

**STRUCTURING DISTRIBUTED ALGORITHMS AND
SERVICES FOR NETWORKS WITH MOBILE HOSTS**

BY ARUP ACHARYA

**A dissertation submitted to the
Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of requirements
for the degree of
Doctor of Philosophy
Graduate Program in Computer Science**

**Written under the direction of
Dr. B. R. Badrinath
and approved by**

New Brunswick, New Jersey

May, 1995

ABSTRACT OF THE DISSERTATION

STRUCTURING DISTRIBUTED ALGORITHMS AND SERVICES FOR NETWORKS WITH MOBILE HOSTS

by ARUP ACHARYA

Dissertation Director: Dr. B. R. Badrinath

Integration of mobile computers within existing data networks introduces new issues in the design of distributed algorithms and services. Location of a mobile host changes with time, and so the message count of a distributed algorithm should account for the "search" necessary to locate mobile participants. Further, mobile hosts are faced with resource constraints not commonly encountered by their tethered counterparts, viz. a low-bandwidth connection to the rest of the network, and tight restrictions on power consumption.

This dissertation introduces a system model for networks with mobile hosts. To bridge the resource disparity between mobile and static hosts, we propose a two-tier principle for structuring distributed algorithms in this model. We also propose that location-management of mobile participants be integrated with algorithm design.

We first consider a simple, yet fundamental distributed algorithm: a logical ring with a token circulating amongst participants, and restructure it for servicing token requests from mobile hosts. Second, we tackle the problem of delivering a multicast message to mobile recipients from exactly-one location. Third, we present a checkpointing algorithm to record a consistent global state of a distributed application executing on mobile hosts.

ACKNOWLEDGMENTS

First and foremost, I deeply thank Professors Tomasz Imielinski and B. R. Badrinath for their work on mobile computing, providing the necessary impetus for this dissertation. This dissertation would not have been possible without the constant encouragement, camaraderie, patience and guidance of B. R. Badrinath. I am also grateful to Tomasz Imielinski for his active participation in defining many portions of this work.

I thank Professors Naftaly Minsky and Dan Duchamp for agreeing to participate on my thesis committee. I am indebted to Professor Barbara Ryder for her support and encouragement during the course of my graduate study.

I should also acknowledge the financial support provided by the Computer Science Department for my graduate studies.

I have benefitted greatly from the critiques and suggestions offered by other members of the DATAMAN group — Ajay, Girish, Vish, Aashu and Monish. I would also like to thank my fellow graduate students, past and present, for their companionship — Partha, Jana, Darshan, Vipul, Sriram, Shiv and Jaikumar.

Finally, it is unlikely that this dissertation would have been completed had it not been for the fine squash courts next door.

DEDICATION

To Amal, Nupur, Belghoria and Bepin Pal

TABLE OF CONTENTS

ABSTRACT OF THE DISSERTATION	ii
ACKNOWLEDGMENTS	iii
DEDICATION	iv
1 Introduction	1
1.1 Related Work on Mobile Computing	4
1.2 Dissertation overview	8
2 The System Model for Host Mobility	10
2.1 Characterizing features of the model	13
2.2 The two-tier principle	16
2.3 Chapter summary	19
3 Structuring Distributed Algorithms for Mobile Hosts	20
3.1 Prevalent model of distributed computation	21
3.2 Impact of mobility	23
3.3 Applying the two-tier principle to Lamport's algorithm	28
3.4 Structuring a token-based logical ring for mobile hosts	30
3.5 Data replication: a mechanism for location management	41
3.6 Chapter summary	44
4 Checkpointing Distributed Applications on Mobile Computers	45
4.1 Consistent global state	46
4.2 Motivation	47
4.3 The <i>two phase</i> rule	51
4.4 The checkpointing algorithm	54
4.5 Discussion	58
4.6 Chapter summary	59
5 Delivering Multicast Messages to Mobile Hosts	61
5.1 Design Considerations	62

5.2 Multicast Protocols	68
5.3 Ordered delivery of multicast messages	84
5.4 Chapter summary	85
6 Location Management for Multicast Groups of Mobile Hosts	87
6.1 Overview	87
6.2 Attributes of multicast groups	90
6.3 Host view membership protocol	95
6.4 Message delivery to a multicast group G	102
6.5 Correctness sketch	104
6.6 Future Extensions	107
7 Conclusions	109
Bibliography	113
Vita	118

LIST OF FIGURES

2.1	The system model for supporting host mobility	11
3.1	Effects of mobility on a unidirectional logical ring	26
4.1	The problem with independently recorded local checkpoints	52
5.1	Effect of mobility on delivery of multicast messages	64
5.2	Handoff	71

© 1995

Arup Acharya

ALL RIGHTS RESERVED

Chapter 1

Introduction

Mobile Computing represents a new paradigm that aims to provide continuous network connectivity to users regardless of their location. A wide spectrum of portable, personal computing devices have recently been introduced in the market that range from laptop computers to handheld personal digital assistants. Coupled with the advent of wireless networking, this has given rise to a new style of computing wherein the computer can move with the user and yet maintain its network connections.

The emergence of computer networking allowed isolated computers to share resources giving rise to distributed computing. Computer networking eliminated the physical isolation between computers, and enabled users connected to the system from one computer to use the resources available at another. This paved the way for distributed computing where a collection of networked computers cooperate to achieve a common goal. However, the access points to such a distributed system were still tethered machines and a user did not have the flexibility to move while accessing such a system.

Mobile computing extends distributed computing in a direction where the services of such a system are available to an user regardless of location and more importantly, changes in location. Wireless networking eliminated the need to remain tethered to a wired, static infrastructure and yet avail of the services of a distributed system. A distributed system with mobile hosts thus consists of a wired infrastructure of static hosts (representing a conventional distributed system) that connects areas of wireless coverage ('cells') to mobile computers.

Mobility of computers is not obtained for free. The ability to be mobile introduces a set of new constraints that are not associated with the desktop computers. The goal of mobile computing is therefore, to provide users with a comparable level of

service as would be available to them from a distributed system of static computers, without compromising their ability to move.

Solutions to communication and synchronization problems in distributed systems have so far been designed for networks comprising solely of static hosts. In systems with static hosts, connectivity of the underlying network does not change in the absence of link and/or host failures. On the other hand, *mobile hosts* are capable of moving between different locations (“roaming”) while maintaining their connection to the network, e.g. via a cellular connection or a wireless LAN. Consequently, a mobile host must first be *located* within the network before a message can be delivered to it. Second, as hosts move, physical connectivity of the network changes. Hence, any *logical structure*, which many distributed algorithms exploit, cannot be statically mapped to a set of physical connections within the network. Third, mobile hosts have severe *resource constraints* in terms of limited battery life and often operate in a “doze mode” or entirely disconnect from the network to prolong battery life. Fourth, communication between a mobile host and the rest of the network occurs via a *wireless* medium with a significantly lower bandwidth than wired links; additionally, message transmission and reception at a mobile host consumes battery power, which is a critical resource. Next, mobile hosts are not equipped with *stable* storage; disk storage at a mobile host is not considered stable due to vulnerability of such devices to loss, theft and physical damage (arising from their personalized nature of usage and portability). Lastly, mobile computers utilize processors and storage systems that trade computing, communication speed and storage capacity for reduced power consumption and portability (weight and size). Thus, though a mobile computer can be taken anywhere, it delivers performance an order of magnitude less than their desktop counterparts. These aspects are characteristic of mobile computing and need to be addressed before distributed systems can be extended to include mobile hosts.

This thesis considers how distributed algorithms and services should be structured for mobile hosts. A fundamental problem in distributed systems is that of providing *mutually exclusive* access to a shared resource to a group of competing

entities such that there is no centralized permission granting authority. In the context of a mobile computing environment, the competing entities are mobile hosts. We show the drawbacks of applying existing solutions (designed implicitly for static hosts) to a distributed system with mobile hosts, and propose a *two tier* principle to structure distributed algorithms for mobile hosts, taking into consideration the unique resource constraints of this environment. We then introduce different *location management* strategies to handle mobility of participants requesting mutual exclusion. Our aim here is to show that distributed algorithms for mobile hosts should be designed to explicitly cope with varying location of mobile participants and their associated resource constraints, instead of relying on the underlying communication network to route messages to mobile hosts (thereby excluding mobility considerations from algorithm design).

The second problem that we investigate in this thesis arises solely due to mobility, viz. how should multicast messages be delivered to mobile hosts so that a message is delivered to a destination from exactly-one location. Reliable delivery of a multicast message requires that the message be buffered at different locations, for delivery to potential mobile recipients at each location. Since a mobile host may move between different locations, it is possible that it receives copies of the same message from multiple locations. It is also possible that depending on its mobility pattern (i.e. it moves between locations which do not have a copy of the message for delivery), the mobile host may miss receiving the message altogether. Besides ensuring exactly-once delivery, a solution to this problem must also respect the resource constraints of mobile hosts. The first solution that we present allows a multicast message to be addressed to any arbitrary group of mobile hosts. We then refine this solution to allow multicasting only to pre-defined groups of mobile hosts; we present a protocol to track the location of group members and this aggregate location information determines which locations should receive a copy of a multicast message addressed to a specific group.

The third section of this thesis focuses on the problem of recording a consistent global checkpoint of a distributed application executing on mobile hosts. For

reliability, it is vital that global state of such applications be checkpointed from time to time. Given that many of mobile computers will be personalized portable computing devices, such as PDAs and personal communicators, they will be used for multiperson interaction that require group communication between mobile computers; an user must be assured that a lost or misplaced device will not jeopardize the entire interaction. The problem of recording a consistent global checkpoint has been widely studied in the context of distributed systems with fixed hosts. With mobile hosts however, the problem takes a totally new turn due to varying location of a mobile host, lack of stable storage to store local checkpoints and disconnection of mobile hosts.

The unifying theme of this thesis is to provide a comparable level (and set) of services to users of mobile computers as would be available to users of desktop machines without compromising on user mobility. To this end, we selected three fundamental problems in distributed systems with static hosts, viz. selecting one out of many competing participants (mutual exclusion), one-to-many communication represented as a multicast service and checkpointing mechanisms to provide reliability. We observe that varying location and voluntary disconnection of mobile hosts, lack of stable storage, constraints on power consumption and low bandwidth of wireless connections imply that existing solutions (if any) for static hosts are not viable for a mobile computing environment, and thus there is a need for algorithms that solve these problems taking into account, the specific constraints of this environment.

1.1 Related Work on Mobile Computing

Given the relative infancy of mobile computing, it is not surprising that there has been little prior work on structuring distributed algorithms per se for mobile hosts. Research has primarily focussed on providing network-layer protocols to route data packets to mobile hosts regardless of location, and on data management issues for low-powered clients that access data stored within the static infrastructure using a low-bandwidth connection. Below, we summarize current research in this field:

Extending IP protocols to handle host mobility Hosts in the Internet are associated with addresses, which determine the route a data packet takes to reach a particular destination. In effect, the address determines the “location” of a host vis-a-vis the rest of the network. However, with mobile hosts, this is no longer valid since the location of a mobile host changes. If the address associated with a mobile host remains the same regardless of its location, then this address can no longer be used to route a packet to it (since this address cannot encode the current location of the mobile host). On the other hand, if a mobile host is assigned a new address reflecting its current location after every move, then all other entities in correspondence with this host need to be informed of changes in its address. Various schemes have been proposed to extend IP protocols and routing schemes for mobile hosts [18, 21, 40, 61, 26, 63, 42], and a comparison of the key approaches can be found in [54].

Effects of host mobility on transport and higher layers One of the aims for providing network-layer support for mobile hosts is that transport-layer protocols do not need to be aware of host mobility, i.e. provide functional transparency to TCP/IP. However, a recent study [22] shows that active TCP connections, with Mobile-IP [40] at the network layer, show considerable degradation in performance: mobility of a host from one location to another increases delays and packet losses while the network learns how to route data to the host’s new location. TCP interprets these events to be a result of network congestion and throttles further packet transmission, thereby leading to further drop in throughput. As a remedy, [22] suggests that TCP be made aware of host mobility. In addition to the transport layer, [12] proposes that host mobility also be made explicit at the application layer as well.

File systems for mobile users The primary motivation behind designing file systems for mobile users is to support *disconnected* operation [46, 32, 34]. Prior to disconnection, a user caches files from static servers that (s)he needs to use while disconnected from the network. When the user reconnects, modifications to the files that were updated by the user in disconnected mode are propagated to the

server. The three main issues here are: (1) hoarding, i.e. which files should be brought into the mobile client's local store prior to disconnection, (2) emulation, i.e. satisfying file requests from the local cache while disconnected, and what actions to be taken if such a request cannot be satisfied, and (3) reintegrating all modified files at the mobile client with the static server, on reconnection. Another motivation for designing file systems to handle mobility is to minimize synchronous operations and allow mobile clients to determine the level of consistency desired between the client copy and copies at the server(s) [60]. A third approach is the design of the storage subsystem at the mobile computer using flash memory to reduce power consumption [53].

Operating systems Research in this area has been fueled by tight constraints on power consumption at mobile computers [17, 62]. [17] presents enhancements to the Unix kernel for power management and checkpointing system state (to protect the user from undesirable state loss when the battery is low). Besides power management, the operating systems for personal digital assistants must also deal with multiple, different communications technologies with widely varying latency, bandwidth, connectivity characteristics and usage cost, and be able to discover and interact with nearby devices and services in a location-specific fashion [62]. Guided by limitations of wireless bandwidth and power, [9] proposes an inter-process communication mechanism that provides (a) an agent within the static infrastructure that stores and filters messages on behalf of a mobile user (b) a hierarchical structure to messages, with immediate-delivery data at the top, and (c) the ability to automatically start/resume a server program at the mobile host on delivery of a message (to the server).

Data Management Challenges to the database field due to the advent of mobile computing were first identified in [35], namely controlling the flood of location updates arising out user mobility and the need for a new paradigm for answering queries where acquisition of location information is integrated with the query evaluation process. The issue of optimizing queries from a mobile computer to a database server on the static network is discussed in [5]: this involves optimization

along two different axes, viz. energy spent by the mobile client to process the query locally and overall server throughput. The constraints on power consumption has also lead to research on the organization of data that is broadcast to a large number of users within a “macrocell” [39]. Mobility of users is not a consideration here. Instead, the competing criteria for optimization are power consumption at a mobile computer and the time it takes to select one item from a set of broadcast data items. Allocation and replication of data items between a mobile computer and a static server based on the read-write pattern is investigated in [13, 33]. On a related note, [16] presents different cache invalidation strategies for mobile clients that cache frequently accessed data items from a local static server; the choice of a particular strategy is governed by whether a mobile client is a “sleeper” (disconnected most of the time) or a “workaholic” (stays connected to the network most of the time). General discussions on the impact of mobility on data management can be found in [38, 36, 7].

Location management for mobile users To manage the changing location of mobile users, [10] proposes a hierarchy of regional directories, where each directory is based on a decomposition of the network into regions. A change in location of a mobile user updates “nearby” directories, which point to the user’s new location. A forwarding pointer is left at the old location of the user. Searches that originate at “remote” directories are guided by this pointer to the user’s current location. [14] examines strategies to reduce search costs and control the volume of location updates by employing user profiles. User profiles are used to define partitions and a location update is generated only when a user crosses partitions. The problem of location management is split into three components in [48]: search for the current location of an user, location updates after a move and location updates after a successful search. Multiple strategies are presented for each sub-problem which are then combined to formulate different location-management strategies. A different twist to the problem is explored in [58], viz. providing location information users may constitute an invasion of privacy of users, and therefore, users should be able

to control how precisely their individual locations are revealed to location-based applications.

Fault tolerance There has been very limited research work in this area. [47] briefly discusses the issues involved in recovery from failures in a mobile environment, and suggests that the static infrastructure be used to backup the state of mobile hosts. The reverse problem, viz. failure of a static host that serves as an access point to the rest of the static infrastructure, can result in loss of communication with mobile hosts. As a solution, [4] suggests replicating the services of such a static host amongst several secondary static hosts so that a mobile host can switch its access point to one of the secondaries in case the primary fails.

1.2 Dissertation overview

The next chapter presents the system model for a network with mobile hosts, and proposes a *two tier* principle to handle the constraints associated with such networks.

Chapter 3 discusses the impact of host mobility on the design of distributed algorithms. The problem of distributed mutual exclusion and two classical solutions to this problem, are used to illustrate why distributed algorithms need to be “mobile aware”. The proposed two-tier principle is then applied to restructure the two algorithms to meet the constraints of mobile computing. In conjunction with the two-tier principle, we also develop strategies to track the location of “migrant” MHs, i.e. MHs that invoke a service from the fixed network at one location, but move between several locations before receiving the result. Therefore, to deliver the desired result to a migrant MH, a distributed algorithm must now explicitly incorporate location management of migrant MHs in its design. Lastly, we present an alternative approach based on data replication to provide distributed mutual exclusion for mobile hosts.

In chapter 4, we consider the problem of recording a consistent global checkpoint of a distributed application executing on mobile hosts, and present a distributed

checkpointing scheme that takes care of changing locations and disconnection of one or more mobile participants, and the lack of stable storage at a mobile computer.

Chapters 5 and 6 presents protocols for delivery of multicast messages to mobile recipients. The interplay of host mobility with delivery of multicast messages is first identified as follows: a multicast message may be delivered to the same recipient from multiple locations or may not be delivered at all, contingent on the recipient's mobility. We then present algorithms for delivering multicast messages to mobile destinations from *at least one* location, and from *at most one* location, and combine the two algorithms for message delivery from *exactly-one* location. Chapter 6 introduces the notion of *multicast groups* of mobile hosts, and associates a "host view" with each group; the host view represents the aggregate location information of group members and changes dynamically as members change their locations. A host-view membership protocol is then combined with the exactly-once multicast algorithm, thus incorporating location management of multicast groups into delivery of multicast messages.

Chapter 2

The System Model for Host Mobility

The term “mobile” implies *able to move while retaining its network connections* [40]. A host that can move while retaining its network connections is a *mobile host* (MH). To allow migration of mobile hosts within a network of fixed hosts, the fixed network is augmented with *mobile support stations* (MSS) that serve as access points for mobile hosts to connect to the fixed network.

A MH needs a untethered medium of communication to maintain its network connections while on the move and so, mobile hosts connect to MSSs via wireless links. A MSS is a fixed host that provides wireless coverage to MHs within a predefined logical or geographical area, called a *cell*. A MH can *directly* communicate with a MSS (and vice versa) only if the MH is physically located within the cell serviced by the MSS. At any given instant of time, a MH may (logically) belong to only one cell; its current cell defines a MH’s “location”, and the MH is considered *local* to the MSS providing wireless coverage in this cell.

The system model for supporting host mobility (Fig. 2.1) consists of two distinct sets of entities: a large number of mobile hosts and relatively fewer number of MSSs. All fixed hosts and the communication paths between them constitute the *static / fixed* network. The fixed network connects islands of wireless cells, each comprising of a MSS and the local MHs.

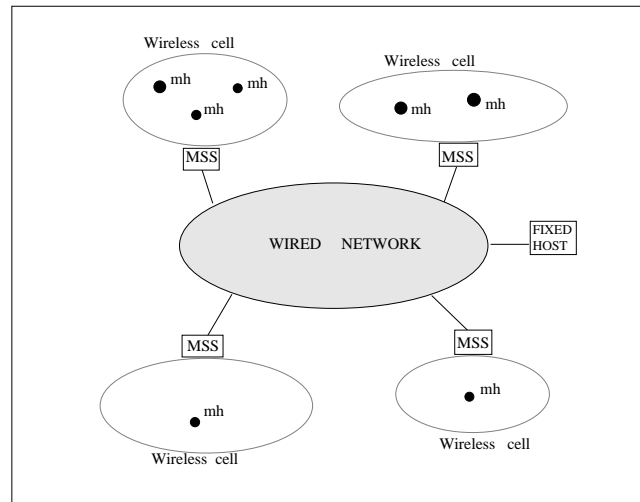


Figure 2.1: The system model for supporting host mobility

The static network provides reliable, sequenced delivery of messages between any two MSSs, with arbitrary message latency. Similarly, the wireless network within a cell ensures fifo delivery of messages between a MSS and a local MH, i.e. there exists a fifo channel from a MH to its local MSS, and another fifo channel from the MSS to the MH. If a MH did not leave its cell, then every message sent to it from the local MSS would be received in the sequence in which they are sent. However, a MH may leave its current cell at any time and the sequence of messages received at the MH is a prefix of the sequence of messages sent from its local MSS, i.e. if m_1, m_2, \dots, m_s be the sequence of messages sent from the local MSS, then the sequence of messages received at the MH prior to leaving the cell is m_1, m_2, \dots, m_r , where $s \geq r$.

The model requires that a MH send a *leave(r)* message on the MH-to-MSS channel supplying the sequence number of the last message, viz. r , received on the MSS-to-MH channel. Once this message is sent, the MH neither sends nor receives any further message within the current cell. Each MSS maintains a list of ids of MHs that are local to its cell; on receipt of *leave()* from a local MH, it is deleted from the list. When a MH enters a new cell, it sends a *join(mh-id)* to the new MSS; it is then added to the list of local MHs at the new MSS.

Host mobility is represented in this model as migration of MHs between cells. The specific mechanism for detecting that a MH has moved to a new cell is not relevant to our model; the model only assumes that a MH is able to detect such a change and send a final message (*leave()*) prior to leaving its cell¹. A possible mechanism is the *beacon protocol* of [40], wherein each MSS periodically broadcasts its identity (*beacon*) in the local cell. When two successive beacons received by a MH differ in the identity of their senders, it implies that the MH has switched cells.

The above abstraction of requiring a MH to send *leave()* and *join()* messages during a move, is useful for designing distributed algorithms. However, multicast-ing protocols described in chapters 5 and 6, are affected by how such an abstraction is implemented, and specification of one such implementation will be described in chapter 5.

Message communication from a MH h_1 to another MH h_2 occurs as follows. h_1 first sends the message to its local MSS M_1 using the wireless link. M_1 forwards it to M_2 , the local MSS of h_2 , via the fixed network. M_2 then transmits it to h_2 over its local wireless network. However, since the location of a MH changes with every move and its current location is not universally known in the network, M_1 needs to first determine where h_2 is located before it can forward the message from h_1 to M_2 . This is essentially the problem faced by network-layer routing protocols for mobile hosts, such as [18, 40, 61, 63]. Our system model is not tied to any particular routing scheme and instead, we assume that any message destined for a mobile host incurs a fixed *search cost*.

Mobile hosts often disconnect from the rest of the network. In our model, disconnection² is distinct from failure: disconnections are *elective* by nature and so, a mobile host can inform the system of an impending disconnection prior to its occurrence and execute a separate application-specific disconnection protocol if necessary. A MH disconnects by sending a *disconnect(*r*)* message to its local MSS

¹ Note that the *leave()* message needs to be *logically* sent before leaving the cell, i.e. it is possible for the MH to enter a new cell and then send the *leave()* message to its previous MSS via its current MSS. A possible scheme is explained in section 5.2.2.

² Disconnection can either be voluntary or involuntary [46]. We use the term “disconnection” to always imply a voluntary disconnection; we refer to an involuntary disconnection as a failure.

M, where r is the sequence number of the last message it received from M (similar to a *leave(r)* message). M deletes the MH from its list of local MHs; however it sets a “disconnected” flag for the particular MH-id. When some other MSS M' attempts to contact this MH (while it is disconnected from the network), M informs M' of the disconnected status of the MH. Later, the MH may reconnect at a MSS N by sending a *reconnect(mh-id, previous mss-id = M)* message. N informs M of the MH's reconnection, and as a result, M unsets the “disconnected” flag for the MH while N adds it to its list of local MHs.

In our system model, the number of MSSs will be denoted by N_{mss} and that of MHs by N_{mh} with N_{mh} being much greater than N_{mss} . For simplicity of presentation, we assume that all fixed hosts can act as MSSs and use the terms MSS and “fixed host” interchangeably.

All components of the system, viz. MSSs, MHs, wired and wireless links, are assumed to be reliable. Since, a MH communicates with the rest of the network via its local MSS, failure of the MSS implies that that the MH can no longer communicate with the rest of the system and vice versa, i.e. failure of a MSS renders a local MH incommunicado. This problem can be overcome by having overlapping cells — a MH can now select any one of the MSSs covering its location to communicate with the rest of the network. In the absence of overlapping cells, a MH must physically move to an adjacent cell when its local MSS fails. The distributed algorithms discussed in this dissertation require state to be maintained at a MSS on behalf of local MHs, and thereby introduces an additional point of failure (compared to maintaining state only at the MHs). Though this dissertation does not handle failure of MSSs, it is possible to protect against MSS failures by replicating this state at additional MSSs besides just the local MSS. Possible approaches are sketched in [4, 47].

