms", *Proceedings of Hicss,* Maui, Hawaii, 2001.

[21] S. Bulgannawar, and N. H. Vaidya, "A Distributed *K*–Mutual Exclusion Algorithm", *Proceedings of 15th IEEE International Conference on Distributed Computing Systems*, pp. 153–160, May—June 1995.

[22] S. Bulgannawar and N. H. Vaidya, "A distributed k–mutual exclusion algorithm", *Tech. Rep. 94–066*, Department of Computer Science, Texas A&M University, College Station, November 1994.

[23] S. Cantarell, A. K. Datta, F. Petit, and V. Villain, "Token based group mutual exclusion for asynchronous rings", *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems*, 2001.

[24] J. Cao, J. Zhou, D. Chen, and J. Wu, "An Efficient Distributed Mutual Exclusion Algorithm Based on Relative Consensus Voting", *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium*, 2004.

[25] O. Carvalho and G. Roucairol, "On mutual exclusion in computer networks, Technical Correspondence", *Communications of the ACM*, Vol. 26, No. 2, pp. 146–147, Feb. 1983.

[26] R. Casteñeda and S. R. Das, "Query localization techniques for on–demand routing protocols in ad hoc networks", *Proceedings of the 5th ACM/IEEE International Conf. on Mobile Computing and Networking*, 1999, pp. 186–194.

[27] Y. Chang, M. Singhal, and M. Liu, "A Fault Tolerant Algorithm for Distributed Mutual Exclusion," *Proceedings of the 9th IEEE Symposium on Reliable Distributed Systems*, pp. 146–154, 1990.

[28] Y. Chen and J. L. Welch, "Self–Stabilizing Mutual Exclusion Using Tokens in Mobile Ad Hoc Networks", *Proceedings of the 6th Annual International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication*, pp. 34–42, September 2002.

[29] T. W. Chen, M. Gerla, and T. C. Tsai, "QoS routing performance in multihop Wireless networks", *Proceedings of IEEE ICUPC97*, San Diego, 1997.

[30] C. Chiang and M. Gerla, "Routing and multicast in multihop, mobile wireless networks", *Proceedings ICUPC '97*, pp. 546–551, 1997.

[31] C. E. Chow, P. J. Fong, and G. Godavari, "An exercise in constructing Secure Mobile Ad hoc Network (SMANET)", *Proceedings of the 18th IEEE International Conference on Advanced Information Networking and Application (AINA '04)*, 2004.

[32] M. Choy, "Robust Distributed Mutual Exclusion", *Proceedings of the 16th ICDCS*, pp. 760–767, 1996.

[33] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks", *Wireless Networks*, Vol. 1, No. 1, pp. 61–81, 1997.

[34] S. Corson and J. Macker, "Mobile ad hoc networking (MANET)", IETF RFC 2501, January 1999.

[35] M. S. Corson and A. T. Campbell, "Towards supporting quality of service in mobile ad–hoc networks", *Proceedings of the First Conference on Open Architecture and Network Programming*, San Francisco, April 1998.

[36] 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.

[37] E. W. Dijkstra and C. S. Scholten, "Termination detection for diffusing computations", *Information Processing Letters*, Vol. 11, No. 1, pp. 1–4, 1980.

[38] E. W. Dijkstra, "Self–Stabilizing Systems in Spite of Distributed Control", *Communication of the ACM*, Vol. 17, No. 11, pp. 643–644, 1974.

[39] E. W. Dijkstra, "Solution of a Problem in Concurrent Programming Control", *Communication of the ACM*, Vol. 8, No. 9, pp. 569, 1965.

[40] D. Dolev, E. Gafni, and N. Shavit, "Towards a non–atomic era: *l*–exclusion as a test case", *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, New York, pp. 78–92, 1988.

[41] D. Dolev, M. J. Fischer, R. Fowler, N. A. Lynch, and H. R. Strong, "An efficient Byzantine agreement without authentication", *Information and Control*, Vol. 52, No. 3, pp. 257–274, March 1982.