2.1 Characterizing features of the model

With the introduction of mobile hosts, the computing entities in our system model are partitioned into two distinct classes, viz. mobile and static hosts:

1. Mobile hosts, especially palmtops and “personal digital assistants” (PDAs), have significantly lower processing speed, memory capacity and disk storage compared to their desktop counterparts [7, 30, 36, 52]. It is believed that irrespective of technological advances, the ability to be portable or mobile will cause mobile computers to lag static hosts in these respects. As an example, consider the hardware configurations of a high-end non-mobile workstation (DEC AXP) with a mobile computer (EO 880) [52]:

System	Processor	MIPS	Memory	Disk	Bandwidth	Network
EO 880	25 MHz Hobbit	13	12 MB	64 MB	14.4 Kb/s	Cellular
DEC AXP	150 MHz AXP(500)	150	1 GB	15 GB	600 Mb/s	ATM

Table 2.1: Comparison of hardware configurations of high-end mobile and non-mobile computers

2. A moving host cannot draw on a tethered source of power, and is forced to use a stand-alone power source such as battery cells. Given the limited lifetime of such a source, and that CPU operations, memory accesses, data transmission and reception consume power, mobile hosts are faced with tight constraints on power consumption [9, 17, 39]. For example, the Apple Newton MessagePad uses 4 AAA batteries with an expected lifetime of 6–8 hours.
3. Mobile hosts operate in two novel modes not usually associated with desktop computers, viz. “doze” mode and disconnection. To reduce power consumption, mobile hosts may in a “doze” mode wherein the computer shuts/slows down most of its system functions and only listens for incoming messages. On reception of a message, the mobile host resumes its normal mode of operation. While in doze mode, a mobile host performs no user computation. Both disconnection and doze mode are voluntarily initiated. However, in doze mode, a mobile host is reachable from the rest of the system and can be induced to resume its normal operating mode, while disconnection and

subsequent reconnection can be initiated only from the mobile host with the mobile host completely unreachable from the network in the intervening period.

In our system model, messages sent over wired links between a pair of fixed hosts are counted separately from those sent over the wireless link between a MH and its local MSS :

1. The wireless link between a mobile host and its local MSS has a significantly lower bandwidth than a link between static hosts [9, 52, 30, 36]. For example, NCR's WaveLAN provide a data rate of 2 Mbps using radio-frequency communication while Ethernet and fast Ethernet provide 10 Mbps and 100Mbps respectively. Other typical data rates for wireless networks are 19.2Kbps to 1 Mbps using infrared technology, 19.2Kbps using cellular links e.g. CDPD, or Specialized Mobile Radio (SMR) networks such as RAM and ARDIS.
2. Message transmission and reception of a message over the wireless link consumes power at a MH, which is a constrained resource. On the other hand, fixed hosts do not suffer from power constraints and therefore, power consumption is not an issue for messages exchanged over wired links.

Thus, the cost of a message sent on a wireless link ($C_{wireless}$) and the cost of a message sent on a fixed link (C_{fixed}) are two distinct measures of communication cost incurred by a distributed algorithm in our system model.

Besides $C_{wireless}$ and C_{fixed} , there is a third measure of communication complexity in our model, viz. C_{search} . The *search cost* is the number of messages exchanged within the fixed network to locate a MH and forward a message to its current local MSS, from a source MSS. A typical search strategy for a MSS, e.g. [40], would be to query all other MSSs within a search area to determine if a MH is local to its cell. The MSS currently local to the MH will respond, and the original MSS can then forward a data packet to this MSS. Using such a search strategy, where the search cost is linearly proportional to N_{mss} , the number of locations (MSSs), the search cost C_{search} is equal to $(N_{mss} + 1) \times C_{fixed}$.

Based on the above cost parameters, a message sent from a MH to another MH incurs a cost $2 C_{wireless} + C_{search}$, while a message sent from a MSS to a non-local MH incurs a cost $C_{search} + C_{wireless}$.

2.2 The two-tier principle

We cast distributed systems with mobile hosts into a two-tier structure: (1) a network of fixed hosts with more resources in terms of storage, computing and communication, and (2) mobile hosts, which may operate in a disconnected or doze mode, connected by a low-bandwidth wireless connection to this network. We propose a *two tier* principle for structuring distributed algorithms in this model:

To the extent possible, computation and communication costs of an algorithm is borne by the static portion of the network. The core objective of the algorithm is achieved through a distributed execution amongst the fixed hosts while performing only those operations at the mobile hosts that are necessary for the desired overall functionality.

Below, we present justifications for this choice:

- A message sent from a MSS to a non-local mobile host incurs a *search cost*. The same is also true for a message exchanged between two mobile hosts in different cells. To reduce the search component of the overall execution cost, it is desirable that communication between a fixed host and a mobile host occur locally within the same cell.
- The ability of a MH to operate while on the move, requires a stand-alone source of power viz., batteries. Given the limited life of batteries, power consumption is a serious practical consideration at a MH[9, 17, 39]. In addition to disk accesses and cpu operations, a MH has to expend its limited power resources to send and receive wireless messages; such a constraint is not faced by messages exchanged between fixed hosts over the wired network. Additionally, wireless channels have a significantly lower bandwidth than those within the fixed network. Thus, the number of wireless messages

exchanged in any algorithm execution should be minimal, possibly at the expense of a higher number of messages exchanged within the fixed network.

The two points mentioned above suggest that the data structures encapsulating the “state” of an algorithm execution, should mostly reside at the fixed hosts; thus, messages generated to update these data structures will be addressed to fixed hosts. Communication necessary to execute an algorithm may be split into three components: *global*, *local* and *search*. The global component consists of messages whose source and destination are both fixed hosts, e.g. to update appropriate data structures, and mostly represents the communication necessary for the progress of an algorithm execution. The local component refers to communication within a single wireless cell between a MH and its local MSS, and will often be used to initiate an algorithm execution from a MH or to communicate the final result of an execution from a MSS to a local MH. The search component consists of messages that the fixed hosts exchange to determine the current location of a MH so that a message addressed to this MH, may be forwarded to the appropriate MSS. Thus, our approach suggests that the global component dominate the overall communication.

- The two unique modes of operation of mobile hosts viz., disconnected and “doze-mode”, provide compelling arguments against executing an algorithm directly on MHs. When operating in a doze-mode, the MH shuts/slow down most of its system functions to reduce power consumption, and only listens for incoming messages. Like disconnection, this is a voluntary operation. However, the implications are different. In doze mode, a mobile host is *reachable* from the rest of the system and thus, can be induced by the system to resume its normal operating mode, if required. In contrast, disconnection and subsequent reconnection is initiated from the mobile host; it is cut off from the system in the intervening period.

- A distributed algorithm designed for the mobile computing environment,

should not require each MH to participate in every execution of the algorithm. Otherwise, it prevents those MHs from operating in a doze-mode that neither initiated the computation nor is the result of an execution significant to them and consequently, attempts at conserving power by operating in doze-mode are completely thwarted. Thus, by downloading most of the communication and computation requirements to the fixed segment of the network, the static hosts are responsible for the progress of an algorithm execution and a mobile host will not be required to intervene unless it is interested in the outcome of the execution.

- Algorithms that directly execute at the mobile hosts need to consider the possibility that one or more of the participants may disconnect while an execution of the algorithm is in progress. This has a two-fold effect: (1) the algorithm should be designed to handle a variable number of participants while an execution is in progress, and (2) a search overhead will be incurred if the remaining (mobile) participants need to be informed of a MH's disconnection (and again of its subsequent reconnection). Though distributed algorithms for static systems that are designed to be fault-tolerant, do handle changes in the number of participants, it is inefficient to tackle disconnections of mobile hosts using these algorithms. Disconnection is a voluntary operation and therefore, a MH may inform the system *prior* to an impending disconnection: thus, disconnection should not be associated with the same semantics as failure. The two-tier principle makes it easy to handle disconnections: since the fixed hosts are responsible for the progress of an algorithm execution, disconnection of one or more MHs does not alter the number of participants in the algorithm. Further, prior to disconnecting from the network, a MH can download any data to the fixed network that is necessary for progress of the algorithm.

- Many distributed algorithms rely on an underlying logical structure such as

a ring [50], tree [3] or grid[51], amongst the participants to carry out the needed communication. The main purpose of such a structure is to provide a certain degree of order and predictability to the communication amongst the participants; messages exchanged within such structures follow only selected logical paths. Consider now, the effects of the different operational modes of the mobile hosts on a logical structure comprising of MHs.

- Disconnection of a mobile node may require that the logical structure be reconfigured, resulting in additional message traffic and possible search overhead.
- Further, a logical structure predefines the sequence of nodes that a message should traverse starting from a given sender to its destination; thus, the intermediate nodes if operating in doze-mode, are forced to resume normal operation to forward such messages.

Thus, the cost of maintaining a logical structure amongst the mobile hosts may override the benefits of using such an underlying structure for algorithm design. Instead, it may be possible to obtain similar benefits by maintaining the logical structure amongst the fixed hosts without experiencing the disadvantages associated with that of mobile hosts.

2.3 Chapter summary

This chapter defined our system model for a network with mobile hosts. The communication efficiency of a distributed algorithm in this model is measured using three separate message counts: C_{fixed} , $C_{wireless}$ and C_{search} . We then proposed the *two tier* principle for structuring distributed algorithms in this model.

The next chapter discusses the impact of host mobility on the design of distributed algorithms. Using the problem of distributed mutual as a running example, we show the advantages of using the *two tier* principle and also shows how different location management strategies can be incorporated into the design of distributed algorithms for mobile hosts.

Chapter 3

Structuring Distributed Algorithms for Mobile Hosts

This chapter investigates how distributed algorithms should be structured for mobile hosts using the system model presented in chapter 2. To illustrate how the two-tier principle can be used to meet the constraints of mobile computing, we consider distributed mutual exclusion for mobile hosts and two classical algorithms [50, 49] for this problem. We first demonstrate that both algorithms are inefficient if executed directly at the mobile hosts since they were not designed for mobile hosts. The two-tier principle is then applied to restructure Lamport's algorithm [49] that reduces the search cost within the static network and satisfies the resource constraints of MHs viz., reduced power consumption and fewer messages over the low-bandwidth wireless links.

In conjunction with the two-tier principle, we also need strategies to track the location of "migrant" MHs, i.e. MHs that invoke a service from the fixed network at one cell, but move between several locations (cells) before receiving the result. Therefore, to deliver the desired result to a migrant MH, a distributed algorithm must now explicitly incorporate location management of migrant MHs in its design. The benefits of the two-tier principle and various location-management strategies for migrant MHs, viz. *search*, *inform* and *proxy*, are illustrated by structuring a token-based logical ring for mobile hosts. It is observed that mobility introduces the possibility of unfair accesses to the token (circulating in the logical ring), and we present a scheme to ensure at most one access to the token by a MH per traversal of the ring.

Lastly, we consider an alternative approach for mutually exclusive access to the token amongst the MHs, that does not require explicit location management. This is achieved by *replicating* token requests at all locations (MSSs).

3.1 Prevalent model of distributed computation

The model commonly used as a basis for distributed algorithms is an undirected graph, where the nodes represent processes executing on static hosts, and the edges represent logical communication links between nodes. A logical link between a pair of nodes is implicitly mapped to a path in the underlying physical network. In the absence of failures, this mapping does not change. Communication between processes occurs only through exchange of messages; processes do not share memory nor do they access a common clock. A node is allowed to send a point-to-point message to any other node as long as there is a logical link between the two nodes. The communication complexity of a distributed algorithm in this model is measured by the number of messages sent on these logical links. There are variations to this basic model such as synchronous or asynchronous message delivery, fifo message delivery, incorporating failures of hosts and/or links, assigning weights to edges, etc.

We now review two classic distributed algorithms in this model. Both are solutions to the problem of distributed mutual exclusion: a set of processes compete for access to a shared resource (“critical region”) such that no more than one process is allowed to simultaneously access that resource, and there be no privileged process to determine the order of access. Lamport’s algorithm [49] allows a host to send messages to any other host, i.e. there is a logical link between every pair of hosts, and is based on replicating a *request queue* at all hosts. In contrast, Le Lann’s algorithm [50] imposes a logical communication pattern amongst the hosts by arranging them in a unidirectional ring: a host can send a message only to its successor in the ring and receive messages only from its predecessor.

Lamport’s algorithm Each host maintains a logical clock that is used to assign timestamps to messages; rules for updating the clocks and timestamping messages are specified in [49], and are not crucial to this discussion. Instead, we will focus on the communication induced by the algorithm. Briefly, the algorithm is as follows :

- Each host maintains a *request queue*, initially empty, that contains *request* messages in increasing order of their timestamps. A host h_i requests a

mutually exclusive access to a resource, by sending a timestamped *request* message to all other hosts and inserts the message in its own queue.

- A recipient of the *request* message sends a timestamped *reply* message to the sender of the request, and inserts the request in its own queue.
- A host is allowed access to a resource when
 - its own *request* message is at the front of the queue, and
 - it has received a message from every other host with a timestamp greater than the timestamp of its own request.
- To release the resource, a host deletes its request from the queue, and sends a *release* message to all other hosts.
- A recipient of the *release* message deletes the sender's *request* message from its queue.

The underlying model for the above algorithm consists of N fixed hosts, with a *fifo* communication channel between every pair of hosts. The algorithm requires $3 \times (N - 1)$ messages for an execution; each round of *request*, *reply* and *release* messages are sent to $N - 1$ other hosts.

Le Lann's algorithm All hosts are logically arranged in an unidirectional ring and a *token* moves around this ring. At any instant, the process currently holding the token is allowed to enter the critical region. If a host holding the token does not need to enter the critical region it simply passes the token to the next host in the ring; otherwise, it completes its access to the critical region and then passes the token. A host thus executes the following algorithm :

- *wait receipt of token from its predecessor in the ring;*
- *enter <critical region>, if desired;*
- *send token to its successor in the ring.*

The algorithm trivially satisfies two important properties: (1) mutual exclusion is trivially guaranteed to the current holder of the token, and (2) it allows fair access to the token by allowing each participant to access the token at most once in one

traversal of the ring. With N hosts, N messages are needed for the token to traverse the ring once. Let the number of mutual exclusion requests that were satisfied during one traversal of the ring by the token, be K where $0 \leq K \leq N$. Thus, the cost of the algorithm is N messages per K accesses to the critical region.

All communication in this algorithm takes place only along those channels that define the logical ring, and is representative of distributed algorithms that use a logical communication structure amongst participants. Besides mutual exclusion, this algorithm has been used for termination detection[29], group membership and message-ordering protocols[8], and appears to be a fundamental algorithm in the field of distributed systems.

Summary of characteristic features As can be observed from the discussion so far, the traditional model of distributed computation can be characterised by the following three features:

1. Relative location of the endpoints of a logical link is invariant for the duration of an algorithm execution, i.e. once it has been established which sequence of physical links in the underlying communication network constitute the logical link, this mapping of logical to physical links does not change.
2. The efficiency of an algorithm is measured by the number of messages required for one execution of the algorithm.
3. Algorithms do not distinguish different classes of participants, and all participants are considered to be at par with each other in terms of processing power, memory and communication bandwidth.

3.2 Impact of mobility

We now investigate why it is inefficient to execute classical distributed algorithms at mobile hosts, that are neither aware of host mobility nor of the resource constraints associated with such hosts.

Algorithm L-MH We first look at the drawbacks of executing Lamport’s algorithm directly at N mobile hosts, which we refer to as algorithm L-MH. Without delving into the details, consider only the communication pattern and data structures required by the algorithm. To secure mutual exclusion, a participant sends a *request* message to all other participants, waits for a *reply* message from each of them, and then sends a *release* message to all others after completing its access to the critical region. Each participant is required to maintain a *request queue* of pending requests, which is updated on receipt of *request* and *release* messages. This approach has the following drawbacks:

- *High search cost.* Each message in the algorithm is addressed to a mobile host and therefore, incurs a *search cost*. The overall cost of one execution of the algorithm is $3 \times (N_{mh} - 1) \times (2 \times C_{wireless} + C_{search})$. Note that the search overhead is proportional to N , the number of MHs in the system.
- *Battery consumption at MHs.* Updates to the *request queue* and message transmission and reception over the wireless links, consumes power at the MHs. The source and destination of each message in this scheme is a MH, and therefore, consumes battery power at both the sender (to transmit it to the local MSS) and the destination (to receive the message from its local MSS). The overall energy consumption for one execution of the algorithm is thus proportional to $6 \times (N_{mh} - 1)$. The energy consumed at an initiator is proportional to $3 \times (N_{mh} - 1)$, while each of the other $(N_{mh} - 1)$ MHs consume energy to receive two messages (*request* and *release*) and send one (*reply*) message each.
- *Fifo channels between MHs.* Correctness of the algorithm requires that messages are delivered in sequence (fifo) at a destination. Since in algorithm L-MH, the source and destination of every message is a MH, this requirement places an additional burden on the underlying network protocols to maintain a logical fifo channel between each pair of MHs, regardless of their location in the network.
- *Doze and disconnected modes.* Algorithm L-MH requires the participation

of every MH in every execution of the algorithm. Consequently, it does not permit any MH to disconnect or to operate in a doze mode *without interruption* even for the time interval during which it does not attempt to access the critical region.

Algorithm R-MH Consider an execution of Le Lann's algorithm where each participant is a mobile host and the logical ring comprises of all the MHs in the system. We will refer to this algorithm as R-MH. Algorithm R-MH is the extreme case of executing an existing distributed algorithm at the mobile hosts. It shows that *correctness* of an existing algorithm, which did not explicitly consider host mobility, is not compromised when applied to the mobile environment. However, such an approach is not sensitive to the resource constraints specific to the mobile environment and consequently suffers from the following drawbacks:

- *High search cost* Each message in the algorithm is addressed to a mobile host and therefore, incurs a *search cost*. Since the algorithm circulates the *token* amongst all MHs, the overall search cost incurred by the algorithm is proportional to N_{mh} .
- *Excessive usage of wireless links* Both sender and destination of each message is a MH; the message is thus transmitted over the wireless links between the respective local MSSs of both the sender and destination MHs. Therefore, the wireless cost component of every message in this algorithm is $2 \times C_{wireless}$.

The cost of each message (to pass the token between two adjacent MHs in the ring) is thus $2 \times C_{wireless} + C_{search}$ and the overall cost of the algorithm is $N_{mh} \times (2 \times C_{wireless} + C_{search})$. Note that this cost is independent of the number of mutual exclusion requests satisfied in one traversal of the ring.

- *Power consumption at MHs* The wireless component of the overall cost is indicative of the power consumed at MHs for transmission and reception of messages. Each message in this algorithm consumes power at both the sender (to transmit it to the local MSS) and the destination (to receive the

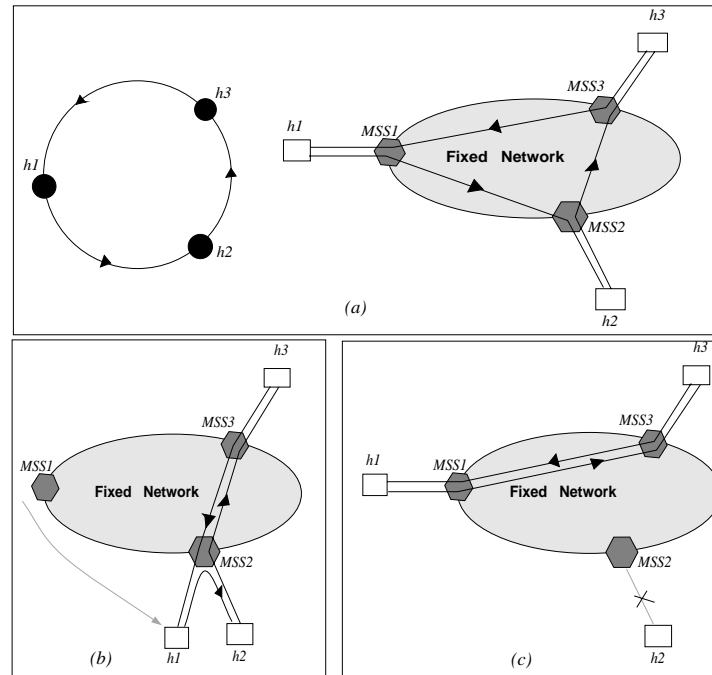


Figure 3.1: Effects of mobility on a unidirectional logical ring

message from its local MSS). Thus, an execution of the algorithm requires *every* MH to expend power for accessing the wireless link twice, and the cumulative power consumption for one execution of the algorithm is proportional to $2 \times N_{mh}$.

- *Doze and disconnected modes.* Algorithm R-MH requires the participation of *every* MH to maintain the logical ring and cannot therefore permit *any* MH to disconnect. To allow disconnections, the logical ring will need to be reconfigured amongst the remaining participants. Secondly, a MH operating in doze mode, is forced to resume normal operation on receipt of the *token* from its predecessor in the ring. It may revert to doze mode after forwarding the *token* to its successor. Thus, algorithm R-MH does not allow a MH to operate in doze mode *without interruption* even though it does not need to access the shared resource (represented by the token); it must still receive and forward the token to enable other MHs to access the resource.

As an illustration of the above drawbacks, consider a logical ring among three mobile hosts $h1$, $h2$ and $h3$ shown in Fig. 3.1a. In our system model, a single logical link in the ring, e.g. between $h1$ and $h2$, consists of: (1) the wireless

connection between $h1$ and its “current” local MSS, $MSS1$, (2) a logical point-to-point connection between $MSS1$ and $MSS2$, the local MSS of $h2$ and (3) a wireless connection between $h2$ and $MSS2$. Now, if $h1$ changes its location and moves to the local cell under $MSS2$, then the logical links of the ring between the three mobile hosts need to be re-mapped as shown in Fig. 3.1b. This is manifested as search cost when $h3$ forwards the token to $h1$. Next, assume $h2$ is presently operating in doze-mode; it is forced to resume its regular operating mode on receipt of the token from $h1$ and then forward the token to $h3$; $h2$ itself may not have any use for the token. In addition to doze-mode of operation, the logical ring is also affected by disconnections. Specifically, the disconnection protocol should ensure that the logical structure can be re-established amongst the remaining participants. Consider disconnection of a MH $h2$ from the ring as shown in Fig. 3.1c. Prior to physically detaching itself, it can inform its immediate predecessor in the ring ($h1$) of its intent to disconnect; $h1$ can then permit $h2$ to disconnect after establishing a logical link directly to $h3$. The logical ring now consists of the remaining participants, viz. $h1$ and $h3$. Thus, disconnection causes deletion and addition of logical links from the ring and thereby a reconfiguration of the physical links; the set of participants changes as well. In contrast, a change in location of a mobile host causes only a reconfiguration of the physical links comprising the logical ring (Fig. 3.1b): neither the set of participants nor the set of logical links change. The impact of host mobility on logical structures is apparent in our model.

It is important to emphasize here that the drawbacks of both algorithms, L-MH and R-MH are not intrinsic to the algorithms per se, but stem from an (inappropriate) application of the algorithms to an environment for which they were not designed. For example, in case of L-MH, the logical ring is established *amongst the mobile hosts*. Unlike fixed hosts, physical connectivity amongst the mobile hosts is redefined on every move and this is manifested through an increase in the search component of the algorithm; further, fixed hosts do not suffer from the constraints of power consumption and low-bandwidth wireless connectivity unlike mobile hosts. We remedy these drawbacks below by applying Lamport’s and Le Lann’s algorithm to the mobile environment in accordance with the two-tier principle.

3.3 Applying the two-tier principle to Lamport's algorithm

We adapt Lamport's algorithm to the mobile computing environment, by shifting the communication and computation requirements of the algorithm to the static segment. Instead of the N_{mh} MHs executing the algorithm, it is the MSSs that maintain the necessary data structures viz., the *request queue*, and exchange *request*, *reply* and *release* messages amongst each other within the static network to secure mutual exclusion on behalf of a requesting MH. The necessary modifications to Lamport's algorithm are as follows, resulting in algorithm L-MSS:

- Only messages exchanged between MSSs follow the timestamping rules of [49]; messages between a MH and a MSS are not timestamped.
- A MH $h1$ initiates algorithm L-MSS by sending a message, *init(h1)*, to its local MSS $m1$. $m1$ now executes Lamport's algorithm with other MSSs, on behalf of $h1$: the *request*, *reply* and *release* messages are tagged with the initiating MH's id $h1$.
- When $m1$ secures mutual exclusion for $h1$ (following the rules of Lamport's algorithm), it sends a *grant_request* message to $h1$. Since $h1$ may have changed its cell in the meantime, this requires a search cost to locate $h1$.
- Having completed its access to the critical region, $h1$ sends a *release_resource* message to $m1$ (relayed via $h1$'s current local MSS). $m1$ then deletes $h1$'s request from its queue and sends a *release(h1)* message to every other MSS (as required by Lamport's algorithm).
- If $h1$ disconnects prior to receiving the *grant_request* message from $m1$ then, on receiving the *grant_request* message from $m1$ (to be forwarded to $h1$), its current local MSS will note that the "disconnected" flag is set for $h1$ and in response, notify $m1$ of $h1$'s disconnected status. Since $h1$ is unreachable, its request will not be satisfied and $m1$ sends a *release* message to all other MSSs. If $h1$ disconnects after receiving the *grant_request* message but without sending *release_resource*, then L2 requires that it reconnect to send the *release_resource* message. Disconnection of $h1$ at any other time does not affect the progress of L-MSS.

Correctness sketch Lamport's algorithm ensures that if the timestamp of a request R1 is less than that of another request R2, then R1 will be satisfied before R2. In algorithm L2, a request from a MH is timestamped when the *init()* message is received by its local MSS, i.e. though MHs do not maintain logical clocks, the timestamp assigned to *request(h1)* by *m1* can be considered as the timestamp of *h1*'s request for mutual exclusion. Since the MSSs execute Lamport's algorithm without any modifications to the algorithm per se, a *grant_request* message will be sent to *h1* before another MH *h2* if the timestamp assigned to *request(h1)* is less than that of *request(h2)*.

Communication Costs First, the *init* message costs $C_{wireless}$. Delivery of *grant_request* requires a search for the current local MSS of *h1* followed by a wireless transmission, i.e. $C_{wireless} + C_{search}$. Delivery of *release_resource* incurs a cost $C_{wireless} + C_{fixed}$. Thus the overall cost in this algorithm is :

$$(3 C_{wireless} + C_{fixed} + C_{search}) + (3 \times (N_{mss} - 1) \times C_{fixed})$$

Comparison of algorithms L-MH and L-MSS It can be clearly observed from the cost structures of algorithms L-MH and L-MSS, that L-MSS eliminates the drawbacks of L-MH by *explicitly* acknowledging mobility of hosts and shifting the required data structures onto the static portion of the system: the participation of the mobile hosts was kept to a minimum.

1. L-MH incurs a search overhead proportional to N_{mh} , while L-MSS incurs only a *constant search cost* per execution. Also, since $C_{search} > C_{fixed}$ and $N_{mh} \gg N_{mss}$, the overall message cost is lower for L-MSS than L-MH.
2. L-MH requires only *constant* number of wireless messages, viz. $3 C_{wireless}$, and does not store the *request queues* at MHs; L-MSS is thus more energy efficient than L-MH in terms of consuming battery power at the MHs.
3. L-MSS does not require fifo communication channels with mobile endpoints.
4. L-MH does not provide for the disconnection of any MH. L-MSS is not affected by disconnection of a MH unless it has a pending request for mutual exclusion, in which case it tackles it efficiently.

3.4 Structuring a token-based logical ring for mobile hosts

The two-tier principle suggests that the logical ring should be established within the fixed network. The logical ring now consists of all MSSs with the token visiting each MSS in a predefined sequence. A MH that wishes to access the token is required to submit a request to its local MSS. When the token visits this MSS, all pending requests are serially serviced. However, a MH may have changed its location since submitting its request, and therefore, the location of such migrant MHs need to be explicitly managed. Below, we present two location management strategies, viz. *search* and *inform*, for the case when all MSSs constitute the logical ring. An alternative method of structuring the ring is to partition the set of all MSSs into “areas” and associate a designated fixed host, called a *proxy*, with each area. The token now circulates only amongst the proxies, and each proxy is responsible for servicing token requests from MSSs within its area. A combination of search and inform strategies is used to manage the location of migrant MHs in this case.

SEARCH Strategy Algorithm R-MSS:search maintains a unidirectional logical ring *amongst the MSSs*, and the token circulates in this ring. It consists of two distinct interactions: one, that is executed solely within the static segment to circulate the token amongst the MSSs, and the other takes place when a MH submits a request for the token to its local MSS.

Actions executed by a MSS M

- On receipt of a request for the token from a local MH, M adds the request to the rear of its *request queue*.
- When M receives the token from its predecessor in the logical ring, it executes the following steps:
 - a. Pending requests from M 's *request queue* are moved to the *grant queue*.
 - b. *Repeat*
 - Remove the request at the head of the *grant queue*
 - If the MH making the request is currently local to M , then deliver the token to the MH over the wireless link.

- Else, search and deliver the token to the MH in its current cell.
- Await return of the token from the MH.

Until *grant queue* is empty.

- c. Forward token to M's successor in the logical ring.

Actions executed by a MH h

- When *h* requires access to the token, it submits a request to its *current* local MSS.
- The MSS where *h* submitted its request will eventually send the token to *h*. After *h* accesses the critical region, it returns the token to the same MSS.

The above algorithm assumes that a MH does not submit a second request if its previous request has not yet been serviced. Secondly, when a MH receives the token, it must return it to the sender MSS after accessing the critical region, i.e. it may not disconnect *permanently* after receiving the token. Note that a MH could change its cell after submitting its request and prior to receiving the token. The above algorithm locates migrant MHs using a *search* strategy and hence the name R-MSS:search. .

Correctness sketch Mutual exclusion is trivially guaranteed by the algorithm since at most one mh may hold the token at any given time. Next, consider why starvation does not occur, i.e. every request submitted by a mh is eventually granted. It needs to be shown the token can not reside forever at a fixed host and thus, it eventually visits every fixed host in the ring. First, note that the maximum number of requests serviced by a MSS holding the token is bounded: only those requests that were made prior to arrival of the token (since the token's last visit) are serviced. When the token arrives at a MSS, contents of the *request queue* are transferred to the *grant queue*; new requests are then added to the request queue. Only those requests that belong to the *grant queue* are satisfied by the MSS. It follows that the *grant queue* contains at most N_{mh} requests, one per mh.

Communication cost

- Cost incurred by the token for one traversal of the logical ring : $N_{mss} \times C_{fixed}$
- Cost of submitting a request from a MH to its local MSS: $C_{wireless}$
- Cost of satisfying a token request: $C_{search} + C_{wireless}$

The above cost is incurred for a *migrant* MH since the MSS (holding the token) first needs to search for this MH and forward the token to the MSS currently local to the MH.

- Cost of returning the token from a MH to the sender MSS: $C_{wireless} + C_{fixed}$
The C_{fixed} component is incurred when the MH returns the token in a cell other than where it submitted its request.
- Thus, the (worst case) cost of submitting and satisfying a single request is $3 C_{wireless} + C_{fixed} + C_{search}$.

The total cost of satisfying K requests in one traversal of the ring is thus,

$$K \times (3 C_{wireless} + C_{fixed} + C_{search}) + N_{mss} \times C_{fixed}$$

It is reasonable to expect that $K \ll N_{mh}$, i.e. the number of MHs requesting access to the token in a *single* traversal of the ring is much less than the total number of MHs.

Benefits of using the two-tier principle The advantages of algorithm R-MSS:search over R-MH are obtained by modifying algorithm R-MH in accordance with the *two tier principle*, viz. the mobile hosts interact with the static segment via a simple request-response protocol while the brunt of the required computation and communication are met by the MSSs. The two-tier principle suggests that the logical ring be maintained amongst the static hosts instead of the MHs, and algorithm R-MSS:search is thereby successful in meeting the constraints of mobile computing, as listed below:

1. *Reduced power consumption.* The total power consumption of algorithm R-MH is proportional to $2 N_{mh}$, while that of R-MSS:search is $3K$, i.e. R-MH requires *every* MH to expend power to receive and forward the token while R-MSS:search expends power at only those MHs that request the token.

Given the tight restrictions on power consumption at MHs, R-MSS:search is clearly preferable to R-MH.

2. *Fewer wireless messages.* Algorithm R-MH sent $2N_{mh}$ messages over wireless links while R-MSS:search sent $3K$ messages. Since bandwidth of wireless links between a MSS and local MHs is a critical resource, the two tier principle is instrumental in reducing its usage.
3. *Ease of handling disconnections* Algorithm R-MH is vulnerable to disconnection of *any* MH; disconnections can be handled only by executing yet another algorithm to reconfigure the ring amongst the remaining MHs. In comparison, disconnection of a MH that has not submitted a request, does not affect execution of R-MSS:search in any way. Disconnection of a MH with a pending request can be easily handled since a “disconnected” flag is set for the particular MH at its local MSS. For example, when the token is received by the local MSS (possibly, as a result of a search executed by another MSS), the token is simply be returned back to the sender MSS. This in effect implements a policy of nullifying a pending request if the MH disconnects; other policies better suited to specific applications can also be designed.
4. *Doze mode* R-MH also interrupts a MH operating in doze mode if it happens to be the next recipient of the token in the logical ring, irrespective of whether it needs access to the token. In contrast, R-MSS:search interrupts a MH in doze mode only to satisfy a prior request from that MH.
5. *Search cost* The total search overhead incurred by algorithm R-MH is *proportional to N_{mh}* , the number of MHs constituting the ring and is independent of K , the number of mutual exclusion requests satisfied in one traversal of the ring. Algorithm R-MSS:search incurs a search overhead *proportional to K* .

INFORM Strategy An alternative to the search strategy to locate a migrant MH is to require the MH to notify the MSS (where it submitted its request) after every change in its location till it receives the token.

Actions executed by a MSS M

- On receipt of a request from a local MH h , M adds a request $\langle h, M \rangle$ to the rear of its *request queue*.
- Upon receipt of a *inform*(h, M') message, the current value of *locn*(h) is replaced with M' in the entry $\langle h, locn(h) \rangle$ in M 's *request queue*.
- On receipt of the token, M executes the following steps:
 1. Entries from the *request queue* are moved to the *grant queue*.
 2. *Repeat*
 - Remove the request $\langle h, locn(h) \rangle$ at the head of the *grant queue*
 - If *locn*(h) == M , then deliver the token to h over the local wireless link
 - Else, forward the token to *locn*(h), i.e. the MSS currently local to h , which will transmit it to h over the local wireless cell.
 - Await return of the token from h .

Until grant queue is empty
 3. Forward token to M 's successor in the logical ring.

Actions executed by a MH h

- When h needs access to the token,
 - it submits a request to its current local MSS, say M , and
 - stores M in the local variable *req_locn*.
- When h receives the token from the MSS *req_locn*, it accesses the critical region, returns the token to the same MSS and then sets *req_locn* to \perp .
- After every move, h now includes *req_locn* with the *join*() message, i.e. it sends *join*(h, req_locn) message to the MSS M' upon entering the cell under M' .
 - If *req_locn* received with the *join*() message is not \perp , then M' sends a *inform*(h, M') message to the MSS *req_locn*.

Comparison of search and inform strategies To compare the search and inform strategies, let a MH h submit a request at MSS M and receive the token at M' . Assume that it makes MOB number of moves in the intervening period. Then, after each of these moves, a *inform()* message was sent to M , i.e. the *inform cost* is $MOB \times C_{fixed}$. In algorithm R-MSS:search, on the other hand, M would *search* for the current location of h and the cost incurred would be C_{search} . Thus, the inform strategy is preferable to search strategy when $MOB \times C_{fixed} < C_{search}$ i.e., if h changes cells “less often” after submitting its request, then it is better for h to inform M of every change in its location rather than M searching for h .

PROXY Strategy The efficiency of search and inform strategies is determined by the number of moves made by a migrant MH. While a search strategy is useful for migrant MHs that “frequently” change their cells, the inform strategy is better for migrant MHs that change their locations less often. We now present a third strategy that combines advantages of both search and inform strategies, and is tuned for a mobility pattern wherein a migrant MH moves frequently between “adjacent” cells while rarely moving between non-adjacent cells.

The set of all MSSs is partitioned into “areas”, and MSSs within the same area is associated with a common *proxy*. The proxy is a static host, but not necessarily a MSS. The token circulates in a logical ring, which now comprises of only the proxies. On receiving the token, each proxy is responsible for servicing pending requests from its *request queue*. Each request in the queue is an ordered pair $\langle h, proxy(h) \rangle$, where h is the MH submitting the request and $proxy(h)$ represents the area, i.e. proxy, where h is currently located. A MH makes a “wide area” move, when its local MSS before and after the move, are in different areas, i.e. the proxies associated its new cell is different from the cell prior to the move. Analogously, a “local area” move occurs when its proxy does not change after a move. This assumes that a MH is aware of the identity of the proxy associated with its current cell; this could be implemented by having each MSS include the identity of its associated proxy in the periodic *beacon* message.

Actions executed by a proxy P

- On receipt of a token request from a MH h (forwarded by a MSS within P 's local area), the request $\langle h, P \rangle$ is appended to the rear of the *request queue*.
- When P receives a *inform*(h, P') message, the current value of *proxy*(h) in the entry $\langle h, \text{proxy}(h) \rangle$ is changed to P' .
- When P receives the token from its predecessor in the logical ring, it executes the following steps:
 1. Entries from the *request queue* are moved to the *grant queue*.
 2. *Repeat*
 - Delete the request $\langle h, \text{proxy}(h) \rangle$ from the head of *request queue*.
 - If $\text{proxy}(h) == P$, i.e. h located within P 's area, then deliver the token to h after searching for h within the MSSs under P .
 - Else, forward the token to *proxy*(h) (different from P) which will deliver the token to h after a *local* search for h within *its* area.
 - Await return of the token from h .

Until grant queue is empty
 3. Forward token to P 's successor in the ring.

Actions executed by MH h

- When h requires access to the token, it submits a request to its local MSS and stores the identity of the local proxy in *init_proxy*.
- When h eventually receives the token from *init_proxy*, it accesses the critical region, returns the token to *init_proxy* and then sets *init_proxy* to \perp .
- After a move, h sends a *join*($h, \text{init_proxy}$) message to the new MSS.
 - If *init_proxy* is not \perp , and *init_proxy* is different from the proxy P' serving the new MSS, i.e. h has made a wide-area move, then the new MSS sends a *inform*(h, P') message to *init_proxy*.

Communication cost Let the number of proxies constituting the ring be N_{proxy} , and the number of MSSs be N_{mss} ; the number of MSSs within each area is thus

N_{mss} / N_{proxy} . Let MOB_{wide} be the number of wide-area moves made by a MH in the period between submitting a token request and receiving the token; MOB_{local} represents the total number of local-area moves in the same period, and MOB is the sum of local and wide area moves.

Prior to delivering the token to a MH, a proxy needs to locate a MH amongst the MSSs *within its area*. We refer to this as a *local search*, with an associated cost $C_{l-search}$ and formulate the communication costs as follows:

- Cost of one token circulation in the ring: $N_{proxy} \times C_{fixed}$
- Cost of
 - submitting a token request from a MH to its proxy: $C_{wireless} + C_{fixed}$
 - delivering the token to the MH: $C_{fixed} + C_{l-search} + C_{wireless}$
(the C_{fixed} term can be dropped if the MH receives the token in the same area where it submitted its request).
 - returning the token from the MH to the proxy: $C_{wireless} + C_{fixed}$

The above costs add up to: $3 C_{wireless} + 3 C_{fixed} + C_{l-search}$

- Informing a proxy after a wide-area move: C_{fixed}
- The overall cost (worst case) of satisfying a request from a MH, including the inform cost, is then $(3 C_{wireless} + 3 C_{fixed} + C_{l-search}) + (MOB_{wide} \times C_{fixed})$

Comparison with search and inform strategies It is obvious that the cost of circulating the token amongst the proxies is less than circulating it amongst all MSSs (as is the case for search and inform strategies) by a factor N_{proxy} / N_{mss} . The cost of circulating the token within the logical ring is a measure of *distribution* of the overall computational workload amongst static hosts. If all three schemes service the same number of token requests in one traversal of the ring, then this workload is shared by N_{mss} in search and inform schemes, while it is spread amongst N_{proxy} fixed hosts under the proxy method. To compare the efficiency of each algorithm to handle *mobility*, we need to consider the communication cost of satisfying token requests.

The three algorithms incur the following costs to satisfy a MH's request for the token:

$$3 C_{wireless} + C_{fixed} + C_{search} \dots \text{(search)}$$

$$3 C_{wireless} + C_{fixed} + (MOB \times C_{fixed}) \dots \text{(inform)}$$

$$3 C_{wireless} + (3 + MOB_{wide}) \times C_{fixed} + C_{l-search} \dots \text{(proxy)}$$

Based on the above cost measures, it can be seen that the proxy scheme performs better than search when:

$$(3 + MOB_{wide}) \times C_{fixed} + C_{l-search} < C_{fixed} + C_{search}$$

$$\text{i.e., } MOB_{wide} + 2 < (C_{search} - C_{l-search}) / C_{fixed} \dots (I)$$

A typical search strategy, e.g. [40], for a MSS would be to query all other MSSs within a search area to determine if a MH is local to its cell. The MSS currently local to the MH will respond, and the original MSS can then forward a data packet to this MSS. If N_{area} is the number of MSSs within the search area, the search cost is equal to $(N_{area} + 1) \times C_{fixed}$. Using such a search strategy, where the search cost is linearly proportional to the number of locations within the search area, formula (I) above reduces to:

$$MOB_{wide} < N_{mss} - (N_{mss} / N_{proxy}) - 2$$

i.e., the proxy scheme performs better than search when the number of wide-area moves made by a migrant MH is two less than the total number of MSSs that is *outside* any given area.

Now compare the proxy and inform schemes. The proxy scheme incurs a lower cost to satisfy a token request when:

$$(3 + MOB_{wide}) \times C_{fixed} + C_{l-search} < (MOB + 1) \times C_{fixed}$$

$$\equiv C_{l-search} < (MOB - MOB_{wide} - 2) \times C_{fixed}$$

$$\equiv C_{l-search} < (MOB_{local} - 2) \times C_{fixed} \dots (II)$$

Using a search strategy based on [40], the cost of a local search equals $(N_{mss}/N_{proxy} + 1) \times C_{fixed}$, and formula (II) above reduces to:

$$(N_{mss} / N_{proxy}) + 2 < MOB_{local}$$

i.e., when the number of local-area moves made by a migrant MH is two more than the number of MSS under each proxy, the proxy scheme outperforms the inform scheme.

Ensuring fair access to the token regardless of mobility In the three search strategies discussed so far, there are two entities whose location varies with time: (a) the token, and (b) MHs. This allows for a situation where a MH accesses the token at its current cell, moves to a cell under a MSS that is the next recipient of the token in the logical ring, and accesses the token again. Thus, a MH by virtue of its greater mobility (compared to a stationary MH) could access the token multiple times in one traversal of the ring by the token, while a stationary MH can access the token at most once. If “fairness” of access amongst all MHs, independent of mobility, is a desirable goal, then additional synchronization mechanisms need to be built into the algorithm to achieve this goal.

Note also that algorithm R-MH, which maintains the logical ring amongst MHs, allows fair access to the token. Therefore, when we establish the logical ring within the fixed network, we need to also preserve the functionality of algorithm R-MH by providing a scheme that will ensure fair access to the token regardless of a MH’s mobility.

As an example, consider two MHs $h1$ and $h2$ initially located in the same cell under MSS M1. Let MSS M2 be the successor of M1 in the logical ring. If both MHs contend for the token at M1, then both requests will be satisfied when the token reaches M1. Now, assume $h2$ moves to the cell under M2 and contends for the token. If its request is received at M2 prior to the token reaching M2 from M1, then $h2$ can access the token for a second time. In contrast, if $h1$ continues to remain in M1’s cell, it can again access the token only after the token revisits M1 after completing one traversal of the ring.

The scenario outlined above does not cause starvation, but it increases the delay for a “stationary” MH to access the token. In the worst case, a token request from a MH such as $h1$ may be satisfied only after every other MH has accessed the token

1. The token is associated with a loop counter (*token_val*) which is incremented every time it completes one traversal.
2. Each MH maintains a local counter *access_count* whose current value is sent along with the MH's request for the token to the local MSS.
3. A pending request is moved from the *request queue* to the *grant queue* at the MSS (or proxy) holding the token, only if the request's *access_count* is less than the token's current *token_val*.
4. When a MH receives the token, it assigns the current value of *token_val* to its copy of *access_count*.

once from each of the N_{mss} cells, i.e. after a total of $(N_{mh} - 1) \times N_{mss}$ requests have been satisfied.

Below, we present a scheme that is applicable to all three location management strategies, which prevents a MH from accessing the token more than once in one traversal of the ring, i.e. how often a MH can contend for the token is not affected by its mobility (or lack of it).

Note that in step (4) above, a MH's *access_count* is reset to the token's current *token_val*; thus, even if a MH has a low *access_count*, it can access the token only once in a given traversal. With this scheme, the number of token accesses K satisfied in one traversal of the ring is limited to N_{mh} (when the ring comprises of all MSSs), while K could be $N_{mss} \times N_{mh}$ in the absence of this scheme. In effect, this scheme represents a trade-off between "fairness" of token access amongst the MHs and satisfying as many token requests as possible per traversal.

Other alternative criteria for "fairness" are also possible. For example, a MH may be allowed to access the token multiple times in one traversal of the ring, subject to the limitation that its total number of accesses to the token not exceed the current *token_val*. This criterion can be implemented by modifying step (4) above, as follows: the *access_count* of a MH is incremented on every access, instead of being assigned the current value of *token_val*.

3.5 Data replication: a mechanism for location management

In the algorithms that we developed so far, explicit location management strategies were incorporated into the base algorithm, viz. a token circulating within a logical ring. This was needed to handle migrant MHs. We now consider a totally different algorithm for distributed mutual exclusion that is based on *replicating* requests at all MSSs.

The advantage of using a replication-based approach is that location of migrant MHs do not need to be explicitly managed, since regardless of its current location, the local MSS has a copy of a migrant MH's pending request. However, this also introduces a problem that if a MSS is presently servicing a MH's request, no other MSS should attempt to service another request; otherwise, mutual exclusion will be violated. This requires that requests from MHs be *globally* ordered amongst all MSSs, and only the first request in this ordered sequence be serviced at any time.

The problem of creating a total order of delivery of messages also arises in static systems, viz. all common recipients of two messages m and n should either receive m before n or vice-versa. However, we intend to use the ordering of messages for a different purpose, viz. when all requests are globally ordered, then only the request at the head of this order should be serviced by exactly one MSS. Choice of this MSS is determined by the current location of the MH that made this request: its local MSS can unilaterally decide to service the request with the assurance that no other MSS will simultaneously process any another request.

Besides the need for explicit location management of migrant MHs, the algorithms for distributed mutual exclusion using a logical ring that we discussed earlier, also suffer from a high latency in satisfying a MH's request for mutual exclusion. In the worst case, the token may visit all fixed hosts in the logical ring before a particular request is serviced. Further, MHs that frequently switch cells may access the token more than once in one traversal of the ring. In this context, algorithm R-MSS' is useful since it ensures that at most one request per MH is

ensured per traversal. We now propose an alternative scheme for ensuring mutual exclusion that is based on replicating requests amongst all MSSs.

Below, we present our algorithm for distributed mutual exclusion for mobile hosts that is derived from a message-ordering protocol [19] for static systems.

Actions taken by a MH

- Each MH h maintains a local counter, h_count , which is incremented prior to submitting a new request for mutual exclusion. A request for mutual exclusion $req(h, h_count)$ is submitted to its local MSS.
- On a move, h includes its current value of h_count with the $join()$ message to its local MSS.
- When a MH receives a $grant$ message from its local MSS, it accesses the shared resource and then replies with a $release(h, h_count)$ message to its current local MSS.

Actions taken by a MSS

- Each MSS maintains a *delivery queue* of pending requests; each request is flagged as either *deliverable* or *undeliverable*, and is assigned a priority number. The queue is kept sorted in increasing order of message priority numbers.
- When a MSS M receives $req(h, h_count)$ from a MH h , it executes a two-phase protocol to order this request relative to all other pending requests.
 - In the first phase, M sends a copy of $req(h, h_count)$ to all other MSSs.
 - Each MSS assigns a (temporary) priority number to the received request from M , that is greater than any of the requests currently in its queue; the request is tagged *undeliverable* and inserted at the end of the queue. The temporary priority number is sent back to the initiator.
 - In the second phase, M computes the maximum of all temporary priority numbers, and sends it to all MSSs. Each MSS then assigns this number as the final priority number of the request and tags it *deliverable* and re-sorts the delivery queue.

- A MSS can service a pending request $req(h, count)$ if :
 - the request is tagged *deliverable*,
 - the MH h is local to its cell, and
 - the h_count value submitted by h (either with the *join()* message after entry to its cell, or with the *req()* message to this MSS) is equal to $count$, i.e. the request has not already been serviced by another MSS.