[42] D. Dolev, M. J. Fischer, R. Fowler, N. A. Lynch, and H. R. Strong, "An efficient Byzantine agreement without authentication", IBM Research Report RJ3428 (40914), Computer Science, IBM Research Division, Yorktown Heights, NY, March 22, 1982.

[43] R. Dube, C. D. Rais, K. Wang, and S. K. Tripathi, "Signal stability based adaptive routing (SSA) for ad–hoc mobile networks", *IEEE Personal Communications*, pp. 36–45, February 1997.

[44] A. Ephremides and T. V. Truong, "Scheduling broadcasts in multihop radio networks", *IEEE Transactions on Communications*, Vol. 38, No. 4, pp. 456–460, 1990.

[45] M. Fisher, N. Lynch, J. Burns, and A. Borodin, "Distributed FIFO allocation of identical resources using small shared space", *ACM Transactions on Programming Language and Systems*, Vol. 11, No. 1, pp. 90–114, Jan. 1989.

[46] M. Fischer, N. Lynch, J. Burns, and A. Borodin, "Resource Allocation with Immunity to Limited Process Failure", *Proceedings of the 20th Annual IEEE Symposium on Fountains of Computer Science*, pp. 78–92, 1979.

[47] W. Ford, *Computer Communications Security, Principles, Standard Protocols and Techniques*, Upper Saddle River, NJ: Prentice Hall, 1994.

[48] S. Fujita, M. Yamashita, and T. Ae, "Distributed *k*–Mutual Exclusion Problem and *k*–Coteries", *Proceedings of the 2nd International Symposium on Algorithms*, Lecture Notes in Computer Science 557, pp. 22–31, Berlin: Springer, 1991.

[49] E. Gafni and D. Bertsekas, "Distributed Algorithms for Generating Loop–Free Routes in Networks with Frequently Changing Topology", *IEEE Transactions on Communications*, Vol. 29, No. 1, pp. 11–18, May 1981.

[50] J. Garcia–Luna and M. Spohn, "Source tree adaptive routing (STAR) protocol", IETF Internet–Draft, October 1999.

[51] H. Garcia–Molina and D. Barbara, "How to Assign Votes in a Distributed Systems", *Journal of the ACM*, Vol. 32, No. 4, pp. 841–860, October 1985.

[52] H. Garcia–Molina, "Elections in a distributed computing system", *IEEE Transactions on Computers*, Vol. C–31, No. 1, pp. 48–59, 1982.

[53] M. Gerla and T.–C. Tsai, "Multicluster, mobile, multimedia radio network", *Wireless Networks*, pp. 255–265, 1995.

[54] D. K. Gifford, "Weight voting for replicated data", *Proceedings of the 7th ACM SIGOPS Symposium Operating Systems Principles,* Pacific Grove, CA, pp. 150–159, Dec. 1979.

[55] A. Gravey and A. Dupis, "Performance evaluation of two mutual exclusion distributed protocols via Markovian modeling", *Proceedings of the 6th IFIP Workshop Protocol Specification, Testing, Verification,* Montreal, P.Q., Canada, June 10–13, 1986.

[56] I. W. Group, "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications", IEEE specification. (http://standards.ieee.org/getieee802/download/802.11b–1999.pdf), Sep 1999. Work in Progress.

[57] V. Hadzilacos, "A note on group mutual exclusion", *Proceedings of the 20th Annual Symposium on Principles of Distributed Computing,* pp. 100–106, 2001.

[58] J. Helary, N. Plouzeau, and M. Raynal, "A Distributed Algorithm for Mutual Exclusion in an Arbitrary Network", *Computer Journal*, Vol. 31, No. 4, pp. 289–295, 1988.

[59] Y. C. Hsu, T. C. Tsai, and Y. D. Lin, "QoS Routing in multihop packet radio environment", *Proceedings of IEEE ISCC'98*, 1998.

[60] S. T. Huang, J. R. Jiang, and Y. C. Kuo, "K–Coteries for Fault–Tolerant K Entries to a Critical Section", *Proceedings of International Conference on Distributed Computing Systems*, pp. 74—81, May 1993.