If these conditions are satisfied, then the MSS sends a *grant()* message to h .

- On receiving a *release(h, h_count)* message from a local MH, a MSS first deletes the entry $req(h, h_count)$ from its *delivery queue*. It then sends a *delete(h, h_count)* message to all other MSSs, which then delete the corresponding entry from their respective *delivery queues*.

Correctness sketch Correctness of the message-ordering portion of the above algorithm follows directly from the correctness of the two-phase ABCAST protocol of [19], and is not discussed. In the ABCAST protocol, a message is removed from the *delivery queue* of any participant when it reaches the head of the queue and is tagged *deliverable*. However, in our case, a message is a request for mutual exclusion on behalf of a MH, and is “delivered” to the MSS that is currently local to this MH. Hence, we need an explicit *delete* message from this MSS to all other MSSs. Also, we assign a counter value with each request to avoid a MH receiving *grant* messages from more than one MSS for the same request, as shown below:

1. A MH h , currently local to MSS M_1 , receives the *grant* message from M_1 (since the MH’s request is at the head of M_1 ’s queue) and replies with the *release* message to M_1 . M_1 sends a *delete* message to all other MSSs.
2. Before the *delete* message reaches a MSS M_2 , h moves to M_2 ’s cell. The request from h is still in M_2 ’s queue; if the request is at the head of the queue, then M_2 will send a *grant* message to h . Thus h may receive more than one *grant* message for the same request.

Communication costs It is easy to see that each execution of the above algorithm incurs a total cost of $(4N_{mss} - 1) \times C_{fixed} + (3 C_{wireless})$. The interesting

aspect of this cost is that it involves no search component, i.e. the cost is independent of a MH's mobility. The reason is that a request from a MH is replicated at all possible locations, and therefore, a MSS is always available to service a MH's request locally. However, even though the algorithm requires no explicit search, a higher C_{fixed} cost is incurred to replicate and order token requests.

3.6 Chapter summary

This chapter demonstrated that distributed algorithms for mobile hosts need to explicitly cope with the resource constraints of this environment, and also track the location of migrant MHs. We used the *two tier* principle to restructure (a) Lamport's algorithm for distributed mutual exclusion and (b) a token-based logical ring to address the resource constraints of mobile hosts and reduce the search cost (as compared to executing these algorithms directly at the mobile hosts). Besides the default strategy of searching for the current location of a migrant MH, we also proposed the *inform* and *proxy* strategies for tracking migrant MHs. Lastly, it was shown that data replication is a useful mechanism for coping with the effects of mobility.

The next chapter considers a specific distributed algorithm, viz. recording a consistent global checkpoint of a distributed application running on mobile hosts.

Chapter 4

Checkpointing Distributed Applications on Mobile Computers

This chapter considers the problem of recording a consistent global checkpoint of a distributed application executing on mobile hosts and presents a checkpointing algorithm that satisfies the constraints unique to the mobile computing environment.

Algorithms to checkpoint the global state of distributed application executing on *mobile* computers, are confronted with new constraints:

1. The location of a MH within the network, as represented by its current local MSS, changes with time. Checkpointing schemes that send control messages to MHs for example, to checkpoint the local state of a MH, will need to first locate the MH within the network (“search”), and thereby incur a search overhead.
2. Due to vulnerability of mobile computers to catastrophic failures [7], e.g. loss, theft, or physical damage, disk storage on a MH is not acceptably stable for storing message logs or local checkpoints. Checkpointing schemes must therefore, rely on an alternative (stable) repository for a MH’s local checkpoints.
3. Disconnection of one or more MHs should not prevent recording the global state of an application executing on MHs. It should be noted that disconnection of a MH is a voluntary operation, and frequent disconnections of MHs is an expected feature of the mobile computing environment.

Existing checkpointing algorithms for distributed applications have implicitly been designed for static hosts, and are therefore, inadequate to address the above constraints unique to the mobile computing environment.

Given that many of the mobile computers will be personalized portable computing devices, such as PDAs and personal communicators, they will be used for

multi-person interaction that require group communication between mobile computers, e.g. [31]. A user must have the assurance that a lost or misplaced device will not jeopardize the entire interaction. Checkpointing schemes are thus necessary to ensure that in spite of loss/failure of one or more such devices, the group interaction can at least be restarted from the last global checkpoint saved. The focus of our work is however restricted to *recording* checkpoints, and strategies for *recovery* are not discussed.

4.1 Consistent global state

In this paper, we model the global state of a distributed application running on mobile hosts along the lines of [23], as follows. The number of MHs on which the application executes is N , and the number of MSSs to which any of these MHs could move, is M . It will be assumed that underlying message routing protocols ensure that messages sent from MH h_i to any other MH h_j are received in sequence, i.e. there exists a *fifo* channel C_{ij} from h_i to h_j . A global checkpoint G_Ckpt of the distributed application running on the N MHs consists of a local checkpoint $local_ckpt_i$ for each MH h_i such that :

1. $local_ckpt_i$ includes a recorded *local state* of MH h_i .

The local state of a MH changes due to receipt of a message, sending a message or a local event that involves no communication. For a MH h_i with an initial state s_i^0 , the state after a sequence of events E_i , is represented as $\langle s_i^0 E_i \rangle$; if h_i is checkpointed at this local state, then all events $e \in E_i$ are said to be *included* in the local checkpoint and thereby, in G_Ckpt .

2. $local_ckpt_i$ also includes a copy of every message m sent from h_i to any other MH h_j if the event $send(m)$ is included in G_Ckpt while $recv(m)$ is not.

A global checkpoint G_Ckpt is “consistent” if it satisfies the following property: for any message m , if $recv(m)$ is included in G_Ckpt then $send(m)$ is also included.

4.2 Motivation

4.2.1 Inadequacy of existing checkpointing algorithms

Checkpointing mechanisms for static hosts has been widely studied and existing schemes can be broadly classified into two categories: synchronous and asynchronous. In synchronous checkpointing schemes such as [23], all participants coordinate their local checkpoints to ensure that this set of local checkpoints is guaranteed to be consistent. In the asynchronous approach, e.g. [59, 43, 44], each participant can choose to checkpoint its local state independent of others; however, an arbitrary set of local checkpoints, one for each participant, does not necessarily form a “consistent” global checkpoint. Asynchronous checkpointing schemes thus need to *a posteriori* select one local checkpoint per participant that will form a consistent global checkpoint.

Current synchronous and asynchronous checkpointing mechanisms are inadequate for the mobile computing environment. Synchronous checkpointing requires that all MHs (running a distributed application) coordinate with each other to checkpoint their local states; this ensures that the set of local checkpoints form a “consistent” global checkpoint of the application. As a result, control messages need to be sent to MHs to checkpoint local states. This entails a *search cost* to determine a MH’s current location in the network, i.e. its local MSS, so that control messages may be routed to the appropriate MSS. This problem is further compounded by the fact that a MH may switch cells before the checkpointing algorithm completes: if the algorithm requires the MH to be contacted multiple times during the course of the algorithm (e.g. in [23], a participant receives a *marker* message for every incoming channel), then the search cost may be multiplied that many times.

An alternative to searching for a MH’s current location is that a MH continually *informs* all interested MHs of every change in its location. So, when a MH needs to be contacted during an execution of the checkpointing algorithm, its current location is available to other interested MHs. However, if the MH changes its cell often and the number of MHs interested in its location is large, then this alternative will incur

a high *inform cost*. Thus, synchronous checkpointing algorithms will suffer a high *search / inform cost* overhead in the mobile environment.

Synchronous schemes are also adversely affected by disconnection of a MH. Note that disconnection does *not* imply that a MH has failed and therefore, a recovery procedure should not be initiated; disconnection is a voluntary operation. At the same time, since a disconnected MH is unreachable from the rest of the network, *its local checkpoint is inaccessible to a synchronous checkpointing algorithm*. Thus, either the algorithm will have to suspend its execution till the disconnected MH reconnects to the network or the primary goal of such an algorithm has to be sacrificed, viz. *coordinated* recording of local checkpoints of *all* MHs to ensure a consistent global checkpoint. Given the voluntary nature of a disconnection, it is also possible for a mobile host to record and deposit its local checkpoint at a static host prior to its disconnection. This ensures that although a mobile host may itself not be reachable, its local checkpoint is accessible to a checkpointing algorithm.

The utility of existing asynchronous checkpointing algorithms in the mobile environment is questionable due to “instability of stable storage” [7] on MHs. It is pointed out in [7] that in practice, while disk storage is treated as acceptably stable for log records for static hosts, this does not hold true for a mobile computer; a user may physically drop a MH especially if it is a laptop or palmtop, or the data stored on a MH’s disk may be totally wiped out by an airport security system or simply the MH may be stolen or lost. A mobile computer is thus uniquely vulnerable to a total catastrophic failure. Consequently, asynchronous checkpointing algorithms cannot assume that the saved local state and message logs of a MH will be available to it when required, and therefore cannot be directly applied to this environment. Note that disconnection also causes the same effects as a failure with respect to accessibility of a MH’s stable storage; however, in contrast to a failure, the stored local checkpoints and message logs become available after the MH reconnects.

Another issue to be considered is that the global checkpoint needs to be taken on a per-application basis, i.e. it is not necessary to checkpoint all MHs on a system-wide basis, but only those that participate in the (distributed) application. The

number of such MHs can be expected to be much smaller than the number of MSSs in the system (which represent the set of all potential locations between which MHs may move). Thus, schemes which require a MH (or a MSS on its behalf) to contact all other MSSs either 1) to request other mobile participants to checkpoint their state, or 2) to determine the locations of local checkpoints that the other participants transferred to the fixed network, will also incur an (implicit) search overhead which we would like to avoid.

4.2.2 Our approach to checkpointing MHs

For a checkpointing scheme to be practically realizable, it is necessary that the local checkpoints be stored on stable storage. However, it was pointed out in the previous section that disk storage on a MH cannot be considered to be stable; additionally, unlike a static host, every mobile computer is not necessarily equipped with disk storage. So where should the local checkpoints of the mobile hosts be stored? The key idea is :

Utilize the stable storage at a MSS to store local checkpoints of MHs.

A MSS, by virtue of being a static host, has access to stable storage (disk) and is also a more powerful machine in terms of computing and communication capacity. This is essentially a corollary of the *two tier* principle used in previous chapters. Similar to the multicasting schemes of chapters 5 and 6 where a MSS buffers messages and stores relevant data structures on behalf of a local MH, a checkpointing scheme for mobile hosts can use the disk space of a MSS as stable repository for the checkpoints of local MHs. A checkpoint is always stored at the MSS that is “currently” local to the MH: therefore, due to changes in a MH’s location in the network, *successive local checkpoints of the same MH may be found at different locations* (MSSs) within the static portion of the network. This issue does not arise in checkpointing a system with only static hosts: the local checkpoints of a given host are always available at the same “location” namely, the host being the checkpointed.

Next, consider how we handle disconnection of MHs. A disconnected MH is unreachable from the rest of the network; it can neither send nor receive messages. If not handled properly, disconnection of a MH may suspend execution of an ongoing

checkpointing algorithm till the MH reconnects. However, the *voluntary* nature of a disconnection suggests that it should be possible for a MH to checkpoint its local state and transfer it to the fixed network, e.g. its local MSS, as part of the disconnection protocol¹; a checkpointing algorithm can then use this checkpoint as a substitute for the MH's most recent local checkpoint in constructing the global checkpoint. Note that host mobility coupled with disconnections could also result in local checkpoints of the *same* MH to be stored at *different* locations (MSSs) in the fixed network, e.g., if successive disconnections of a MH occur in two different cells, then MSSs of both cells will store a local checkpoint on behalf of the MH.

Our checkpointing algorithm does not require explicit control messages to be sent to MHs (thereby, avoiding the search overhead of synchronous schemes) and local checkpoints of MHs are not recorded in a coordinated fashion. Instead, each application message is piggybacked with additional control information, based on which a recipient MH can asynchronously checkpoint its local state. Disconnections are handled by requiring a MH to checkpoint its local state prior to disconnection. The lack of stable storage at the MHs is circumvented by transferring the checkpointed state of a MH to the (stable) disk storage of its local MSS. Additionally, the algorithm also efficiently tackles the following two issues:

1. *Search cost:* In order to construct a global checkpoint, a local checkpoint needs to be included for every MH. However, successive checkpoints of a MH are scattered at different MSSs within the static network and therefore, one of the goals of a checkpointing scheme for MHs should be to reduce the search necessary to locate the required checkpoint(s).
2. Since each MH checkpoints its local state independently, it must be ensured that a *consistent* global checkpoint can be constructed *a posteriori* by selecting an appropriate set of local checkpoints, one per MH. An arbitrary collection of local checkpoints is not guaranteed to be consistent as illustrated in Fig. 2 below. At the same time, a MH should not be required to checkpoint its local state after *every* change in its local state since in addition

¹ An opposite action is taken by the Coda file system [46], where data is transferred *to* the disconnecting MH from the static segment.

to the overhead of saving its local state, this requires that the saved state be transferred to the local MSS over the wireless cell. Thus, the checkpointing algorithm must specify *when* should a MH checkpoint its local state, i.e. what is the maximum interval (sequence of events) that it can wait between two successive local checkpoints. Note that this decision has to be taken independently by each MH since the checkpointing algorithm cannot be synchronous (as was explained earlier).

4.3 The *two phase* rule

The *two phase* rule addresses the second issue mentioned above, viz. when should a MH checkpoint its local state so that a consistent global checkpoint can be formed out of the asynchronously recorded local checkpoints. To illustrate the problems that arise when each MH checkpoints its local state independently, consider a system with three MHs $h1$, $h2$ and $h3$ shown in Fig. 2. Let $C1$, $C2$ and $C3'$ be the most recent local checkpoints of the MHs $h1$, $h2$ and $h3$ respectively. Let $C3$ be a previous local checkpoint of $h3$ stored at a MSS different from that of $C3'$. Now assume that the checkpointing algorithm begins to construct a global checkpoint starting with the $C3$. Inclusion of $C3$ in the global checkpoint implies that the event $recv(M2)$ is included; therefore, to keep the global checkpoint consistent, $send(M2)$ and thereby $C2$ needs to be included as well in the global checkpoint. Since $recv(M1)$ is included in $C2$, and thereby the global checkpoint, the event $send(M1)$ should also belong to the global checkpoint. The earliest local checkpoint at $h1$ after the event $send(M1)$ is $C1$ and so $C1$ is selected as the representative checkpoint of $h1$. But, inclusion of $C1$ implies $recv(M3)$ belongs to the global checkpoint and so $send(M3)$ should also be included to keep the global checkpoint consistent. However, this is possible only if the representative checkpoint selected for $h3$ is $C3'$ and not $C3$, i.e. the checkpointing algorithm is unable to construct a consistent global checkpoint starting from any given local checkpoint of a MH, viz. local checkpoint $C3$ of $h3$.

As the above example shows, a checkpointing algorithm must ensure that though MHs may independently checkpoint their local states, they should do it

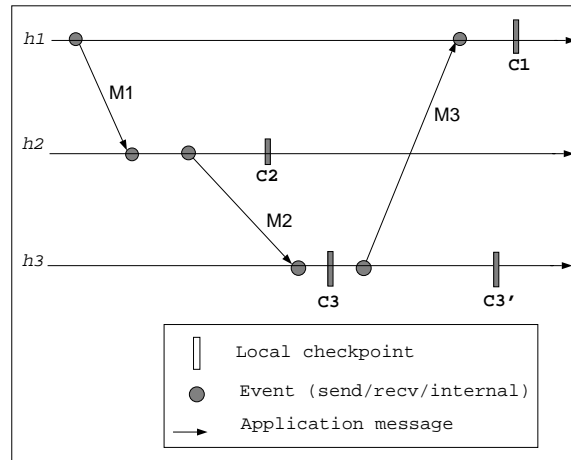


Figure 4.1: The problem with independently recorded local checkpoints

according to some *local protocol*. Given a MH's local checkpoint as a starting point, the algorithm should be able to select a local checkpoint for every other MH so that this set of local checkpoints forms a *consistent* global checkpoint. The *two phase* rule below dictates when a MH h_i should checkpoint its local state:

The two phase rule

1. If ($phase_i == SEND$) and it receives a message M , then it checkpoints its local state and switches to the RECV phase, i.e. $phase_i := RECV$, before delivering the message to the application.
2. If ($phase_i == RECV$), then prior to sending a message M to another MH, it switches to the SEND phase, i.e. $phase_i := SEND$.

A MH stores a copy of every message sent by it in a *message log*. Whenever a MH checkpoints its local state, the saved state information and the message log up to that point is transmitted to the local MSS. We next show that this rule solves the problem illustrated in Fig. 4.1.

We first define the *depends on* (" \succ ") relation between local checkpoints of MHs. A local checkpoint, $ckpt_a$ of MH h_i depends on a local checkpoint $ckpt_b$ of a MH h_j , i.e. $ckpt_a \succ ckpt_b$ if and only if there exists messages m and m' such that :

- $recv(m')$ belongs to $ckpt_a$, and
- $send(m)$ belongs to $ckpt_b$, and

- $send(m) \rightarrow recv(m')$, where “ \rightarrow ” is Lamport’s *happened before* relation [49]

Intuitively, $ckpt_a \succ ckpt_b$ means that if $ckpt_a$ is included in a global checkpoint then $ckpt_b$ should also be included to maintain consistency of the global checkpoint.

Lemma 1. If every MH follows the *two phase* rule to independently checkpoint its local state, then it is not possible that $ckpt_{i1} \succ ckpt_{i2}$, where $ckpt_{i1}$ and $ckpt_{i2}$ are local checkpoints of the same MH h_i such that $ckpt_{i2}$ is taken after $ckpt_{i1}$.

Proof sketch. It follows from the definition of the “happened before” relation that if event e happened-before event f , then e must have occurred at an earlier time than f . Thus, if $ckpt_{i1} \succ ckpt_{i2}$, then there must exist messages m and m' such that

1. $recv(m')$ belongs to $ckpt_{i1}$, and
2. $send(m)$ belongs to $ckpt_{i2}$ but not to $ckpt_{i1}$, and
3. $send(m) \rightarrow recv(m')$

However, (1) and (2), together with the *two phase* rule, implies that $send(m)$ occurred after $recv(m')$ which violates (3). Thus, by contradiction, it is not possible that $ckpt_{i1} \succ ckpt_{i2}$. •

The *two phase* rule ensures that no dependency is created between two local checkpoints of the same MH. Two other issues needs to be now addressed by our checkpointing algorithm :

1. Given a local checkpoint of some MH as a starting point, which local checkpoints should be selected for the other MHs so that the selected set of local checkpoints forms a consistent global checkpoint ?
2. Each of these local checkpoints could be located at any of the MSSs: how can these checkpoints be located with minimum search effort?

Both these issues are tackled by piggybacking control information with every application message. The idea of using a vector of state variables to capture dependency between hosts has been widely used before, such as in [59, 20]. In

our case, each MH h_j maintains two vectors, $LOC_i[1..N_{mh}]$ and $CKPT_i[1..N_{mh}]$. $CKPT_i[i]$ is initialized to 1; other elements initially are 0. Whenever, h_j checkpoints its local state, it increments $CKPT_i[i]$; $CKPT_i[i]$ thus always stores the sequence number assigned to the next local checkpoint of h_j . $LOC_i[i]$ stores the MSS-id of the MSS currently local to h_j ; it changes its value whenever h_j moves to a new cell. The role of these two arrays is as follows: if $CKPT_i[j]$ is p and $LOC_i[j]$ is q , then a global checkpoint that includes $CKPT_i[i]$ as the representative local checkpoint of h_i must include the p^{th} local checkpoint of MH h_j which is located at the MSS, MSS_q . Piggybacking the LOC array with every application message requires that a MH checkpoint its local state prior to *every* move, in addition to those mandated by the *two phase* rule.

4.4 The checkpointing algorithm

The overall checkpointing algorithm consists of two parts: one executed by every MH to independently checkpoint its local state based on the *two phase* rule (with appropriate modifications) and the other initiated from a MSS (possibly, on behalf of a local MH) to collect one local checkpoint per MH at a common location to construct a global checkpoint.

4.4.1 Algorithm executed at a MH h_i

Sending a message M

- If $phase_i$ is RECV, then $phase_i$ is set to SEND.
- Piggyback a copy of $CKPT_i[]$ and $LOC_i[]$ along with the message M , i.e.
 - $M_CKPT[] := CKPT_i[]$
 - $M_LOC[] := LOC_i[]$
- Append M to the message log, log_i .

Receiving a message M from MH h_j

- If $phase_j == SEND$, then execute *checkpointing procedure*.
- $phase_j := RECV$

- $1 \leq \forall j \leq N_{mh}$, if $M_CKPT[j] > CKPT_i[j]$

then

$CKPT_i[j] := M_CKPT[j]$

$LOC_i[j] := M_LOC[j]$

- Deliver message to the application.

Switching cells

- Execute *checkpointing procedure*.
- Leave current cell and enter a new cell under a MSS, say MSS_p .
- $LOC_i[i] := p$

Disconnected mode of operation

- Prior to disconnecting from the network, h_i executes the *checkpointing procedure*.

Checkpointing procedure

- Record the local state, $local_state_i$ of the distributed application running on h_i .
- Transfer $local_state_i$, log_i , $CKPT_i[]$ and $LOC_i[]$ to the local MSS.
- $CKPT_i[i] := CKPT_i[i] + 1$

4.4.2 Algorithm executed at the MSSs

Given a local checkpoint of a MH h_i consisting of $local_state_i$, log_i , $LOC_i[]$ and $CKPT_i[]$ ² from which to create a global checkpoint, a MSS initiates an execution of the following algorithm to collect a consistent set of local checkpoints.

Initiator MSS

- $1 \leq \forall j \leq N_{mh}$, send a *request*(h_j , $CKPT_i[j]$) message to MSS_p , where p is the value $LOC_i[j]$, asking for the local checkpoint of h_j with a sequence number $CKPT_i[j]$.

² Note that these refer to the values that were saved as part of the local checkpoint and not the current values at h_i .

Responding to request(h_j , $CKPT_i[j]$)

- If the local checkpoint of MH h_j with a sequence number $CKPT_i[j]$ is available at the MSS that received the *request()* message, then it immediately replies to the initiator MSS with the local checkpoint in question; otherwise, it waits for h_j to take its next checkpoint (its sequence number will equal $CKPT_i[j]$). In this case, the MH has not yet taken a local checkpoint with the specified sequence number. Moreover, it can be easily shown that the succeeding local checkpoint of the MH must have a sequence number that equals that specified in the *request()* message. Therefore, the MH will record and transfer the required checkpoint to the MSS prior to its next disconnection or before it receives (delivers) any further application message. However, neither of these two events are guaranteed to occur; so, the MSS may force the MH to checkpoint its state. It is possible that h_j fails before taking the required checkpoint: in this case, the MSS informs the initiator of h_j 's failure which will then initiate appropriate recovery protocols.

Reconstructing a consistent global checkpoint

- The initiator waits till it receives a local checkpoint for each MH. If a MSS replies that a desired local checkpoint of a MH is not available due to failure of the MH, then the global checkpointing algorithm aborts and necessary recovery protocols may be initiated. It is shown in Lemma 2 that the collection of local states, one per MH, constitutes a consistent global checkpoint.
- As part of the local checkpoint of a MH h_i , the message log, log_i , of all messages sent by h_i is available to the global checkpointing algorithm. However, it has to identify the messages that were sent by h_i but which were not received before the destination MH was checkpointed (that was included in the global checkpoint). To accomplish that, each MH h_i needs to maintain two vectors, $SENT_i[N]$ and $RECV_i[N]$, whose current values are saved whenever h_i checkpoints its local state. Prior to sending an application message to a MH h_j , h_i increments $SENT_i[j]$ and piggybacks this number on the message. Since messages between MHs are received in fifo order, h_j increments $RECV_j[i]$ on

delivery of this message. The global checkpointing algorithm uses the contents of these two arrays, saved with the local checkpoints of h_i and h_j , to identify the sequence of “in-transit” messages from h_i to h_j ; those messages in log_i with a sequence number greater than $RECV_j[i]$ constitute the sequence of “in-transit” messages.

4.4.3 Correctness sketch

Lemma 2. Let $CKPT^{p_i}[]$ be the CKPT array stored with the p_i^{th} local checkpoint of a MH h_i . If for each MH h_j , the local checkpoint selected is $CKPT^{p_j}$, where p_j is the value of the element $CKPT^{p_i}[j]$, then this collection of local checkpoints forms a *consistent* global checkpoint.

Proof sketch. By contradiction. Assume there exists a message m sent from h_j to h_k such that

1. the event $send(m)$ is not included in the global checkpoint, i.e. $m_CKPT[j] > CKPT^{p_i}[j]$, and
2. the event $recv(m)$ is included in the global checkpoint, i.e. $CKPT^{p_k}[] \geq m_CKPT[]$

From (1) and (2) and the rules for updating the CKPT arrays on receipt of a message, it follows that $CKPT^{p_k}[j] > CKPT^{p_i}[j] \dots$ (I)

Given that $CKPT^{p_i}[j]$ is p_j , it can be shown that there exists messages m_k and m_i , such that m_k is sent from h_k and m_i is received at h_i , such that

- $send(m_k) \rightarrow recv(m_i)$, i.e. $m_k_CKPT[] \leq m_i_CKPT[]$
- m_k is sent before the p_k^{th} local checkpoint of h_k and after the $(p_k-1)^{\text{th}}$ checkpoint; the *two phase* rule ensures that no messages are delivered at h_k after sending m_k and before taking the p_k^{th} local checkpoint and so, $m_k_CKPT[] = CKPT^{p_k}[]$.
- m_i is received before the p_i^{th} checkpoint at h_i , i.e. $CKPT_i^{p_i}[] \geq m_i_CKPT[]$.

It thus follows, $CKPT^{p_i}[] \geq CKPT^{p_k}[] \dots$ (II)

From (I) and (II), $CKPT^{p_i}[j] > CKPT^{p_i}[j]$, which is a contradiction. •

4.5 Discussion

4.5.1 Maintaining data structures at the MSSs

Power consumption at a MH is often a critical issue since mobile hosts such as laptops or palmtops use batteries as power source. Transmitting a message from a MH to its local MSS consumes power proportional to the size of the message. Thus, the size of the local checkpoints and the information piggybacked with each application messages should be kept as small as possible. The key idea here is to note that all communication to and from a MH passes through its local MSS: therefore, the local MSS can also execute the bulk of the checkpointing algorithm on behalf of a MH, i.e. all the arrays, viz. $CKPT[]$, $SENT[]$ and $RECV[]$, and the message logs can be maintained at the local MSS. When a MH sends an application message, it needs to first send the message to its local MSS over the wireless cell. The MSS can then piggyback a copy of the appropriate vectors onto the application message (which would otherwise have been done by the MH itself) and then route it to the appropriate destination. Conversely, when the MSS receives an application message to be forwarded to a local MH, it first updates the relevant vectors that it maintains for the MH, strips all piggybacked information from the message thereby reducing its size and only then does it forward the message to the MH. Thus, the MH sends and receives application messages that do not contain any additional information piggybacked by the checkpointing algorithm; it is only responsible for checkpointing its local state appropriately and transferring it to the MSS. Note that shifting the data structures especially the message log to the MSS also reduces the size of the local checkpoints of the MH.

4.5.2 Location-dependent cost of a global checkpoint

For a distributed application executing on fixed hosts, local checkpoints of a particular host are always stored at the same “location”, viz. the local stable storage

of the host. In contrast, local checkpoints of the same MH may be stored at the stable storage of different MSSs: a different set of MSSs need to be contacted every time a global checkpoint is reconstructed. Thus, the “cost” of reconstructing a global checkpoint varies from checkpoint to checkpoint in a mobile environment, while it is fixed for a distributed application executing solely on fixed hosts.

The overall cost of reconstructing a global checkpoint in the mobile environment can be quantified by assigning a cost to transfer a local checkpoint of a MH from one MSS to another, i.e. each MSS, M_i maintains an array $DIST_i[M]$, where $DIST_i[j]$ is the cost of transmitting a message between M_i and M_j . Thus, the cost of reconstructing a global checkpoint *at* MSS M_i , given a local checkpoint of a MH h_l to begin with, is proportional to $\sum_{j=1}^{N_{mh}} DIST_i[CKPT_l[j]]$.

4.5.3 Discarding local checkpoints

Let $CKPT_i^k[]$ be the array associated with the k^{th} local checkpoint of MH h_i . This checkpoint can be deleted from the stable storage of a MSS, if the following condition is satisfied:

For each MH h_j , $CKPT_j^{\text{last}(j)}[i] > k$

where $\text{last}(j)$ is the ordinal number of the most recent checkpoint of h_j that is stored at any MSS.

This condition implies that if a global checkpoint is constructed from the most recent local checkpoint of any MH, then a local checkpoint of h_i that is more recent than the k^{th} checkpoint, will be included in the global checkpoint; therefore, the k^{th} local checkpoint of h_i can be deleted. Garbage collection procedures based on the above condition, can then be executed amongst the MSSs independent of a checkpointing algorithm execution.

4.6 Chapter summary

This chapter presented the problems associated with recording a consistent global checkpoint of mobile hosts. Synchronous checkpointing mechanisms incur a search cost to deliver control messages to mobile hosts. Asynchronous checkpointing

mechanisms, on the other hand, are confronted with lack of stable storage at the mobile hosts to store local checkpoints. To cope with these problems, the solution presented in this chapter relied on MSSs to store the local checkpoints of MHs. However, since MHs move between different cells, location of the stored checkpoint needs to be propagated to other MHs. At the same time, it needs to be guaranteed that a global checkpoint can be reconstructed *a posteriori* from the local checkpoints taken by MHs independent of each other. To this end, we proposed the *two phase* rule, which requires a MH to record its local checkpoint when it switches from a *send* to a *receive* phase. The *two phase* rule however does not specify which set of local checkpoints form a consistent global snapshot. Dependency between local checkpoints is propagated amongst MHs by piggybacking each application message with LOC[] and CKPT[] arrays.

The problem of recording a consistent checkpoint is not unique to mobile hosts; it exists in the context of static hosts as well. However, existing solutions for static hosts are not viable for mobile hosts, and hence the need for new solutions specifically targeted at mobile hosts.

The next chapter introduces a problem that arises solely due to host mobility, viz. delivering multicast messages to mobile hosts with the guarantee that a message is delivered to a recipient from exactly one cell. Mobility of a host between cells allows for the possibility that it could pick up copies of the same message from multiple cells. On the other hand, it is also possible for an intended destination of a message not to receive it all due to mobility.

Chapter 5

Delivering Multicast Messages to Mobile Hosts

This chapter considers the effects of host *mobility* on reliable delivery of *multicast* messages. Multicasting necessitates sending copies of a message to different MSSs (cells); due to a MH's ability to move between different cells, a multicast message may be delivered to a recipient at multiple cells or may not be delivered at all depending on the recipient's mobility [2]. Delivering multicast messages, regardless of the location of recipients, can be used to invalidate or update copies of shared data cached at different mobile hosts (e.g., files, stock market information), or for supporting multiperson interaction based on group communication between mobile computers [31].

Our work differs from current research on network protocols for mobile hosts [18, 40, 61, 63] in three key respects. First, a common design goal of the referenced work is to keep host mobility transparent to transport layer and above, and therefore, tackle host mobility through new addressing mechanisms and protocols at the network and link layers. Second, they are primarily concerned with *routing* messages (datagrams) to/from a MH regardless of its location in the network. Third, their focus has been restricted to point-to-point messages, and multicasting to MHs has not been addressed so far. On the other hand, there exists network-layer multicast protocols for networks with *static hosts*, e.g. [15, 28, 56]; however, extending such protocols to handle host *mobility* is non-trivial and we are not aware of any work in this direction.

This chapter presents algorithms/ upper layer protocols for delivering multicast messages to mobile hosts. The multicast schemes rely on an underlying transport mechanism that delivers messages in sequence between MSSs within the "wired" network, *or* between a MSS and a local MH within the same wireless cell; they do not require any network layer support to route messages from a MSS to a "non-local" MH, and vice versa.

Our multicasting protocols are explicitly designed to respect the two main physical constraints of a mobile computing environment: low-bandwidth of the wireless connection between a MSS and a local MH, and tight constraints on power consumption at the MHs. Additionally, given the substantially greater processing and communication capability of static hosts, the protocols are structured in accordance with the *two-tier* principle so that the communication and computation requirements of a protocol execution are fulfilled within the static segment of the network to the extent possible. This is accomplished by partitioning the multicast protocols into two interactions: one, that occurs entirely within the static network between MSSs and the second, that takes place solely within a single cell between a MSS and a local MH. Such a split is not possible with existing network-layer “mobile-IP” protocols [18, 40, 61, 63] since host mobility is not visible above the network layer under these schemes.

This chapter is organized as follows. First, we identify the effects of host mobility on delivery of multicast messages to MHs: a multicast message may be delivered to a recipient at multiple locations (cells) or may not be delivered at all depending upon the recipient’s mobility. We then present algorithms for delivering multicast messages to mobile destinations from *at least one* location, and from *at most one* location, and combine the two algorithms for message delivery from *exactly-one* location. Reliable delivery of a multicast message to mobile hosts requires that each recipient acknowledge receipt of the message. However, sending each ack separately to the initiator of the multicast, would lead to an *ack implosion* problem. This is tackled in our multicast schemes by having the local MSS collect acks from all recipients within its cell, before forwarding a list of such acks to the initiator.

5.1 Design Considerations

5.1.1 Constraints associated with mobile hosts

Mobile hosts, such as palmtops and “personal digital assistants”, have significantly lower processing speed and memory capacity compared to their desktop counterparts [6, 37, 52]. It is believed that irrespective of technological advances, the ability

to be portable or mobile will cause mobile computers to lag static hosts in these respects. Second, the wireless link between a mobile host and its local MSS has a significantly lower bandwidth than a link between static hosts [9, 52]. These two points imply that protocols for mobile hosts should rely more on the computation and communication power of the static network than those of the MHs.

A moving host cannot draw on a tethered source of power, and is forced to use a stand-alone power source such as battery cells. Given the limited lifetime of such a source [9, 17, 39], and the fact that CPU operations, memory accesses, data transmission and reception all consume power, the number of operations executed by a MH should be minimal to reduce power consumption. This can be achieved if the relevant state information and data structures required for a protocol execution are distributed within the static segment of the network; messages to update the state information will then be processed at the static hosts (instead of at the MHs).

To reduce power consumption further, mobile hosts often operate in a “doze” mode, i.e. the computer shuts/slow down most of its system functions and only listens for incoming messages [11]. On reception of a message, the mobile host resumes its normal mode of operation. The purpose of doze mode would be thwarted if a significant portion of the protocol’s state is maintained at a MH since, control messages to update the protocol’s state would force the MH to resume its normal mode.

In networks where neighboring cells overlap, a MH would not lose network connectivity while moving between adjoining cells. However, when cells do not overlap, a MH may be physically unreachable by the rest of the network for brief intervals of time, while moving from one cell to the next. In such a situation, it is of advantage that the MSSs buffer a multicast message, since the message could then be delivered to the MH once it enters the coverage area under a MSS.

To summarize, the physical characteristics of a mobile environment dictate that protocols for MHs be structured so that the communication and computation load of a protocol execution is fulfilled within the static network to the extent possible.

5.1.2 Effects of host mobility on multicast delivery

Multicasting necessitates sending copies of a message to multiple locations within the static network. To ensure reliable delivery, messages may be buffered at multiple MSSs. Since a MH could connect to the static network from different MSSs at different times, it may receive more than one copy of the message. Alternatively, it may not receive the message at all. The following factors contribute to the problem: (1) two MSSs may receive copies of the same multicast message at different times, (2) the MSSs may schedule the copies for wireless transmission at different times within their respective cells, (3) latency of the *wireless* network inside a cell, and (4) a MH can lose network connectivity while switching cells, especially when cells do not overlap.

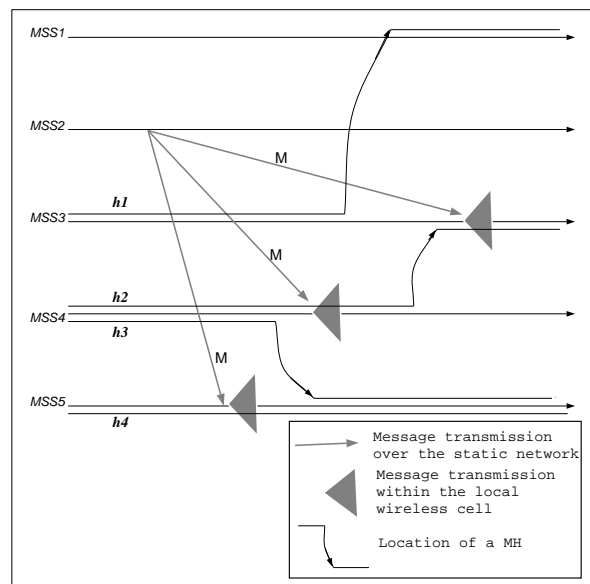


Figure 5.1: Effect of mobility on delivery of multicast messages

Consider a multicast message m sent from MSS2, intended for MHs $h1$, $h2$, $h3$ and $h4$. A copy of m is sent to all cells where a recipient is located i.e., MSS3, MSS4 and MSS5. Each recipient MSS is now responsible for forwarding m to local mobile destinations within its wireless cell. Now, consider the following cases:

- $h3$ does not receive m from either MSS4 or MSS5 in spite of the fact that both MSSs received m .
- $h2$ receives a copy of m from both MSS4 and MSS3.

- *h4* does not change its location while the multicast is in progress, and hence receives *m* exactly once from MSS5. Contrast this with *h1* and *h2*, which violate exactly-once delivery solely on account of their *mobility*.

The above example also raises two important questions viz., which MSSs should be sent a copy of *m*, and how long should a copy be buffered at a MSS before it can be safely deleted? For example, *h1* moves to the cell under MSS1 without receiving *m* at MSS3. To ensure that *h1* receives *m* from *at least one* cell, it becomes necessary to send a copy to MSS1. Next, consider the movement of *h3*. If *h3* should receive *m* from *at least one* location (in this case MSS5), then the copy at MSS5 cannot be deleted till *h3* receives *m*. If MSSs do not track the current locations of MHs, then to ensure at least once delivery, it is necessary to send a copy of *m* to all MSSs. Moreover, MSSs can delete their local copies only after *m* is delivered to *all* intended recipients.

5.1.3 Multicast vs. multiple unicasts

It is possible to achieve the functionality of multicasting a message to a set of MHs by separately sending a copy to each recipient using point-to-point mobile internet-working schemes such as [18, 40]. However, this violates the two primary physical constraints of a mobile computing environment: (1) it requires the MH sending the multicast, to transmit a copy separately to each recipient; this transmission overhead increases the power consumption at the sender MH, and (2) it results in a poor utilization of the wireless link from the sender MH to its local MSS since multiple copies of the same message are transmitted on this link. Similarly, if more than one recipient of the message is located in a cell, the local MSS will unnecessarily receive multiple copies, each intended for a different local recipient. More importantly, *mobility* of the recipients poses a more serious objection to this scheme, as explained below.

Routing protocols in a network with MHs incur either a *search cost* [40] or a *inform cost* [18] to track the location of individual MHs. A search cost is incurred when a MSS must forward a message, addressed to a specified MH, to the MSS currently local to the MH, but is not aware of the MH's current location. In [40],

a MSS is then forced to query (“search”) all MSSs within a “campus” to determine the MH’s current location. Therefore, under this scheme, a search cost may be incurred for *each* recipient if a multicast is simulated by routing separate copies to the recipients.

In [18], the current location of each MH is kept at a “location directory” (LD): on each move, the MH informs the LD of its new location. To route a packet to a specific MH from a non-local fixed host, the datagram is first sent to the LD which then forwards it to the MH’s current cell. This forwarding overhead multiplies when multicast functionality is obtained by multiple unicasts. Additionally, since location information is maintained on a per-MH basis, *every* move by *every* MH leads to an update of the LD (i.e., “inform” cost). However, such fine-grained location information is not necessary for multicasting. Instead, as will be seen in chapter 6, location updates can be reduced by aggregating MHs into *multicast groups*: a set of “locations” (MSSs) is associated with each group which is updated only when a member moves to a MSS outside this set, or when a MSS in this set is no longer local to any member of the group.

5.1.4 Motivating factors

The discussion in this section provides us with the following motivating factors for structuring multicast schemes for mobile hosts:

1. Minimal participation by MHs in terms of CPU operations, memory accesses, and communication over the wireless link.
 - MHs have tight constraints on power consumption.
2. Data structures necessary to implement a multicast protocol, are stored at the MSSs on behalf of the local MHs. A MSS can filter which multicast messages need to be transmitted over the wireless cell to a local MH. This avoids
 - wasting bandwidth of the wireless link to transmit redundant copies of a message that has already been received by a local MH at another cell,

- unnecessary power consumption at the recipient MH to receive a duplicate message, and discard it subsequently.
3. The communication and computation requirements for a protocol execution should be fulfilled within the static network to the extent possible.
 - MSSs have greater processing power and storage capacity than MHs. Further, MSSs are connected via (relatively) high-speed, high bandwidth “wired” links within the static network, compared to the wireless link between a MH and local MSS.
 4. Multicast necessitates sending copies of the same message to multiple locations within the network. A *mobile* recipient may pick up the same message from multiple locations or not receive it all. This provides the motivation for designing multicast algorithms that deliver messages to recipients from *exactly one* location, irrespective of the recipients’ mobility, and which additionally respect the constraints on power consumption at the MHs and low-bandwidth of the wireless links.
 5. Multicast algorithms for mobile hosts should not be structured to deliver messages on a *per destination* basis, since this either incurs a search cost to locate individual destinations or requires each destination to inform potential senders of its current location on every move. In chapter 6, we apply the concept of a *multicast group* to MHs and associate a *host view* with each group. The host view is a set of MSSs that embodies the aggregate location information of a group. Consequently, no location updates are generated as long as a MH moves within the current host view of its group. Further, no search cost is incurred to deliver a multicast since it suffices to send a copy of the message to exactly those MSSs in the host view.
 6. In addition to the above, an important motivation for our multicast protocols is to provide *low delay* delivery of multicast messages i.e., when a MH enters a new cell, copy of a multicast message should already be available at the local MSS instead of the MSS waiting for a request from the MH to fetch a copy from the initiator. Low-delay delivery has motivated new

multicasting protocols for internetworks of static hosts [27]. It has been argued in [27, 25] that as resources available in modern internetworks — bandwidth, processing and memory — have become cheaper and more abundant, importance of low-cost schemes has diminished in preference to schemes that aim at low-delay delivery. Based on this premise, multicast protocols in mobile environment should perform *eager communication* [25], i.e. dissemination of data in anticipation of need rather than responding only on request. Since, in a mobile computing environment, resources within the “wired” segment of the network are abundant in contrast to those in the wireless segments, copies of a multicast message should be available at MSSs in anticipation of need, i.e. when a MH enters a new cell, a copy should be available at the local MSS for wireless transmission to the MH. The criterion of eager communication thus nicely complements our design choice of splitting the desired overall functionality of multicast protocols into one interaction amongst the static network, and the other, that occurs within the local wireless cells.

In the next section, we present a multicast algorithm that provides exactly-once delivery of messages to MHs, independent of their location. It is specifically tailored to take advantage of the better communication and processing capability of the MSSs, minimize message transmission over the wireless links and conserve power at the mobile hosts. In chapter 6, we combine this algorithm with a host-view membership protocol to deliver messages exactly-once to multicast groups.

5.2 Multicast Protocols

This section presents protocols for delivering multicast messages to mobile hosts. An execution of the protocol may be requested either by a mobile host or a fixed host. When invoked from a MH, the local MSS is responsible for executing the protocol on behalf of the MH. Similarly, a non-MSS fixed host could also request an execution of the protocol; we will assume that the request will be forwarded to a MSS which will then execute the desired protocol. Thus, the term *initiator* will refer to the MSS

executing a protocol on behalf of a MH or another fixed host. Given our exclusive focus on host mobility and its impact on message delivery, we will only consider multicast messages that are addressed solely to mobile hosts.

Three schemes are presented with different message delivery semantics, viz. a MH may receive a copy from *at least one* cell, or from *at most one* cell, or at *exactly one* cell. Our system model ensures that within a cell, communication between a MH and the local MSS occurs in *fifo* order, i.e. messages are *delivered* exactly-once and in sequence. Protocols in this section share three common features: (1) an arbitrary set of MHs can be specified as the recipients of a multicast message (2) every MSS in the system receives a copy of the multicast message from the initiator, and (3) each MSS is considered to be a potential initiator of a multicast. All three schemes rely on a *handoff* procedure to transfer a MH's state from its previous cell to its current cell, after a move.

5.2.1 Mobility assumption

It is theoretically conceivable that due to a combination of very small cell sizes (e.g., “picocells”) and the speed with which a MH crosses cells (e.g., vehicular speeds), handoffs occur so frequently that no multicast message can be delivered to the MH in any of the cells which it visits. When a MH changes its location, the static segment is required to transfer the necessary state information from its previous MSS to its current MSS to facilitate delivery of multicast messages at its current location. If this handoff process cannot be completed within the time interval between a MH's entry and departure from a cell, then the MH would not receive any multicast message, even if available for delivery, in this cell. Without any expectation of performance from the static network, it is not meaningful to discuss delivery of a multicast message to a MH in at least one, and consequently exactly-one location. Therefore, we augment the system model presented in chapter 2 with the following *mobility assumption*: the static segment of the network has sufficient host processing power and communication speed, so that after a MH enters its present cell and before it subsequently leaves the cell,

- the handoff procedure is completed, and

- if there are pending multicast messages at the local MSS that can be delivered to the MH, then the MSS be able to deliver at least one such message.

The *mobility assumption* implies that the delay between a MH entering a cell, and till it receives the first available multicast message (if any), is always less than the interval of time a MH resides within any cell. Note that we require a message to be delivered *only if* it is currently available in the local MSS's buffer. This assumption will be more specifically stated later, in the context of individual multicasting schemes.

5.2.2 Handoff

A handoff is triggered when a MH “discovers” that it has moved to new cell. How exactly this discovery takes place is not crucial to our protocols, and is currently, the subject of much discussion [40, 55]. Our schemes use an abstract representation of the network-layer handoff procedure presented in [40], as described next.

Consider a MH h that physically moves from the cell under M to N (step (1) of Fig. 5.2). On discovering that it has moved to a new cell, h sends a network-layer packet, viz. *greeting*(h, M), to N supplying its own identity h and also the id of its previous MSS. N responds to h with a *greeting ack* packet. If the greeting packet is lost, the MH will simply resend it. If the acknowledgment packet is lost, the MH will send another greeting unless it gets an acknowledgment. Thus, we assume that when a MH moves to new cell, a *greeting* message is received reliably by the local MSS, as shown in step (2) of Fig. 5.2.

On receiving *greeting*(h, M), N sends *deregister*(h) to M (step (3)). M deletes h from M_Local , and transfers the data structures pertaining to h , to N via a *register*($h, <data>$) message (step (4)). In response, N adds h to the list N_Local . The lists, M_Local and N_Local , contain ids of the MHs local to the cells of M and N respectively.

So long as a MH resides within a cell, reliable and sequenced delivery of messages over an unreliable link (wireless) can be achieved through appropriate link level protocols, e.g. sliding window protocols. Thus, our multicast protocols

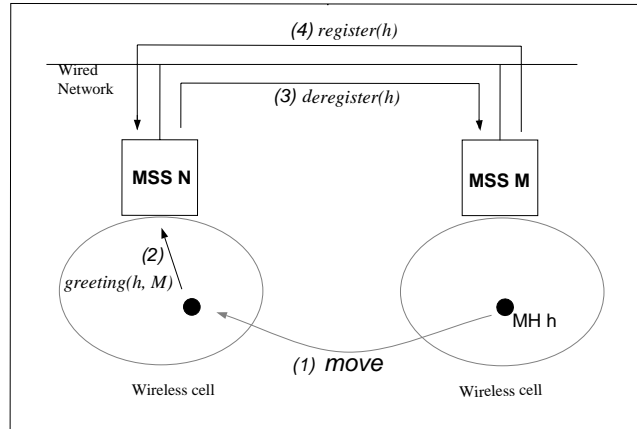


Figure 5.2: Handoff

assume bidirectional fifo communication between a MSS and a local MH. However, when a MH moves to a new cell, we need to ensure that the fifo channels between the MH and the previous MSS are flushed properly. This is done as follows.