[61] J.–P. Hubaux, J.–Y. Le Boudec, S. Giordano, M. Hamdi, L. Blazevic, L. Buttyan, and M. Vojnovic, "Towards mobile ad–hoc WANs: Terminodes", *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'00)*, Chicago, September 2000.

[62] J.–P. Hubaux, J.–Y. Le Boudec, S. Giordano, and M. Hamdi, "The nodes project: Towards mobile ad–hoc WANs", *Proceedings of the International Conference on Mobile Multimedia Communication* (*MOMUC99*), November 1999.

[63] A. Iwata, C. C. Chiang, G. Pei, M. Gerla, and T. W. Chen, "Scalable routing strategies for ad hoc wireless networks", *IEEE Journal on Selected Areas of Communications*, Vol. 17. No. 8, pp. 1369–1379, August 1999.

[64] P. Jacquet, P. Muhlethaler, A. Qayyum, et al., "Optimized link state routing protocol", IETF Internet–Draft draft–ietf–manet–olsr–02.txt, July 2000.

[65] P. Jayanti, S. Petrovic, and K. Tan, "Fair group mutual Exclusion", *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pp. 51–60, 2003.

[66] P. Jayanti, S. Petrovic, and K. Tan, "Fair group mutual exclusion (full paper)", Technical Report TR 2003 447, Dartmouth College Computer Science Department, 2003.

[67] J.–R. Jiang, S.–T. Huang, and Y.–C. Kuo, "Cohorts Structures Fault–Tolerant k Entries to a Critical Section", *IEEE Transactions on Computers*, Vol. 46, No. 2, February 1997.

[68] M. Joa–Ng and I. T. Lu, "A peer–to–peer zone–based two–level link state routing for mobile ad hoc networks", IEEE Journal on Selected Areas of Communications, Vol. 17, No. 8, pp. 1415–1425, August 1999.

[69] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek, and M. Degermark, "Scenario–based performance analysis of routing protocols for mobile ad–hoc networks", *Proceedings of the 5th ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 195–206, 1999.

[70] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks", *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, chapter 5, pp. 195–206, Kluwer Academic Publishers, Seattle, WA, 1996.

[71] Y. Joung, "Asynchronous group mutual exclusion", *Distributed Computing*, Vol. 13, pp. 189–206, 2000.

[72] Y. Joung, "Asynchronous group mutual exclusion", In Yehuda Afek, editor, *Proceedings of the 17th ACM Symposium on Principles of Distributed Computing*, pp. 51–60, June 1998.

[73] H. Kakugawa, S. Fujita, M. Yamashita, and T. Ae, "Availability of *k*–Coterie", *IEEE Transactions on Computers*, Vol. 42, No. 5, pp. 553–558, May 1993.

[74] P. Keane and M. Moir, "A simple local–spin group mutual exclusion algorithm", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 12, No. 7, pp. 673–685, July 2001.

[75] P. Keane and M. Moir, "A simple local–spin group mutual exclusion algorithm", *Proceedings of the Symposium on Principles of Distributed Computing*, pp. 23–32, 1999.

[76] Y.–B. Ko and N. H. Vaidya, "Location–added routing (LAR) in mobile ad hoc networks", *Wireless Networks*, Vol. 6, No. 4, pp. 307–321, July 2000.

[77] Y.–B. Ko and V. H. Vaidya, "Location–aided routing (LAR) in mobile ad hoc networks", *Proceedings of the 4th ACM/IEEE International Conf. on Mobile Computing and Networking*, pp. 66–75, 1998.

[78] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan, "A cluster based approach for routing in dynamic networks", *Proceedings of the ACM SIGCOMM Computer Communication Review*, pp. 372–378, 1997.

[79] A. Kumar, "Hierarchical quorum consensus: A new algorithm for managing replicated data", IEEE *Transactions on Computers*, vol. 40, no. 9, pp. 996–1004, Sept. 1991.

[80] L. Lamport, "The mutual exclusion problem: Part II–statements and solutions", *Journal of the ACM*, Vol. 33, No. 2, pp. 327–348, 1986.

[81] L. Lamport and P. M. Smith. "Synchronizing clocks in the presence of faults", *Journal of the ACM*, Vol. 32, No. 1, pp. 52–78, 1985.