Let a MH h switch cells from M to N . h now includes an additional parameter $seq_{M_to_h}$ in the greeting message; it is the sequence number of the last message received on the fifo channel from M to h . Also, h discards all messages received from M with a sequence number greater than $seq_{M_to_h}$ (since their delivery would violate fifo order). This number is sent to M by as an additional parameter of the *deregister()* message from N . At M , it acts as an acknowledgment of delivery of all messages to h , whose sequence numbers are less than or equal to $seq_{M_to_h}$. Similarly, M includes $seq_{h_to_M}$ in the *register* message to N , which is the sequence number of the last message received in sequence from h at M . N then sends a *init_channel* message to h ; the sequence number of this message is set to $seq_{M_to_h} + 1$. Additionally, $seq_{h_to_M}$ is piggybacked on *init_channel* which acknowledges the reception of all messages from h with a sequence number less than or equal to $seq_{h_to_M}$. Delivery of *init_channel* enables h to infer that sequence number of messages on the fifo link from h to N should begin with $seq_{h_to_M}+1$.

In essence, one fifo link is maintained *from* a MH *to* the static network irrespective of the MH's mobility: the MH acts as the sending source with its current local MSS as the receiver. Another fifo link is maintained in the reverse direction from the static network to the MH, with the local MSS as the sender.

In context of the handoff procedure outlined above, the mobility assumption in our system model ensures that, after h moves into N 's cell, the MSSs M and N complete the *deregister/register* message exchange following which h receive the *init_channel* message from N prior to its next move.

5.2.3 at least-once message delivery

The scheme to deliver a message m to a set $dests(m)$ of mobile hosts, is as follows:

1. The initiator M sends a copy of m , along with $dests(m)$, to all MSSs in the system.
2. On receiving m , a MSS N buffers the message and creates an empty list, $ack_list(m)$, to keep track of those local MHs to which m is delivered.
 - a. For every local MH h , that is included in $dests(m)$, N transmits m over the wireless cell to h .
 - b. When h acknowledges receipt of m , N inserts h in $ack_list(m)$ and deletes h from $dests(m)$.
 - c. When $dests(m)$ no longer includes a local MH and $ack_list(m)$ is non-empty, N sends $ack_list(m)$ to M . N then re-initialises $ack_list(m)$ to an empty list.
 - d. If a MH, that is included in $dests(m)$, enters the local cell, N will then re-execute steps (a), (b) and (c) to deliver m to the MH.
3. When M receives a $ack_list(m)$ message, it deletes those MHs from $dests(m)$ that are listed in $ack_list(m)$. When $dests(m)$ becomes empty i.e., each intended recipient has acknowledged delivery of m , M sends a *delete(m)* message to all MSSs.
4. Receipt of *delete(m)* at a MSS results in deletion of m and the associated $ack_list(m)$, from the local buffer.

The at least-once protocol does not specify how M_{init} should send a copy of m to all MSSs. Since the static network offers reliable delivery of point-to-point messages, M_{init} could send m to each MSS via a point-to-point messages. On the other hand, if the static network offers multicast support for efficient delivery of multi-destination

messages such as [28, 56], then it could be utilized to efficiently deliver a copy of m to all MSSs. Similarly, within a single cell, the local MSS could forward a copy of m to individual mobile recipients via point-to-point messages, or could utilize the broadcast capability of the wireless medium. The at least-once protocol does not mandate a specific mechanism for either delivery of multi-destination messages within the static network, or within a single cell.

Effects of host mobility are manifested in the at-least once delivery scheme in the following ways:

1. Mobility of recipients require m to be sent to each MSS in the system. Since, no MH informs a non-local MSS of its current location (to avoid inform cost per move), the initiator is unaware of a recipient's current location (unless it is a local MH). Secondly, the initiator should not incur a search cost per recipient to separately locate each MH in $dests(m)$. Since, a destination MH could be local to any of the MSSs in the system, the initiator sends a copy of the multicast message to each MSS; the overhead of propagating m to all MSSs is amortized by the number of recipients of m .
2. In spite of reliable message delivery within the fixed network and locally within a cell, a initiator needs to wait for an explicit acknowledgment of delivery from each mobile destination. This need arises solely due to *mobility* of recipients and not due to reliability of communication links. No MSS can delete m from its local buffer till every recipient receives m ; otherwise, a MH, that has yet to receive m , may enter a cell where m is no longer available for delivery.
3. For efficiency, a MSS collects as many acks as possible into a single $ack_list()$ before transmitting them to the initiator. It thus delays sending the list as long as there exists a MH in its cell that belongs to $dests(m)$, but has not received m .
4. By virtue of its mobility, a MH may receive copies of m from *different* cells.

5.2.4 at most-once message delivery

Copies of a *multicast* message are available for delivery at multiple MSSs, under

the at least-once delivery scheme. A scheme for at most-once delivery must ensure that a recipient does not receive the message from more than one MSS. Further, to conserve battery life at MHs and reduce wireless transmissions, a multicast message should not even be *transmitted* to a MH if it has already acknowledged delivery of the message in another cell. This requirement immediately eliminates schemes that first ensure at least-once message delivery and then discard subsequent copies at a recipient, since redundant message transmissions wastes wireless bandwidth and unnecessarily consumes battery power at the recipients. A secondary objective is to deliver the multicast to as many intended recipients as possible; otherwise, the at most-once delivery property can be trivially satisfied by not delivering the message to any intended recipient. The following scheme ensures delivery of a multicast message to a MH from at most one MSS:

1. Each MSS maintains a sequence counter that is incremented whenever it initiates a new execution of the protocol, and is assigned as the sequence number of the message being multicast. To multicast m , the initiator M sends a copy to all MSSs along with $dests(m)$; let the sequence number assigned to m be m_seq .
2. Each MH h is associated with an array $h_RECD[1 \dots N_{mss}]$, where N_{mss} is the number of MSSs in the system. The array is *stored at the local MSS*.
3. For each local MH h , a MSS checks if h belongs to $dests(m)$ and $h_RECD[M] < m_seq$. If so, then m is transmitted over the wireless link to h . When h acknowledges receipt of m to the local MSS, $h_RECD[M]$ is set to m_seq .
4. A MSS may delete m anytime after step (3) is completed.
5. When a MH h switches cells, $h_RECD[]$ array is included in the *register()* message during handoff.

The key idea of the protocol is to store $h_RECD[]$ at the local MSS, instead of at the MH h itself. The local MSS *transmits* a message to a h only if the conditions in step (3) are satisfied; redundant wireless transmissions are thus avoided if h has already received the message elsewhere. Further, size of the array is proportional

to N_{mss} and not to the number of MHs; this is an advantage since MHs can be expected to vastly outnumber MSSs. Lastly, during handoff, it is the MSSs that are responsible for transferring the protocol “state” for a MH i.e., $h_RECD[]$; a MH is only required to inform (“greet”) the MSS of its entry to a new cell.

5.2.5 Exactly-once message delivery

Exactly-once delivery is achieved by ensuring both at most-once *and* at least-once delivery. The protocol for exactly-once delivery is thus based on the following observations:

- Copies of a multicast message should be available at each MSS.
- An explicit acknowledgment of message delivery is required from each mobile recipient.
- Associating a $RECD[]$ array with each MH ensures at most-once message delivery to the MH.
- Due to low bandwidth of wireless links and tight power constraints of MHs, the $RECD[]$ array is stored at the local MSS, and transferred within the fixed network during handoff.

The protocol consists of three modules. The WIRED module is responsible for all communication between a initiator and other MSSs, while the WIRELESS module controls communication between a MSS and local MHs. The HANDOFF module transfers relevant state information between MSSs when a MH changes its cell.

5.2.5.1 Data structures

At a MSS M , a list of local MHs is stored as M_Local . For each MH h in M_Local , M maintains an array $h_RECD[1..N_{mss}]$.

A pending multicast message m is stored in M_Buffer along with two lists, $dests(m)$ and $ack_list(m)$, until a $delete(m)$ message is received from m 's initiator. Incoming multicast messages are always added to the end of M_Buffer , while a message may be deleted from anywhere in M_Buffer (upon receipt of a $delete()$ notification).

Messages scheduled for delivery to local MHs over the wireless cell are placed in $M_Transmit$. For each local MH, $M_Transmit$ contains at most one message awaiting delivery. It is not required that the WIRELESS module transmit messages from $M_Transmit$ in the order in which they were inserted; the module is free to implement any scheduling policy appropriate for the data link and physical layer protocols it uses for message transmission over the wireless medium.

5.2.5.2 WIRED module

To multicast m , a MSS M initiates the protocol as follows:

- A1. M increments a local counter, and assigns its current value as m_seq , the sequence number of m . M then sends $multicast(m, dests(m))$ message to all MSSs.
- A2. Recipient MSSs reply to M with their respective $ack_list(m)$ messages. A MH included in a $ack_list(m)$ message, is deleted from $dests(m)$; when $dests(m)$ becomes empty, M sends a $delete(m)$ message to all MSSs.

On receiving $multicast(m, dests(m))$, a MSS N executes the steps below:

- B1. m is appended to N_Buffer , along with $dests(m)$ and an empty $ack_list(m)$.
- B2. **for** each MH h such that $(h \in dests(m))$ **and** $(h \in N_Local)$
 - if** $N_Transmit$ does not contain an entry $\langle m', h \rangle$
 - /* there is no other message m' awaiting delivery to h */*
 - then**
 - if** $(h_RECD[M] < m_seq)$... **(Z1)**
 - then** insert $\langle m, h \rangle$ in $N_Transmit$
 - else** delete h from $dests(m)$
- B3. When N receives $delete(m)$ from M , the entry $(m, dests(m), ack_list(m))$ is deleted from N_Buffer .

The WIRELESS module delivers messages from $N_Transmit$ to the target MH. When h acks receipt of m to the WIRELESS module, the WIRED module at N is notified and it executes as follows:

C1. $h_RECD[M] := m_seq$

C2. h is added to $ack_list(m)$, and deleted from $dests(m)$.

C3. $check_ack_list(m)$ ¹ ... **(Z3)**

C4. **if** ($h \in N_Local$)

then */*h has not left the cell; so deliver the next applicable message in N_Buffer to h */*

*/*Let the sequence of messages in N_Buffer from the front to the rear, be $m_1, m_2, \dots, m_{|N_Buffer|}$ and let m occupy the k^{th} position in this sequence. */*

for $i := k+1$ to $|N_Buffer|$

if ($h_RECD[init_m_i] < m_i_seq$) ... **(Z1)**

then

if ($h \in dests(m_i)$)

then

— add $\langle m_i, h \rangle$ to $N_Transmit$

— exit **for** loop

else

— $h_RECD[init_m_i] := m_i_seq$... **(Z2)**

else

if ($h \in dests(m_i)$)

then

— h is deleted from $dests(m_i)$

— $check_ack_list(m_i)$... **(Z3)**

¹ $check_ack_list(m)$ is a procedure defined as follows:

if ($dests(m) \cap N_Local = \emptyset$) **and** $ack_list(m) \neq \emptyset$

/ there are no local MHs yet to receive m , and at least one ack from a local recipient has not been forwarded to the initiator of m */*

then

send a copy of $ack_list(m)$ to $init_m$

$ack_list(m) := \emptyset$

5.2.5.3 HANDOFF module

Let a MH h switch cells from M to N . At M , the HANDOFF module begins execution on receiving a $deregister(h, seq_{M_to_h})$ message from N .

- D1. h is deleted from M_Local .
- D2. The WIRELESS module is notified of h 's exit, and $seq_{M_to_h}$ is passed to it. The HANDOFF module then waits for the WIRELESS module to hand back $seq_{h_to_M}$, the sequence number of the last message received in order by M from h .
- D3. A $register(h, h_RECD[], seq_{h_to_M})$ is sent to N from M , and $h_RECD[]$ is deleted at M .
- D4. For each message m' in M_Buffer
 - if** ($h \in dests(m')$)
 - then** $check_ack_list(m')$... **(Z3)**

The HANDOFF module at N processes the $register()$ message as follows:

- E1. h is added to N_Local , and the $h_RECD[]$ array received with the $register()$ message, is stored at N .
- E2. The WIRELESS module is then passed the two sequence numbers, $seq_{h_to_M}$ and $seq_{M_to_h}$.
- E3. Let the sequence of messages in N_Buffer from the front to the rear, be $m_1, m_2, \dots, m_{|N_Buffer|}$.
 - for** $i := 1$ to $|N_Buffer|$
 - if** ($h_RECD[init_m_i] < m_i_seq$) ... **(Z1)**
 - then**
 - if** ($h \in dests(m_i)$)
 - then**
 - add $\langle m_i, h \rangle$ to $N_Transmit$
 - exit **for** loop
 - else**
 - $h_RECD[init_m_i] := m_i_seq$... **(Z2)**

else

if ($h \in \text{dests}(m_i)$)

then

— h is deleted from $\text{dests}(m_i)$

— $\text{check_ack_list}(m_i)$. . . **(Z3)**

5.2.5.4 WIRELESS module

It implements a bi-directional communication channel between a MH and its local MSS, using low-level protocols (physical and data link layer) suitable for the wireless medium. It is also responsible for enabling “cell discovery” by a MH, e.g. it may periodically broadcast (“beacon”) the MSS-id within the local cell, similar to the *beacon protocol* of [40].

The interaction between the WIRELESS module at N and a local MH h is as follows:

- F1. When h first enters the cell (from M), it sends a *greeting()* message. The WIRELESS module forwards this message to the HANDOFF module.
- F2. After the handoff between M and N completes, the HANDOFF module provides the WIRELESS module with two sequence numbers, $\text{seq}_{h_to_M}$ and $\text{seq}_{M_to_h}$ (step D2 of HANDOFF module).
- F3. The WIRELESS module initializes $\text{seq}_{N_to_h}$ and $\text{seq}_{h_to_N}$ as follows:

$$\text{seq}_{N_to_h} := \text{seq}_{M_to_h} + 1$$

$$\text{seq}_{h_to_N} := \text{seq}_{h_to_M}$$

It then sends a *init_channel()* message to h with a sequence number $\text{seq}_{N_to_h}$, and piggybacking an acknowledgment for $\text{seq}_{h_to_N}$.

- F4. It then continues to execute the following loop till h leaves the cell:

while ($h \in N_Local$) **do**

{

if $N_Transmit$ contains an entry $\langle m, h \rangle$

then deliver m to h , and notify the WIRED module on delivery

if a request to multicast a message is received from h , then forward it to the WIRED module

}

F5. When h leaves the cell, the WIRELESS module receives (from the HAND-OFF module) the sequence number of the last message received by h from N . If, at this time, the *Transmit* queue contains $\langle m, h \rangle$, then any of the following cases could have occurred:

- h did not receive m .
- h received m , but left M s without acknowledging its receipt. In this case, $seq_{N_to_h}$ will implicitly acknowledge delivery of m .
- h received *and* acknowledged m , prior to its departure. However, the acknowledgment may not have been processed by the WIRELESS module yet.

Thus, the WIRELESS module will inspect any unprocessed acknowledgments from h , *and* the value of $seq_{N_to_h}$, to determine if m was delivered to h . If so, the WIRED module is informed of m 's delivery, which then updates $h_RECD[]$ (steps C1, C2 and C3). When this update completes, the WIRELESS module passes $seq_{h_to_N}$ to the HANDOFF module.

5.2.6 Correctness sketch

In context of the exactly-once delivery scheme, the *mobility assumption* in our system model ensures that after h moves from M to N , the following steps can be completed before h makes its next move, i.e. leaves N 's cell :

1. The *deregister/ register* exchange takes place between M and N following which h receives the *init_channel()* message from N .
2. If the HANDOFF module inserted a multicast message in $N_Transmit$ (step E3), then it is delivered to h .

Without this assumption, it is conceivable that although a message is inserted in *Transmit* after each move, a MH is unable to receive any multicast message due to frequent handoffs resulting from small cell sizes and the physical speed with which

a MH crosses cells. Thus, if the worst-case delay in completing the above two steps is T_{move} , the maximum frequency of moves supported by the exactly-once delivery scheme is $(1 / T_{move})$.

Correctness of the exactly-once delivery protocol can be inferred from the following observations:

1. A multicast message m is buffered at every MSS till an explicit $delete(m)$ message is received. This $delete()$ message is sent by the initiator only after an acknowledgment from each MH in $dests(m)$ has been received.
2. N_Buffer is scanned for the next deliverable message for h during steps B2, C4 and E3:
 - Step E3 is executed during handoff when h enters the cell under N .
 - Step C4 is executed by the WIRED module after a multicast message has been delivered to h (by the WIRELESS module).
 - If there are no deliverable messages, then whenever a new multicast message arrives and is inserted in N_Buffer , it is checked whether the message should be delivered to h (step B2). Note that incoming messages are always added to the end of N_Buffer

The above steps enumerate all cases when N_Buffer is scanned for a deliverable message. Next, consider that in steps C4 and E3, if

$(h_RECD[init_m_i] < m_i_seq)$ **and** $(h \notin dests(m_i))$

then it is correct to assign m_i_seq to $h_RECD[init_m_i]$ in statement **Z2**, for the following reason:

- Multicast messages from the same initiator, $init_m_i$, are received in order of their sequence numbers at all MSSs (N in this case). Therefore, while scanning N_Buffer for the next deliverable message to h , if a message m_i satisfies the above condition, then it follows that all multicast messages from $init_m_i$, whose sequence numbers lie in the range $h_RECD[m_i_init]+1, \dots, m_i_seq-1$, have already been received at N . However, if they are no longer in N_Buffer , they must have been deleted, and from (1) above, it follows that they did not include h as a recipient.

Thus, if a message *addressed to h* is added to *N_Buffer*, it will eventually be checked for possible delivery to *h*. at least-once delivery can now be inferred from observations (1) and (2).

3. at most-once delivery of a message can be observed from the following conditions:

- The condition **Z1** in steps B2, C4 and E3 ensures that a message *m* is inserted in *Transmit* queue for delivery to *h* only if $h_RECD[init_m] < m_seq$.
- Step F5 of the WIRELESS module ensures that at most-once delivery is not violated during the handoff process by eliminating the possibility that a MH receives a multicast message in a cell, moves to another cell without acknowledging its receipt, and then receives it again in the second cell.

4. It can also be observed that if for a multicast message *m*, its *ack_list()* is non-empty at a MSS, say *N*, then it will eventually be forwarded to *m*'s initiator, i.e. the protocol cannot block because a MH has acknowledged receipt of *m* and yet the *ack_list(m)* containing its acknowledgment has not been sent to the initiator. When *m* is first inserted in *N_Buffer*, the list *ack_list(m)* is empty while the list *dests(m)* contains *all* intended recipients of *m*. Thereafter, if *ack_list(m)* becomes non-empty, it is forwarded to *m*'s initiator when the following condition is satisfied:

$$dests(m) \cap N_Local = \emptyset$$

This condition can be satisfied whenever a local MH that is included in *dests(m)* leaves the cell (thereby being deleted from *N_Local*) without receiving *m* (step D4 of HANDOFF module), or when a local MH acknowledges receipt of *m*, thereby being deleted from *dests(m)* (step C3). Additionally, it is also possible that a MH local to *N* may have received *m* at some other cell; in this case, this MH will still be included in the copy of *dests(m)* at *N*. Such a MH will be deleted from *dests(m)* at *N* either during handoff (step E3) or while scanning *N_Buffer* for a deliverable message (step C4). Note that whenever there is a possibility that the above condition may be satisfied, the procedure *check_ack_list(m)* is called (marked by **(Z3)** in the protocol description). Also observe that in step B2, when

N first receives m , it is superfluous to execute $check_ack_list(m)$ even though a MH may be deleted from $dests(m)$; this is because N could not have possibly delivered m to any MH and so, $ack_list(m)$ is guaranteed to be empty.

Exactly-once delivery is thus ensured by at least-once and at most-once delivery of a message.

5.2.7 Variations to exactly-once protocol

In the exactly-once protocol, whenever a MH h changes its cell, the $h_RECD[]$ array is transferred from the previous MSS to the current MSS during handoff. Instead consider the following alternative scheme, where $h_RECD[]$ is not necessarily transferred on every move:

1. A *version number* is associated with $h_RECD[]$. When h receives the first multicast message in its current cell, the version number is incremented at the local MSS; subsequent delivery of multicasts within the same cell do not increment the version number.
2. Each MSS stores a copy of $h_RECD[]$ and a associated version number.
3. When h switches cells from M to M' , the version number v' present at M' is sent with the $deregister(h)$ message to M .
4. Let v be the version number at M . If $v = v'$, then M does not include $h_RECD[]$ with the $register()$ message.
5. If $v > v'$, then M will include $h_RECD[]$ in the $register(h)$ message to M' and additionally, send a copy to all other MSSs.

The essence of the above scheme is that $h_RECD[]$ is updated at all MSSs only when a MH moves to a new location after receiving at least one multicast at its previous cell. To determine when the above variation performs better than the original protocol, we define *mobility to message ratio (mmr)* as the number of moves made by a MH to the number of multicast messages delivered to it, in a given period of time². If $mmr \geq n_{mss}$, then the proposed variation will transfer less data (in terms of $h_RECD[]$) than the original protocol. On the flip side, each MSS is required to maintain a copy of $h_RECD[]$ even though h may not be local to its cell.

² A similar metric *call to message ratio* has been proposed in [14].

Next, consider the size of $h_RECD[]$. The array has one element per MSS in the system, since we assume that MHs will far outnumber MSSs. However, when the number of MHs that send multicast messages are small, it may be better to maintain $h_RECD[]$ as an array with one element per *sender MH*.

5.3 Ordered delivery of multicast messages

It is often useful to order the delivery of multicast messages to recipients, in addition to ensuring that a recipient receives the message exactly-once, i.e. the local sequence of delivery at each mobile host is consistent with the system-wide total order. For example, let M , N and P be three multicast messages such that the total order of delivery established by the protocol is $M < N < P$. Then, at each recipient common to $Dests(M)$ and $Dests(P)$, M should be delivered before P . If such a recipient also belongs to $Dests(N)$, then the delivery of N occurs after that of M and before P at that recipient.

This section shows how multicast messages can be delivered to mobile hosts such that messages are delivered in the *same order* and each message is delivered *exactly once* to a recipient. This requires: (1) establishing a total order of message delivery, and (2) ensuring that a message is delivered to a mobile host from exactly one location (MSS). The second requirement is satisfied by our exactly-once delivery algorithm presented earlier. It now remains to combine it with a message-ordering scheme. By executing this protocol only amongst the MSSs with no participation from MHs, the choice of such a protocol can thus be guided independent of mobility considerations, and instead be based on latency, concurrency or reliability requirements. Thus, the overall protocol is a combination of two separate protocols: an “off-the-shelf” protocol that creates a system-wide delivery sequence, and the other, to deliver a message from this sequence to each MH exactly-once.

The exactly-once delivery algorithm comprised of three modules: WIRED, WIRELESS and HANDOFF. A fourth module, ORDER, is now added to the algorithm to provide the needed ordering of message delivery, as follows: to initiate a multicast on behalf of a local MH, the WIRED module at the initiator forwards it

to the ORDER module. The ORDER modules at the MSSs are jointly responsible for creating a total order of message delivery. Each WIRED module now receives multicast messages from their respective ORDER modules in the same sequence at all MSSs. Earlier, the WIRED modules at different MSSs could receive multicast messages from different initiators in no specific order. By placing a ORDER module in the communication paths between the WIRED modules, it is ensured that each WIRED module receive multicast messages (for delivery to local MHs) in the same sequence.

The ORDER module can implement any of the wide range of message-ordering protocols available for fixed networks. Broadly speaking, two approaches are used to create a total order of message delivery. In the first approach, there is no distinguished host and all hosts participate in the ordering process [19, 1]. In the second approach, a distinguished host assigns a globally-unique sequence number to a message [45, 24]. Alternatively, as in [20, 8, 57], a *token* circulates amongst all hosts and carries the sequence number to be assigned to the next message: only the current token holder can send a message to other hosts (after assigning it the sequence number currently carried by the token).

In summary, our exactly-once algorithm can be easily augmented to deliver multicast messages exactly-once and in a system-wide sequence by ensuring that the WIRED modules of all MSSs receive multicast messages in the same order.

5.4 Chapter summary

This chapter identified the effects of host mobility on delivery of multicast messages to MHs and presented algorithms for delivering multicast messages to mobile destinations from *at least one* location, and from *at most one* location, and combined the two algorithms for message delivery from *exactly-one* location. However, it is necessary for exactly-once delivery to send a copy of a multicast message to all MSSs since a message could be destined for an arbitrary set of recipients. The next chapter introduces the notion of multicast groups, and incorporates location

management into the multicasting protocol by tracking the set of MSSs that are local to at least one member of a group.

Chapter 6

Location Management for Multicast Groups of Mobile Hosts

This chapter introduces the notion of *multicast groups* of mobile hosts, and associates a “host view” with each group; the host view is a set of MSSs that represents aggregate location information of the group and changes dynamically with mobility of individual group members. A host-view membership protocol is then combined with the exactly-once multicast algorithm of chapter 5. As a result, to deliver a multicast message to a group, copies of the message are sent to only those MSSs that belong to the group’s host-view.

6.1 Overview

In the multicast schemes presented in chapter 5, recipients of a multicast message could be any arbitrarily specified set of MHs. Since in our system model, MSSs do not track the locations of MHs, an initiator must send a copy of the multicast message to all MSSs in the system. This ensures that a copy is available at the local MSS of each intended recipient, regardless of its location. If all MSSs are attached to a common broadcast network, e.g. ethernet, the initiator can reach all MSSs by a single broadcast transmission. These multicast schemes are therefore well suited for broadcast networks.

In general, it may not be feasible to send a copy of every multicast message to every MSS in the system, e.g. in an internetwork. On the other hand, ensuring at least-once delivery requires that a copy be available at the local MSS of every intended recipient. One approach to reconcile both requirements, could be to track the location of each MH individually, as proposed in [18] for routing unicast messages; however, in this scheme, *every* move by *each* MH will result in a location

update. Instead, we aggregate mobile hosts into *multicast groups* and track the locations of MHs on a *per group* basis.

The concept of aggregating a related set of processes/ hosts into a *group* has been widely used in networks comprising solely of static hosts, e.g. to provide message-ordering abstractions [20], to handle failures [19, 41], and for network-layer routing of multicast messages [28]. However, host mobility introduces a new aspect to group communication, viz. *location management* at the group level. Location of a *static* host never changes and therefore, the association between its location and its host-id is an invariant; often, the location can be deduced from the host-id (i.e., its network address). Thus, given a set of host-ids comprising a group, location of individual group members needs to be determined at most once, and does not change thereafter. This is not valid for a group of *mobile* hosts: the location of a mobile host is neither fixed nor necessarily related to its host-id. For a MH, simply being a member of a group does not provide any information to other group members, regarding its current location: this requires explicit management of location information on a *per-group* basis.

The novelty of our approach lies in using *multicast groups* to coalesce location of individual members (MHs) into an aggregate “group location”. With each multicast group G , we associate a *host view* H_G as follows: a MSS M belongs to H_G if at least one MH in G is located in its cell. Thus, instead of separately tracking the location of every MH in G , we maintain a set of locations (MSSs) H_G such that every member of G resides in some cell in H_G . This provides several advantages:

1. H_G represents the *aggregate* location information of MHs in G , and can be expected to change much less often than the combined updates to the individual locations of the MHs in G . If MHs in G move between MSSs that are already in H_G , no location updates are generated. H_G changes only when a MSS in H_G is no longer local to any MH in G , or a MH in G enters a cell whose MSS does not belong to H_G .
2. The number of copies of a multicast message that needs to be sent to MSSs within the static network is reduced. Instead of sending a copy to all MSSs

to ensure at least-once delivery, it suffices to send a message addressed to G to only those MSSs in H_G .

3. A message addressed to a multicast group is intended for delivery to all members of the group. Multicast groups thus allow a set of destinations to be identified by a single multicast group-id, thereby eliminating the need to explicitly specify a list of recipients with each multicast message. First, this reduces the overall size of the multicast message; since, a MH transmits this message to the local MSS over the wireless link (prior to the actual multicast operation), this is significant in terms of power consumed at the MH to transmit this message, and bandwidth constraints of the wireless link. More importantly, it allows a MH to request its local MSS for a multicast operation, without the MH being aware of the actual membership of the group.

On the flip side, we allow mostly closed group communication i.e., a multicast can be initiated only by a MH that belongs to the group. Without being a member of G , a MH can multicast to a group G **only if** it is currently local to a MSS that belongs to H_G . This is possible because, an actual execution of a multicast protocol is handled by a MSS and not by a MH; a MH merely requests its local MSS to carry out a multicast. Therefore, so long as a MSS has the necessary “state” information for a particular multicast group G (by virtue of being a member of H_G), it can satisfy requests for multicasting to G from local MHs that do not belong to G .¹

Lastly, guided by the *two tier* principle, the computation and communication necessary to maintain the host-view associated with a multicast group is fulfilled within the static segment, and requires minimal participation from the MHs. Specifically, it is the set of MSSs comprising a groups’ host-view, that execute a *host-view membership protocol* to track and update the host-view appropriately.

¹ Alternatively, a MH, whose local MSS is not a member of H_G , can send a multicast request via its local MSS to a member of H_G ; however, this requires the MH to be aware of at least one MSS that belongs to H_G .

6.2 Attributes of multicast groups

A multicast group G is associated with the following attributes:

Members E_G – The set of MHs that constitute the multicast group G .

Host view H_G – The set of MSSs such that at least one MH from E_G is local to every MSS in this set.

G is thus defined by the tuple $\{E_G, H_G\}$. Host mobility introduces H_G as an additional attribute of G ; H_G may change due to mobility of individual members although E_G remains invariant. Contrast this with a multicast group of static hosts, where H_G changes if and only if E_G changes.

H_G changes in one of two ways:

I. *Host Mobility:* Let h be a MH in E_G currently located in the cell under MSS M .

- If no other MH in E_G is local to M , then h 's departure from M 's cell will cause M to be deleted from H_G .
- When h enters a cell under MSS M' , such that no other MH in E_G is local to the cell, then M' is added to H_G .

Above, H_G changes in spite of the fact that a MH does not join or leave E_G .

II. *Change in group membership:* Let MH h be a member of E_G local to M 's cell. Also, let h' be a MH that is currently not a member of E_G and is located in the cell served by M' .

- If h leaves the multicast group G and no other MH in E_G is local to M , then M is deleted from H_G .
- If h' joins the multicast group G , and no other MH in E_G is local to M' , then M' is added to H_G .

In this thesis, we restrict our focus to managing host-view changes caused solely due to *mobility* of group members, i.e. we assume membership of a multicast group does not change during the group's lifetime. Handling group membership

changes is omitted from this discussion to retain our focus on mobility per se, viz. tackling host-view changes due to members changing their locations, and its impact on multicast delivery to the group. However, it should be noted that changes in group membership effect the host-view in an identical manner as mobility of group members, viz. addition or deletion of a MSS from the host-view.

6.2.1 Handling changes to H_G :

Addition and/or deletion of MSSs from H_G affects other MSSs in H_G in two ways:

- (1) Propagating the information that H_G has changed in a consistent manner, and
- (2) Notifying ongoing executions of multicast protocols of these changes.

Our approach to maintain and propagate changes to H_G is as follows:

- Treat H_G as a data item that is replicated at all MSSs constituting H_G .
- Associate a *incarnation number* with every addition or deletion of a MSS to H_G . To serialize concurrent changes to H_G , we use a central coordinator MSS C_G to assign the incarnation numbers.
- Each MSS in H_G should receive the changes to H_G in the same sequence, i.e. in order of the incarnation numbers. For example, if MSS M belongs to H_G^i , then its local copy of H_G should evolve in the sequence $H_G^i, H_G^{i+1}, H_G^{i+2} \dots H_G^k$ such that $M \in H_G^j, i \leq j < k$ and $M \notin H_G^k$ i.e., M will receive all changes to H_G in sequence till it drops out of G 's host-view.
- The responsibility of informing all MSSs of a change in H_G lies with the MSS initiating the change.
 - When a MSS M leaves H_G , it first obtains a incarnation number from C_G and then notifies all current members of H_G of its departure.
 - A MSS M is added to H_G as follows. Let a MH in G move from the cell under MSS M' to M . During handoff, M' is notified by M that it does not belong to H_G . M' will then obtain a incarnation number from C_G , and inform all current members of H_G to include M in H_G .

In both cases, the incarnation number received from C_G , is sent along with the notification message.

G can now be defined by the tuple $\{E_G, H_G, C_G\}$. We assume that, in addition to E_G , C_G is also fixed for the duration of G 's lifetime. Therefore, G changes only due to changes in H_G , which progresses as a sequence $H_G^0, H_G^1, H_G^2, \dots$ and so on.

Besides propagating changes to H_G in a consistent manner, we need to also consider the impact of these changes on delivering multicast messages to MHs in G . The exactly-once protocol is affected in two ways:

- When a new multicast is initiated, a copy of the message is sent to all MSSs in the current incarnation of H_G at the initiator.
- If a MSS M joins H_G in the $k+1^{\text{th}}$ incarnation, then each MSS in H_G^k will transfer a copy of their respective *pending* multicast messages to M , i.e. messages initiated by the MSS that have not yet been delivered to all MHs in G . The motivation is that when a MSS joins H_G^{k+1} , the same set of undelivered multicast messages should be present in its local buffer as any other MSS in H_G^{k+1} and so, if a multicast message has not been received by a MH (that belongs to G), then the MH should be able to receive it from any of the MSSs in H_G^{k+1} .

In section 6.3, we present a host-view membership protocol (HVMP) for managing changes to H_G , while section 6.4 adapts the exactly-once multicast scheme described in chapter 5 to deliver multicast messages to a specified group G .

6.2.2 Data structures

At a MSS M , the HVMP and the exactly-once multicast scheme share the following data structures.

- M may belong to host-views associated with different multicast groups. The list of all such groups is kept as M_groups . During the time interval when M is in the process of joining H_G , the entry G in M_groups is associated with a *wait_join* flag; this flag is subsequently removed.
- For each G in M_groups
 - $M_state(G, k) \equiv \{H_G^k, E_G, C_G\}$
i.e., $M_state(G, k)$ encapsulates the information required by M to function

as a member of G 's host-view. H_G^k is the k^{th} incarnation of H_G , and represents the most recent incarnation available at M .

- the list M_Local_G contains those MHs in E_G that are local to M . A MH in this list is marked as *wait_handoff* if the handoff procedure (with respect to G) is currently in progress in response to a *greeting()* message from the MH.
- For each h in M_Local_G , M maintains an array $h_RECD_G^k[]$; the array stores the sequence numbers of the last message received by h from each MSS in H_G^k .
- Incoming multicast messages, addressed to G , are appended to the end of M_Buffer_G , while multicast messages to G , initiated by M are additionally also kept in a list M_mcast_G . Since a multicast message m is addressed to all members of G , it is not necessary to piggyback a list of recipients with m . However, to collect as many acks from local recipients as possible, before they are forwarded to m 's initiator, two lists *ack_list(m)* and *dests(m)* are also inserted in M_Buffer_G along with m .
- M_Local is a list of all MHs local to M , regardless of the multicast groups to which they belong. Each local MH h is associated with a sequence number *seq_{h_to_M}* which keeps track of the last message received in sequence on the fifo channel from h to M . Note that the fifo channels to and from the MH is not specific to any multicast group, and is shared by all the multicast groups to which h belongs.
- Messages scheduled for delivery to a local MH are placed in $M_Transmit$. For every group G in which a local MH h is a member, $M_Transmit$ may contain at most one entry $\langle m, G, h \rangle$, i.e. at most one message m per group G is awaiting delivery to h at any given time.
- Each MH stores *h_groups*, a list of multicast group-ids to which it belongs. On a move, this is sent to the new MSS along with the *greeting()* message.

Size of the array $h_RECD_G[]$ changes as the set of MSSs comprising H_G changes. Therefore, to provide a correct indexing mechanism for $h_RECD_G[]$, H_G is kept as a sorted list using MSS-ids as keys: the i^{th} element corresponds to the i^{th} MSS in H_G . We will use the term “a MSS is added to H_G ” to imply that it is inserted into the sorted list using its id as a key and creating a new element in appropriate position in $h_RECD_G[]$. An analogous action is executed when a MSS is deleted from H_G . Also, to refer to the entry corresponding to a MSS M in $h_RECD_G[]$, we will use the notation $h_RECD_G[M]$ for the sake of readability, although a precise usage would require $h_RECD_G[M_index]$ where M_index is the index of M in the current incarnation of H_G .

6.2.3 Significant and non-significant moves

The host-view membership protocol for H_G is triggered whenever a MH h in G switches its cell. Let h move from the cell under MSS M to that under N . M necessarily belongs to H_G , since h was earlier located in its cell. Depending on whether N belongs to H_G , we distinguish between two types of moves:

- If N does not belong to H_G , then it is a *significant* move, since h 's entry to its cell will modify H_G . Otherwise, it is a *non-significant* move.

The two types of moves are handled differently. For a non-significant move, by virtue of being a part of H_G , N is already in a position to provide the multicast service to any MH in G that may enter its cell. However, for a significant move, N has to first transfer relevant “state” from other members of H_G before it can provide a multicasting service to G . A non-significant move will therefore incur a substantially lower overhead in terms of data transferred within the static network, compared to a significant move.

The data structure that is closely tied with the current incarnation of H_G is $h_RECD_G[]$. In case of a non-significant move, *some* incarnation of H_G is present at both M and N ; let the respective current incarnation numbers be m and n . In this case, M transfers $h_RECD_G^m[]$ to N during handoff. However, since n may be different from m , $h_RECD_G^m[]$ may need to be suitably modified so that it corresponds to the n^{th} incarnation.

For a significant move, both H_G^m and $h_RECD_G^m[]$ needs to be transferred from M to N since N does not presently belong to H_G . More importantly, a significant move requires participation of other MSSs in H_G^m as well:

- M sends an update message to all MSSs in H_G^m to include N in the host-view (resulting in H_G^{m+1}).
- Each MSS is then required to transfer a copy of the pending multicast messages initiated by it, to N . The motivation is that once N has been added to H_G (by incurring the overhead of a significant move), further moves by MHs in G to its cell, can then be handled as a non-significant move.

6.3 Host view membership protocol

For each multicast group in the system, a separate host-view membership protocol (HVMP) is executed to maintain their respective host-views. The protocol is described below with reference to a multicast group G .

6.3.1 Role of the coordinator

The only function of the coordinator C_G is to serialize concurrent changes to G 's host-view so that the local copies of H_G progress in the same sequence. It maintains an *incarnation counter* for G , and when requested by a MSS for a new incarnation number, it returns the current value after incrementing the counter.

6.3.2 Responding to changes in host-view

Let L be a MSS which belongs to G 's host-view, i.e. G is listed in L_groups (the list of all groups whose host-views include L), and let its current incarnation of H_G be l . L is informed of changes to H_G^l through *view_change()* messages sent from other MSSs. A *view_change()* message contains the group-id G , the incarnation number k of the change, and the MSS that is added or deleted from H_G . A *view_change(G , k , <change>)* message is processed as follows:

1. If $l < (k - 1)$, i.e. L has not yet received *view_change()* messages with incarnation numbers in the range $l + 1, \dots, k - 1$, then delivery of the message

is delayed till the current incarnation number of G at N increases to $k - 1$. Since *view_change()* messages are sent from different MSSs, *view_change()* messages with consecutive incarnation numbers may be received out of sequence and hence, delivery of a *view_change()* message may be delayed till others with smaller incarnation numbers have been received.

2. If $\langle change \rangle$ is *add(N)* i.e., the MSS N needs to be added to H_G^{k-1} giving rise to H_G^k , then the following actions are executed:
 - N is inserted in the list H_G^{k-1} , and the current incarnation number of H_G at L becomes k .
 - For each MH h in L_Local_G , an entry for N is inserted in $h_RECD_G^{k-1}[\]$, which is initialized to 0.
 - For each pending multicast message m initiated by L (available in L_mcast_G), a copy is sent to N . Then, L sends a *ack_add(G, L)* message to N ; this informs N that it has been included in L 's copy of H_G^k , and additionally by virtue of *fifo* delivery of messages, ensures that all forwarded messages from L_mcast_G have been received at N .
3. If $\langle change \rangle$ is *delete(N)*, then N is deleted from H_G^{k-1} . For each MH h in L_Local_G , the corresponding entries in $h_RECD_G^{k-1}[\]$ are also deleted.
4. If $\langle change \rangle$ is \perp , then the current incarnation number is simply raised to k (without updating any other data structures).

6.3.3 Handoff

The handoff procedure requires some modifications in the presence of multicast groups. Let a MH h switch cells from M to N :

1. h sends a *greeting(h, M, seq_{M_to_h}, h_groups)* message to N . The additional parameter in the message is *h_groups*; other parameters have been described in section 5.2.2.
2. N sends a *deregister(h, seq_{M_to_h})* message to M . The HVMP module at M will first delete h from M_Local and then notify the WIRELESS module (of

the multicast protocol) of h 's exit from the cell. The WIRELESS module executes the following steps at M :

- For each multicast message m awaiting delivery to h in $M_Transmit$, the WIRELESS module inspects $seq_{M_to_h}$ and any unprocessed acknowledgments from h , to determine if m was delivered to h . If so, the WIRED module is notified of m 's delivery to h (so that it can update the corresponding $h_RECD[]$ array).
- All entries in $M_Transmit$, listing h as a recipient, are then deleted.
- The WIRELESS module hands back $seq_{h_to_M}$, the sequence number of the last message received by M from h , to HVMP.

M replies to N with a *register*(h , $seq_{h_to_M}$) message.

3. On receiving *register*(), the HVMP module at N first inserts h in N_Local and then passes the two sequence numbers, $seq_{M_to_h}$ and $seq_{h_to_M}$, to the WIRELESS module. The WIRELESS module then assigns:

$$seq_{N_to_h} := seq_{M_to_h} + 1$$

$$seq_{h_to_N} := seq_{M_to_h}$$

It then sends the *init_channel*() message to h .

4. As a result of the *register*/*deregister* exchange between M and N , h is deleted from M_Local and inserted in N_local , the channel between M and h is flushed and a new channel set-up between N and h . However, h is still included in M_Local_G for each G in h_groups , and $h_RECD_G[]$ is yet to be transferred from M to N . Therefore, after sending the *deregister* message (and without waiting for the *register* message to arrive), N executes an additional step, consisting of a *transfer*/*handoff* message exchange, for each multicast group G listed in h_groups . If G is associated with a *wait_join* flag in N_groups , i.e. N is in the process of being added to H_G on account of a significant move by some other MH in G , then the *transfer*/*handoff* exchange for h is initiated after N joins H_G (and consequently, h 's move will be treated as a non-significant move). We now describe the steps involved in the *transfer*/*handoff* exchange with respect to G . Note that since messages

are delivered in sequence between MSSs, M should have processed the $deregister(h)$ message from N before receiving any $transfer(h)$ message.

G \in **N_groups** This results in a non-significant move, since N already belongs to G 's host-view. N adds h to N_Local_G , flags the entry as $wait_handoff$ and sends a $transfer(h, G, n)$ message to M , where n is its current incarnation of G . M responds with a $handoff(h, G, h_RECD_G[], m)$ message, with m being its current incarnation number of G . At this point, M deletes h from M_Local_G , i.e., h , as a members of G , is no longer considered local to its cell, and the HANDOFF module notified of h 's departure (with respect to G).

It is possible that M may have not yet received some of the $view_change()$ messages received by N , and therefore m be less than n . In this case, M delays sending the $handoff()$ message till its current incarnation number m increases to n . Thus, it is assured that the incarnation number m sent with $transfer()$ is at least as large as the incarnation number n received with $handoff()$.

When the $handoff()$ message reaches N , let its incarnation number of G be n' ($\geq n$).

1. If $n' = m$, then the flag $wait_handoff$ associated with h is removed from N_Local_G . The $h_RECD_G[]$ array received with the $handoff()$ message is stored at N .
2. If $n' < m$, then delivery of the $handoff()$ message is kept pending till N 's current incarnation number increases to m , and step (1) above is executed. This ensures that the incarnation number associated with $h_RECD_G[]$ is the same as N 's current incarnation number.
3. If $m < n'$, then elements in $h_RECD_G^m[]$ do not correspond to MSSs in $H_G^{n'}$. Therefore, we require that N maintain a log of all $view_change()$ messages it receives during the interval between sending $transfer()$ and receiving $handoff()$, i.e. a log all changes to H_G between H_G^n and $H_G^{n'}$. Note that $n \leq m \leq n'$. N can therefore regenerate H_G^m . Now, for each MSS in H_G^m but not in $H_G^{n'}$, the corresponding element is deleted from $h_RECD_G^m$. Conversely, for each MSS in $H_G^{n'}$ not in H_G^m , an element (initialized to 0) is

inserted in the appropriate place in $h_RECD_G^m$. The result is $h_RECD_G^{n'}$ [], and step (1) is executed. The log of changes to H_G can then be erased.

G \notin N_groups This results in a significant move. G is added to N_groups and the entry is flagged as *wait_join*; N then sends $transfer(h, G, \perp)$ to M which executes as follows:

- h is deleted from M_Local_G , and the HANDOFF module of the multicast protocol notified of h 's departure (relative to G).
- M requests a new incarnation number from C_G ; let the number received be m .
- M waits till its current incarnation number rises to $m - 1$ (by virtue of receiving *view_change()* messages from other MSSs).
- It then sends $handoff(h, G, m-1, h_RECD_G^{m-1}[], M_state(G, m-1))$ to N .
- The addition of N to the host-view is propagated to all MSSs in H_G^{m-1} , by sending a $view_change(G, m, add(N))$ message.

When N receives $h_RECD_G^{m-1}[]$ and H_G^{m-1} with the *handoff()* message, it executes as follows:

- Since H_G^{m-1} does not contain N , it is inserted appropriately in the sorted list giving rise to H_G^m . An entry corresponding to N is also inserted in $h_RECD_G^{m-1}[]$ leading to $\bar{h}RECD_G^m[]$; the entry is initialized to 0.
- The tuple $\{H_G^m, E_G, C_G\}$ is now stored as $N_state(G, m)$, and the *wait_join* flag associated with G is removed from N_groups .
- h is added to N_Local_G .

In response to the $view_change(h, G, add(N))$ from M , each MSS L in H_G^{m-1} will transfer their respective pending multicast messages² to N , followed by a $ack_add(G, L)$ message. The multicast messages are then added to N_Buffer_G . When a $ack_add()$ message has been received from every MSS in H_G^{m-1} , the handoff with respect to G is considered complete.

² These could be received by N prior to receiving the *handoff()* message; if so, their delivery is delayed till the *handoff()* message has been received and processed by N .

For each multicast group G in h_groups , on completion of the handoff process described above, HVMP separately notifies the HANDOFF module. The HANDOFF module at N will then insert a multicast message addressed to G (if any), in $N_Transmit$ for delivery to h (as described later).

6.3.4 Departing from the host-view

A MSS N departs from H_G by first obtaining an incarnation number k from C_G . When its current incarnation number rises to $k - 1$, it sends a $view_change(G, k, delete(N))$ to all MSSs in H_G^{k-1} , deletes all data structures associated with G , and is no longer deemed to belong to H_G . However, till the incarnation number at all MSSs in G 's host-view rises to k , they will continue to send multicast and HVMP messages to N ; these messages are simply discarded by N , once it has sent the $view_change()$ message. Should N rejoin the host-view in the meantime due to a (significant) move by a MH h , N will begin to buffer these messages after sending the $transfer(h, G, \perp)$ message. If N rejoins H_G in the n^{th} incarnation (on reception of $handoff(h, G, n-1, _ , _)$), then N will retain a $view_change()$ message for later delivery only if its associated incarnation number is greater than n . For each MSS in H_G^n , any multicast-related message (i.e., $multicast()$ or $delete()$) received by N prior to receiving the pending multicast messages (and a $ack_add()$ message) from the MSS, is deleted.

N may depart H_G when the following conditions are satisfied:

1. No MH local to N 's cell is a member of G and the handoff process associated with the departure of any such MH has completed. This also implies, that $ack_list(m)$ associated with each message m in N_Buffer_G , is empty i.e., the list of MHs that have acknowledged receipt of m , has been forwarded to m 's initiator³.
2. N_mcast_G is empty i.e., delivery of all multicast messages to G , that were initiated by N , has been completed.

³ When the last MH that belongs to G , leaves the cell, HVMP will notify the HANDOFF module; it is the HANDOFF module that is actually responsible for forwarding all non-empty ack_lists (step GD1 in section 4.4.2).

It is possible that, after requesting C_G for a new incarnation number k and before receiving this number, a MH belonging to G enters N 's cell. In this case, N will rescind its decision to leave H_G by sending a dummy message $view_change(G, k, \perp)$ to all MSSs in H_G^{k-1} .

Although N is allowed to depart H_G when the above two conditions are satisfied, correctness of HVMP is not sacrificed if the departure of N is deferred for a time period T_{depart} . Recall that when a MSS joins H_G , it receives a copy of the pending multicast messages initiated by every other MSS in H_G . Therefore, this overhead is avoided if any MH that belongs to G enters N 's cell during this interval T_{depart} , since N continues to be a part of H_G and the MH's move to N will be treated as a non-significant move. On the flip side, if no MH enters N 's cell during T_{depart} , then N will needlessly be sent a copy of every MCAST and $view_change()$ message. Thus, a low value of T_{depart} could lead to N leaving H_G and then joining it again, while a high value of T_{depart} could lead to useless propagation of multicast and $view_change()$ messages. To determine a balanced estimate of T_{depart} from experimental studies, mobility patterns of a multicast group needs to be investigated.