[82] L. Lamport and P. M. Smith. "Byzantine clock synchronization", *Proceeding of the 3rd ACM Symposium on Principles of Distributed Computing*, New York, pp. 68–74, 1984.

[83] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System", *Communication of the ACM*, Vol. 21, No. 7, pp. 558–565, 1978.

[84] L. Lamport, "A new solution of Dijkstra's concurrent programming problem", *Communication of the ACM*, Vol. 17, No. 8, pp. 453–455, August 1974.

[85] S. B. Lee, G. S Ahn, and A. T. Campbell, "Improving UDP and TCP performance in mobile ad hoc networks", *IEEE Communication Magazine*, June 2001.

[86] S. J. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia, "A performance comparison Study of ad hoc wireless multicast protocols", *Proceedings of the IEEE Infocom*, March 2000.

[87] G. Le Lann, "Distributed Systems: Towards A Formal Approach", *IFIP Congress*, Toronto, pp. 155–160, 1977.

[88] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris, "Scalable location service for geographic ad hoc routing", *Mobicom00*, Boston 2000.

[89] J. P. Macker, V. D. Park, and M. S. Corson, "Mobile and wireless Internet services: Putting the pieces together", *IEEE Communication Magazine*, June 2001.

[90] N. Malpani, Y. Chen, N. H. Vaidya, and J. L. Welch, "Distributed Token Circulation in Mobile Ad Hoc Networks", *IEEE Transactions on Mobile Computing*, Vol. 4, No. 2, pp. 154–165, 2005.

[91] N. Malpani, Y. Chen, N. H. Vaidya, and J. L. Welch, "Distributed Token Circulation in Mobile Ad Hoc Networks", *Proceedings of the 9th International Conference on Network Protocol*, November 2001.

[92] M. Maekawa, "A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems", *ACM Transaction on Computer Systems*, Vol. 3, No. 2, pp. 145 – 159, 1985.

[93] K. Makki, P. Banta, K. Been, N. Pissinou, and E. Park, "A Token Based Distributed K–Mutual Exclusion Algorithm", *Proceedings of the 4th IEEE*

Symposium on Parallel and Distributed Processing, pp. 408—411, December 1992.

[94] Q. E. K. Mamun, M. Ali, S. M. Masum, and M. A. R. Mustafa, "A Two–Layer Hybrid Algorithm for Achieving Mutual Exclusion In Distributed Systems", *WSEAS Transactions on Systems*, Vol. 3, Issue 3, pp. 1193 – 1198, May 2004.

[95] MANET mailing list, ftp://manet.itd.nrl.navy.mil/pub/manet.archive, Discussion on applications for mobile ad–hoc networking with Subject: MANET application scenarios.

[96] S. M. Masum, A. A. Ali, and M. M. Akbar, "Design of An ℓ–Exclusion Algorithm for Mobile Ad Hoc Networks", To appear in *Proceedings of the First IEEE International Conference on Next–Generation Wireless Systems (ICNEWS'06)*, Dhaka, Bangladesh, January 2–4, 2006.

[97] S. M. Masum, A. A. Ali, and M. M. Akbar, "A Fault–Resilient ℓ–Exclusion Algorithm for Mobile Ad Hoc Networks", *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing 2005 (PacRim'05)*, Victoria, B.C., Canada, pp. 586 – 589, August 24–26, 2005.

[98] S. M. Masum, M. A. Al–Mamun, and M. M. Akbar, "A Fault–Tolerant Ring–Based Algorithm for Achieving Mutual Exclusion in Distributed Systems", *Asian Journal of Information Technology*, Vol. 4, No. 2, pp. pp. 185 – 193, February 2005.

[99] S. M. Masum, and M. M. Akbar, "An Optimal and Fault–Tolerant Solution for Distributed Mutual Exclusion", *Proceedings of the 7th International Conference on Computer and Information Technology*, pp. 698 – 704, 26–28 December 2004.

[100] S. M. Masum, M. A. Al–Mamun, K. T. Islam, S. M. S. Mostafa, and M. M. Akbar, "A Fault–Tolerant Ring–Based Algorithm for Achieving Mutual Exclusion in Distributed Systems", *Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems*, pp. 205 – 210, November 9–11, 2004.

[101] R. A. Meyer, "PARSEC User Manual", UCLA Parallel Computing Laboratory, http://pcl.cs.ucla.edu.

[102] S. Mishra and P. Srimani, "Fault–tolerant mutual exclusion algorithms", *Journal of Systems software*, Vol. 11, No. 2, pp. 111–129, February 1991.

[103] J. Misra, "Detecting termination of distributed computations using markers", *Proceedings of the 2nd ACM Annual Symposium on Principles of Distributed Computing*, pp. 237–249, August 1985.

[104] M. Mizuno, M. L. Neilsen, and R. Rao, "A token based distributed mutual exclusion algorithm based on quorum agreements", *Proceedings of the 11th International Conference on Distributed Computing Systems*, pp. 361–368, May 20–24, 1991.

[105] F. Mueller, "Prioritized Token–Based Mutual Exclusion for Distributed Systems", *Proceedings of the Joint Workshop on Parallel and Distributed Real Time System*, pp. 72–80, 1997.

[106] M. Naimi and M. Trehel, "An improvement of the *logn* Distributed Algorithm for Mutual Exclusion", *Proceedings of the 7th IEEE International Conference on Distributed Computing Systems*, pp. 371–375, 1987.

[107] M. L. Neilsen and M. Mizuno, "A DAG–Based Algorithm for Distributed Mutual Exclusion", *Proceedings of the 11th International Conference on Distributed Computing Systems*, pp. 354–360, May 20–24, 1991.

[108] Network simulator – NS–2, http://www.isi.edu/nsnam/ns.

[109] NFS Wireless and Mobile Communications Workshop, Northern Virginia, March 1997.

[110] OPNET Modeler, http://www.opnet.com/products/modeler/home.html.

[111] E. Pagani and G. P. Rossi, "Reliable broadcast in mobile multihop packet networks", *Proceedings of the 3rd ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 34–42, 1997.

[112] V. Park and M. S. Corson, "Temporally–ordered routing algorithm (TORA) Version 1 Functional Specification", IETF Internet–Draft draft–ietf–manet–tora–spec–02.txt, October, 1999.

[113] V. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks", *Proceedings of the INFOCOM'97*, pp. 1405–1413, 1997.

[114] C. Perkins, E. Royer, and S. Das, "Ad hoc on demand distance vector (AODV) routing", IETF Internet–Draft draft–ietf–manet–aodv–06.txt, July, 2000.

[115] C. Perkins, *Ad Hoc Networking*, Reading, MA: Addison–Wesley, 2000.

[116] C. Perkins (Ed.), "IP mobility support", IETF RFC 2002, October 1996.

[117] C. E. Perkins and P. Bhagwat, "Highly dynamic destination–sequenced distance–vector routing for mobile computers", *Proceedings of the ACM SIGCOMM Symposium on Communication, Architectures and Protocols*, pp. 234–244, 1994.

[118] C. E. Perkins and E. M. Royer, "Ad–hoc on–demand distance vector routing", *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, 1999.

[119] R. Ramanathan and M. Streenstrup, "Hierarchically–organized, multihop mobile wireless networks for quality–of service support", *Mobile Networks and Applications*, January 1998.

[120] K. Raymond, "A Tree based Algorithm for Distributed Mutual Exclusion", *ACM Transactions on Computer Systems*, Vol. 7, No. 1, pp. 61—77, February 1989.

[121] K. Raymond, "A Distributed Algorithm for Multiple Entries to a Critical Section", *Information Processing Letters*, vol. 30, pp. 189—193, February 1989.

[122] M. Raynal, "Prime numbers as a tool to design distributed algorithms", *Information Processing Letters*, Vol. 33, No. 1, pp. 53–58, October 1989.

[123] G. Ricart and A. K. Agrawala, "Author response to 'on mutual exclusion in computer networks', by Carvalho and Roucairol", *Communication of the ACM*, Vol. 26, No. 2, pp.147–148, Feb. 1983.

[124] G. Ricart and A. K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks", *Communications of the ACM*, Vol. 24, No. 1, pp. 9—17, January 1981.

[125] E. M. Royer and C. E. Perkins, "Multicast operation of the ad–hoc on demand distance vector routing protocol", *Proceedings of the 5th ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 207–218, 1999.

[126] E. M. Royer and C.–K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks", *IEEE Personal Communication Magazine*, pp. 46–55, April 1999.

[127] B. Sanders, "The information structure of distributed mutual exclusion algorithms", *ACM Transactions on Computer Systems*, Vol. 5, No. 3, pp. 284–299, August 1987.

[128] D. Shou and S. D. Wang, "An Efficient Quorum Generating Approach for Distributed Mutual Exclusion", *Journal of Information Science and Engineering*, Vol. 9, pp. 201–227, June 1993.

[129] M. Singhal, "A Heuristically–Aided Algorithm for Mutual Exclusion in Distributed Systems", *IEEE Transactions on Computers*, Vol. 38, No. 5, pp. 651–662, May 1989.

[130] 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, January 1992.

[131] I. Stojmenovic´, "Location updates for efficient routing in ad hoc networks", Chapter 23, *Handbook of Wireless Networks and Mobile Computing,* John Wiley & Sons Inc., 2002.

[132] I. Suzuki and T. Kasami, "A Distributed Mutual Exclusion Algorithm", *ACM Transactions on Computer Systems*, Vol. 3, No. 4, pp. 344—349, 1985.

[133] T. C. M. Project, "The CMU Monarch Project's wireless and mobility extensions to ns", August 1998. Available from http://www.monarch.cs.cmu.edu.

[134] Z. Tang and J. J. Garcia–Luna–Aceves, "Hop reservation multiple access for multichannel packet radio networks", *Computer Communications*, Vol. 23, No. 10, pp. 877–886, May 2000.

[135] The REAL network simulator,http://www.cs.cornell.edu/skeshav/real/overview.htm.

[136] R. H. Thomas, "A majority consensus approach to concurrency control", *ACM Trans. Database Systems,* Vol. 4, no. 2, pp. 180–209, June 1979.

[137] G. Varghese, "Self–stabilization by computer flushing", *Proceedings of the 13th ACM International Symposium on Principles of Distributed Computing*, August 1994.

[138] S. Vasudevan, J. Kurose, and D. Towsley, "Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks", *Proceedings of the 12th International Conf. Network Protocols*, pp. 350–360, October 2004.

[139] J. E. Walter, J. L. Welch, and N. H. Vaidya, "A Mutual Exclusion Algorithm for Ad Hoc Mobile Networks", *Wireless Networks*, Vol. 7, pp. 585–600, 2001.

[140] J. E. Walter, G. Cao, and M. Mohanty, "A *k*–Mutual Exclusion Algorithm for Wireless Ad Hoc Networks", *Proceedings of the 1st Annual Workshop on Principles of Mobile Computing*, 2001.

[141] J. E. Walter, "A *k*–mutual exclusion algorithm for ad hoc mobile networks", Technical Report TR00–022, Texas A&M University, College Station, TX 77843–3112, 2000.

[142] J. E. Walter and S. Kini, "Mutual exclusion on Multi–Hop, Mobile Wireless Networks", Technical Report TR97–014, Texas A&M University, College Station, TX 77843–3112, December 1997.

[143] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides, "Energy–efficient multicast of session traffic in bandwidth and transceiver–limited wireless", *Cluster Computing,* 2002.

[144] J. Wu, M. Gao, and I. Stojmenovic′, "On calculating power–aware connected dominating sets for efficient routing in ad hoc wireless networks", *Proceedings of the ICPP '01,* Valencia, Spain, September 3–7, 2000.

[145] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a Library for Parallel Simulation of Large–scale Wireless Networks", *Proceedings of the 12th Workshop on Parallel and Distributed Simulations*, Alberta, Canada, May 1998.

[146] R. Zheng and R. Kravets, "On–demand power management for ad hoc networks", *Ad Hoc Networks*, Vol. 3, No. 1, pp. 51–68, January 2005.