6.3.5 Forming a multicast group

The initial creation of a multicast group is not a part of HVMP per se. It assumes that the initial incarnation of G is available at the MSSs comprising H_G^0 , using a separate protocol; the function of HVMP is to update H_G in a consistent manner. Below, we present some approaches for creating a group G :

1. A MSS M , initiates creation of G , possibly on behalf of a local MH, by sending a $init(G)$ to all MSSs within a specified scope, e.g. a *campus*. A MSS, on receiving this message, queries its local MHs if they wish to join G ; ids of all interested MHs are sent back to M which forms E_G . The set of MSSs that each have a local MH in E_G , forms H_G^0 .
2. A MSS executes the at least-once delivery algorithm described earlier, to multicast a $invite(G)$ message to a set of specified MHs. A recipient MH either sends a *yes* or *no* reply to its local MSS. Instead of $ack_list()$, the local MSS sends back a list of *yes/no* answers to the initiator MSS. All MHs that

replied with *yes* form E_G and the set of MSSs that sent at least one *yes* reply, form H_G^0 .

3. A MH may register its *intent* to join a group G with its local MSS. When the local MSS receives a *init(G)* (similar to (1) above), it responds to the initiator with the list of MHs that have registered their intent to join G .

6.4 Message delivery to a multicast group G

We now describe the protocol to deliver multicast messages to G . As in the exactly-once protocol of section 5.2.5, there are three modules: WIRED, HANDOFF and WIRELESS module. The WIRELESS module plays the same role, viz. maintaining fifo links between a MH and its local MSS, and will not be described again. Most of the responsibility of the handoff process is now borne by HVMP; therefore, the HANDOFF module is considerably simplified.

6.4.1 WIRED module

To multicast m to G , a MSS M initiates the protocol as follows:

GA1. M maintains a local counter for every group G in M_groups . The counter for G is incremented and assigned as m_seq . M then sends *multicast(G, m)* to all MSSs in H_G^k , where k is its current incarnation of G 's host-view.

GA2. m is also added to M_mcast_G , and *dests(m)* is initialized to E_G .

GA3. Recipient MSSs reply with respective *ack_list(m)* messages. M deletes a MH from *dests(m)* if it is included in a *ack_list()* message. When *dests(m)* becomes empty, a *delete(G, m)* message is sent to all MSSs in M 's current incarnation of H_G , and m deleted from M_mcast_G .

On receiving *multicast(G, m)*, a MSS N executes the steps below. Let the current incarnation of H_G at N be n .

GB1. m is appended to N_Buffer_G ; a list *dests(m)*, initialized to E_G along with another empty list, *ack_list(m)* are associated with m .

GB2.**for** each MH h in N_Local_G
if ($h \in N_Local$) **and**
 $N_Transmit$ does not contain an entry $\langle G, m', h \rangle$
/ there is no message m' addressed to G , awaiting delivery to h , and h has not left the cell*/*
then
 if ($h_RECD_G^n[M] < m_seq$)
 then insert $\langle G, m, h \rangle$ in $N_Transmit$
 else delete h from $dests(m)$

GB3. When N receives $delete(G, m)$, m is deleted from N_Buffer_G .

The WIRELESS module delivers messages from $N_Transmit$ to the target MH. When h acks receipt of m to the WIRELESS module, the WIRED module at N is notified and it executes as follows:

GC1. $h_RECD_G^n[M] := m_seq$

GC2. h is added to $ack_list(m)$, and deleted from $dests(m)$.

GC3. $check_ack_list(m, G)$ ⁴

GC4. **if** ($h \in N_Local_G$) **and** ($h \in N_Local$)

then */* h has not left the cell; so deliver the next applicable message in N_Buffer_G to h */*

/ Let the sequence of messages in N_Buffer_G from the front to the rear, be m_1, m_2, \dots, m_g , where g is the number of messages in N_Buffer_G ; let m occupy the k^{th} position in this sequence.*

for $i := k+1$ to g

if ($h_RECD[init_m_i] < m_i_seq$)

⁴ The procedure $check_ack_list(m, G)$ is defined as:
if ($dests(m) \cap N_Local_G = \emptyset$) **and** $ack_list(m) \neq \emptyset$ **then**
 send a copy of $ack_list(m)$ to $init_m$
 $ack_list(m) := \emptyset$

```

then
  — add  $\langle G, m_i, h \rangle$  to Transmit queue;
  — exit for loop
else
  —  $h$  is deleted from  $dests(m_i)$ 
  —  $check\_ack\_list(m_i, G)$ 

```

6.4.2 HANDOFF module

Let h switch cells from M to N . The HANDOFF module of the *multicast* protocol at M is notified by HVMP of h 's departure (relative to G). In response, it only executes the $check_ack_list(G, m)$ procedure for each multicast message m in M_Buffer_G .

At N , the HANDOFF module begins executing, when it is notified that the necessary state transfer has been completed by HVMP with respect to G .

GD1. Let the sequence of messages in N_Buffer_G from the front to the rear, be

```

 $m_1, m_2, \dots, m_g$ .
for  $i := 1$  to  $g$ 
if ( $h\_RECD[init\_m_i] < m_i\_seq$ )
then
  — add  $\langle G, m_i, h \rangle$  to Transmit queue
  — exit for loop
else
  —  $h$  is deleted from  $dests(m_i)$ 
  —  $check\_ack\_list(G, m_i)$ 

```

6.5 Correctness sketch

For each multicast group G in h_groups , let T_G be the period of time

- between h sending the *greeting* message, after switching cells (from say, M to N),

- if a message from N_Buffer_G is added to $N_Transmit$ in step GD1, then, till such a message is received by h ; else till step GD1 is completed without inserting any such message in $N_Transmit$.

If the move is significant with respect to G (i.e., N does not belong to G 's host-view), then T_G can be expected to be greater than that for a non-significant move (for reasons cited earlier). Let T_{move} be the maximum of T_G , for all G in h_groups . Then, as specified by the *mobility assumption*, we require that the static segment of the network have sufficient communication and processing power so that T_{move} is smaller than the period of time for which h is located within N 's cell, i.e. it should be possible to complete all associated processing that is triggered by h entry to the cell, before h subsequently leaves the cell. Thus, the maximum frequency of moves that can be supported by HVMP and exactly-once multicast protocol is $(1 / T_{move})$.

It can be inferred from the following observations that HVMP correctly updates H_G :

1. Any addition/deletion of MSSs to H_G is serialized by C_G , and each change is associated with a incarnation number. Thus, local copies H_G evolve in the same sequence at all MSSs in G 's host-view.
2. A MSS N leaves G 's host-view only if (a) there are no MHs local to its cell, that belong to G , and (b) all multicast messages to G , that were initiated from the MSS, have been delivered. Also, the HANDOFF module ensures that all non-empty $ack_list(m)$ lists, for any message m in $Buffer_G$, is forwarded to m 's initiator, i.e. the exactly-once delivery protocol will not block due to a N 's departure from the host-view. Thus, N 's departure from H_G does not affect the delivery of messages to any MH in G .
3. When a MSS M joins H_G , say in the $k+1^{th}$ incarnation, each MSS N in H_G^k transfers a copy of their respective pending multicast messages (from N_mcast_G) to M . Thereafter, M will receive a copy of every multicast message that is initiated by a MSS in the $k+1^{th}$ and later incarnations of H_G .
4. When a MH h makes a non-significant move, the incarnation numbers of H_G at the two MSSs may be different. This is handled by appropriately modifying

$h_RECD[]$, as explained in section 4.3.3.1.

Correctness arguments for the exactly-once delivery scheme in the presence of multicast groups, is similar to that presented earlier in section 5.2.6. However, there is one pathological situation that arises solely due to the dynamically changing set of MSSs comprising a host-view. Let h be a MH in G that shuttles back and forth between the cells under two MSSs $L1$ and $L2$. Also, no other MH in G is local to $L1$ or $L2$, i.e. when h switches from $L1$ to $L2$, it results in $L2$'s addition followed by $L1$'s deletion from the host-view, and vice versa. When h enters $L1$'s (or $L2$'s) cell, HVMP will transfer a copy of pending multicast messages from all other MSSs in G 's host-view, and m will be appended to $L1_Buffer_G$. When h leaves $L1$, $L1_Buffer_G$ and thereby m , is deleted. It is then possible that messages ahead of m in $L1_Buffer_G$ are delivered to h but, by the time m is inserted in $L1_Transmit$ for delivery to h , h moves to $L2$'s cell. A similar situation could occur at $L2$ as well. Thus, although h does not violate the mobility assumption, i.e. it receives at least one multicast message in either cell, it may not receive m from either $L1$ or $L2$; delivery of m may be "starved out" due to delivery of other available messages.

Notice that the above situation does not arise in the exactly-once delivery scheme without multicast groups, since *Buffer* is never deleted and m is deleted only after delivery to all recipients. So, after m is appended to *Buffer* at all MSSs, only those messages ahead of m in *Buffer* at the local MSS can be delivered to h before m . The number of such messages are bounded since incoming multicast messages are always added to the end of *Buffer*, and therefore, regardless of h 's mobility, m will eventually become the first deliverable message to h at any MSS. From the mobility assumption, h will then receive m .

In the presence of multicast groups, this problem can be solved by using the following policy :

1. Every multicast message now carries the incarnation number of its initiator, in which the message was first initiated.
2. After a significant move, when a MSS N is added to H_G , the pending multicast messages that are transferred to it by HVMP, are inserted in

N_Buffer_G in increasing order of their incarnation numbers. (Thereafter, incoming multicast messages are added in order of their arrival).

Under this policy, whenever h makes a significant move, it will receive a message with a incarnation number less than or equal to m . The number of such messages is bounded. Alternatively, if it makes a non-significant move, it will receive a message ahead of m in $Buffer_G$ at the local MSS (possibly with a incarnation number higher than m); the number of such messages is also bounded. Thus, m will eventually be delivered to h .

6.6 Future Extensions

In this chapter, “location” of a MH has been identified with its current “cell”, e.g. HVMP maintained a group’s host-view as a set of cells where at least one member of the group was located. However, both for the exactly-once protocol and HVMP, location of a MH could be maintained at a coarser granularity, e.g. a *set* of adjoining cells and the entire host-view maintained in a hierarchical manner. In addition to reducing location updates, it could also reduce the transmission overhead on a initiator (to send a copy of a multicast message and its corresponding *delete()* message to all MSSs in the host-view); with a hierarchical scheme, the transmission overhead is distributed amongst multiple MSSs at the expense of a higher latency for each message to traverse the multiple levels of the hierarchy. Alternatively, instead of using point-to-point message delivery between MSSs, a (best-effort delivery) multicasting service provided at the network layer [15, 28, 56] could be used to efficiently *route* copies of multicast, *view_change()* or *delete()* messages amongst the MSSs comprising a host-view, with additional transport-layer mechanisms to ensure reliable and sequenced delivery.

The scope of this work was restricted to handling host-view changes due to *mobility* of group members. However, addition and/or deletion of members (MHs) could also result in a change of host-view. To incorporate these changes into HVMP would require propagating changes to E_G to all MSSs in H_G as well; otherwise, a multicast message may be buffered indefinitely awaiting delivery to a MH that no

longer belongs to G . Further, we used a central coordinator C_G to serialize changes to H_G ; a distributed solution to the problem is also feasible, e.g. a token circulating amongst the MSSs in H_G to disseminate changes in H_G as well as E_G .

Another area of study is buffer management at the MSSs. Here we assumed that a MSS had sufficient memory for buffering all multicast messages for each of the host-groups to which it belongs. In practice, a MSS should be allowed to drop messages from their local buffers (even prior to receiving the corresponding *delete* notifications), and if necessary, fetch them later from their initiators. Thus, only the initiator of a message is required to buffer the message till it is delivered to all mobile destinations. A scheme to tackle this issue would also be useful in reducing the number of pending multicast messages that are transferred to a MSS, when it joins the host-view of a multicast group: instead of each MSS in the host-view transferring a copy of their pending multicast messages, these messages could be fetched on demand.

Lastly, this work could also be extended to handle networks where the *mobility assumption* is not satisfied. For example, the assumption could be relaxed to allow a MH to make a *bounded* number of moves without receiving any multicast message (although they may be available for delivery).

Chapter 7

Conclusions

The design of algorithms for distributed systems and their communication costs have been based on the assumptions that the location of hosts in the network do not change and the connectivity amongst the hosts is static in the absence of failures. However, with the emergence of mobile computing, these assumptions are no longer valid. Additionally, mobile hosts have tight constraints on power consumption and bandwidth of the wireless links connecting MHs to their local MSSs is limited. This dissertation first presented a system model for the mobile computing environment that (1) captured the effects of varying location of mobile hosts, and (2) differentiated between messages sent on the wired and wireless links. Efficiency of an algorithm in this model is characterized by the search cost incurred and the number of wireless messages exchanged by the algorithm.

The *two tier* principle defined our approach to designing efficient distributed algorithms in this model, viz. localize the communication and data structures necessary for an algorithm within the static portion of the network to the extent possible; the core objective of the algorithm is achieved through a distributed execution amongst the fixed hosts while performing only those operations at the mobile hosts that are necessary for the desired overall functionality. Power consumption at the mobile hosts is thus kept to a minimum, and since updates to the data structures are performed at the fixed hosts, the overall search cost is reduced. The primary contribution of the *two tier* principle is thus to isolate the effects of mobility, bandwidth and power constraints to the MH-MSS interaction (leaving the interaction between MSSs unaffected).

The efficacy of this principle was demonstrated by restructuring two classic algorithms for distributed mutual exclusion, viz. Lamport's and Le Lann's algorithms. Direct application of the original algorithms to the mobile environment was shown to be inefficient in terms of search cost, power consumption at the MHs, usage of

wireless links and handling disconnection of MHs. These drawbacks were removed in algorithm L-MSS, the result of applying the two-tier principle to Lamport's algorithm.

For structuring a logical ring with a circulating token, the two tier principle by itself was not sufficient to handle the effects of varying location of MHs. This required location management of migrant MHs and we presented three strategies: (1) search (2) inform (3) search within a local area with location updates after wide-area moves. The relative merits of the three strategies were quantitatively compared. It was then shown that mobility of a host could determine how often it was able to access the token per traversal of the ring. Since algorithm R-MH allowed each MH to access the token at most once per traversal, we needed an equivalent functionality when the logical ring was shifted to the static segment. A scheme was presented to this end which was equally applicable to all three location management strategies.

We then considered an alternative approach to handling the effects of varying location of MHs namely, replicating the queue of pending requests at all MSSs. This eliminated the need for location management strategies for migrant MHs, but increased the number of messages exchanged within the fixed network to globally order pending requests amongst all MSSs.

The next topic of this dissertation was to develop a checkpointing algorithm for distributed applications running on mobile hosts. Mobile hosts pose three main problems for a checkpointing mechanism: (1) search costs (2) frequent disconnections, and (3) lack of stable storage at a MH. We presented a checkpointing scheme for mobile hosts that uses the stable storage of the MSSs to maintain local checkpoints of MHs. In this scheme, each MH checkpoints local state whenever it moves to a new cell, prior to disconnecting from the network, and additionally, when required by the *two phase* rule. Each application message is piggybacked with control information in the form of CKPT and LOC arrays: given a local checkpoint of some MH as a starting point, the CKPT array enables the checkpointing algorithm to first select a set of local checkpoints to form a consistent global checkpoint, while

the LOC array provides the location of these local checkpoints.

The *two phase* rule is a new result in the field of checkpointing algorithms that is equally valid for static hosts as well. It permits each participant to decide *locally* when it should checkpoint its local state, without requiring any coordination with other participants. This is especially useful for mobile hosts, since coordination between MHs incur search cost and may not be achievable when MHs disconnect.

The final portion of the dissertation presented a framework for structuring protocols for delivering multicast messages to destinations, regardless of their mobility. We first identified the issues introduced specifically by the fact that recipients of these messages were *mobile* hosts, and applied the two-tier principle for developing multicast protocols that provided delivery semantics to counteract the effects of mobility. It needs to be pointed out that a possible network-layer multicast protocol will only guarantee a best-effort delivery; while this is usually acceptable for fixed hosts, it may not suffice for mobile hosts since movement between cells may introduce additional packet losses.

To the extent possible, communication and computation requirements of a protocol execution was borne by the MSSs. Instead of a MH being solely responsible for executing a multicast protocol, the responsibility was shifted to the local MSS to execute the protocol on behalf of the MH. To ensure location-independent delivery, the initiating MSS sent a copy of the multicast message to all MSSs in the system: this ensured that a mobile recipient is able to receive a copy from *at least one* location. However, this would mean that a MH could receive multiple copies of the same message from different cells, given that a message is buffered at all MSSs till its delivery to all mobile destinations. Since message reception consumes power at a MH, such redundant transmissions to a MH are avoided by keeping an array of sequence numbers at the local MSS on behalf of a MH. By inspecting the array, the local MSS is able to decide whether a buffered multicast message should be transmitted to a MH or not, thereby ensuring delivery from *at most one* location. The two schemes are then combined together leading to a protocol for message delivery from *exactly one* location.

Next, we introduced *multicast groups* of MHs, and associated a *host view* with each group. The host view for a multicast group represents the aggregate location information of the group; each member of a multicast group is located within the cell of some MSS in the host-view. Therefore, to deliver a multicast message to a group of MHs, it suffices to send a copy to only those MSSs in the groups' host-view.

The host-view corresponding to a group can change either due to a change in membership of the group, or due to mobility of members. This dissertation has restricted its focus to handling changes to the host-view due to mobility alone (with the assumption that the group membership is an invariant). Since, the host-view changes dynamically due to mobility of group members, we treat host-view as a replicated data item whose consistency is managed by a host-view membership protocol. Further, the host-view membership protocol is almost entirely executed within the static network; this implies that the critical constraints, viz. battery power at the MHs and bandwidth of the wireless connections, are not utilized for maintaining and updating a groups' host-view. Host-view membership information is then utilized to deliver multicast messages exactly-once to a group of mobile hosts.

This dissertation has shown that by using the two-tier approach, it is possible to utilize the superior processing and communication capabilities of static hosts that neither compromises the ability of a MH to move between different locations, nor violates the constraints on power consumption and low-bandwidth of wireless connections. In the context of multicast delivery, this was achieved by maintaining state at the MSS on behalf of a local MH in the form of a RECD[] array, and transferring this state between MSSs during handoff; a MH was thereby only required to participate in a simple request-response interaction with its local MSS. It is our thesis that this idea of partitioning state between a MH and its local MSS, could prove useful for structuring other communication services and applications for mobile hosts, and the specific representation of this state information and how best to partition such information between the MHs and MSSs should be explored for structuring efficient services for mobile hosts.

Bibliography

- [1] Acharya, A., and Badrinath, B. R. An efficient protocol for ordering broadcast messages in distributed systems. In *The 3rd IEEE Symposium on Parallel and Distributed Processing* (1991).
- [2] Acharya, A., and Badrinath, B. R. Delivering multicast messages in networks with mobile hosts. In *Proc. of the 13th Intl. Conf. on Distributed Computing Systems* (May 1993).
- [3] Agrawal, D., and Abadi, A. E. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Transactions on Computer Systems* 9, 1 (Feb 1992).
- [4] Alagar, S., Rajagoplan, R., and Venkatesan, S. Tolerating mobile support station failures. Tech. rep., University of Texas at Dallas, November 1993.
- [5] Alonso, R., and Ganguly, S. Query optimization for energy efficiency in mobile environments. In *Proc. of the 1993 Intl. Workshop on Foundations of Models and Languages for Data and Objects*.
- [6] Alonso, R., and Korth, H. Database system issues in nomadic computing. Tech. Rep. TR-36-92, MITL, December 1992.
- [7] Alonso, R., and Korth, H. Database system issues in nomadic computing. In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data* (June 1993).
- [8] Amir, Y., Moser, L. E., Melliar-Smith, P. M., Agarwal, D. A., and Ciarfella, P. Fast message ordering and membership using a logical token-passing ring. In *Proc. of the 13th Intl. Conf. on Distributed Computing Systems* (May 1993).
- [9] Athas, A., and Duchamp, D. Agent-mediated message passing for constrained environments. In *USENIX Symposium on Mobile and Location-Independent Computing* (Aug. 1993).
- [10] Awerbuch, B., and Peleg, D. Concurrent online tracking of mobile users. In *Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols* (September 1991).

- [11] Badrinath, B. R., Acharya, A., and Imielinski, T. Impact of mobility on distributed computations. *ACM Operating Systems Review* 27, 2 (April '93).
- [12] Badrinath, B. R., Bakre, A., Imielinski, T., and Marantz, R. Handling mobile clients: A case for indirect interaction. In *Fourth Workshop on Workstation Operating Systems (WWOS-IV)* (Oct. 1993).
- [13] Badrinath, B. R., and Imielinski, T. Replication and mobility. In *Proc. of the 2nd workshop on the management of replicated data* (1992), pp. 9–12.
- [14] Badrinath, B. R., Imielinski, T., and Virmani, A. Locating strategies for personal communication networks. *IEEE Globecom 92 Workshop on networking of personal communications applications* (1992).
- [15] Ballardie, T., Francis, P., and Crowcroft, J. Core Based Trees(CBT) an architecture for scalable inter-domain multicast routing. Submitted for publication.
- [16] Barbara, D., and Imielinski, T. Sleepers and workaholics: Caching strategies in mobile environments. In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data* (May 1994).
- [17] Bender et. al., M. Unix for nomads: Making Unix support mobile computing. In *USENIX Symposium on Mobile and Location-Independent Computing* (Aug. 1993).
- [18] Bhagwat, P., and Perkins, C. E. A mobile networking system based on Internet Protocol (IP). In *USENIX Symposium on Mobile and Location-Independent Computing* (Aug. 1993).
- [19] Birman, K., and Joseph, T. Reliable communications in presence of failures. *ACM Trans. Comput. Systems* 5, 1 (1987), 47–76.
- [20] Birman, K., Schiper, A., and Stephenson, P. Lightweight causal and atomic group multicast. *ACM Trans. Comput. Systems* 9, 3 (1991), 272–314.
- [21] Blackwell et. al., T. Secure short-cut routing for Mobile IP. In *USENIX Summer 1994 Technical Conference* (June 1994).
- [22] Caceres, R., and Iftode, L. The effects of mobility on reliable transport protocols. In *Proc. of the 14th Intl. Conf. on Distributed Computing Systems* (May 1994).
- [23] Chandy, K., and Lamport, L. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Systems* 3, 1 (1985), 63–75.
- [24] Chang, J., and Maxemchuk, N. Reliable broadcast protocols. *ACM Transactions*

on *Computer Systems* 2, 3 (August 1984).

[25] Cheriton, D. R. Dissemination-oriented communication systems. Manuscript.

[26] Cohen, D., Postel, J. B., and Rom, R. IP addressing and routing in a local wireless network. Manuscript, July 16, 1991.

[27] Deering, S. E. *Multicast routing in a datagram internetwork*. PhD thesis, Stanford University, 1991.

[28] Deering, S. E., and Cheriton, D. R. Multicast routing in datagram internetworks and extended lans. *ACM Trans. Computers* (May, 1990).

[29] E. W. Dijkstra and W. H. J. Feijen and A. VanGasteren. Derivation of a termination detection algorithm for distributed computation. In *Information Processing Letters* (June 1983).

[30] Forman, G. H., and Zahorjan, J. The challenges of mobile computing. *IEEE Computer* (April '94).

[31] Grant, W. C. Wireless coyote: A computer-supported field trip. *Communications of the ACM* (May 1993).

[32] Heidemann, J. S., Page, T. W., Guy, R. G., and Popek, G. J. Primarily disconnected operation: Experiences with Ficus. In *Proc. of the 2nd workshop on the management of replicated data* (1992), pp. 9–12.

[33] Huang, Y., Sistla, P., and Wolfson, O. Data replication for mobile computers. In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data* (May 1994).

[34] Huston, L. B., and Honeyman, P. Disconnected operation for AFS. In *USENIX symposium on Mobile and Location-Independent Computing* (Aug. 1993).

[35] Imielinski, T., and Badrinath, B. R. Querying in highly mobile distributed environments. In *18th Intl. Conference on Very Large Databases* (1992), pp. 41–52.

[36] Imielinski, T., and Badrinath, B. R. Wireless mobile computing : Challenges in data management. *Communications of the ACM* (Aug. 1994).

[37] Imielinski, T., and Badrinath, B. R. Mobile wireless computing: Solutions and challenges in data management. Tech. rep., Rutgers DCS-TR-296 / WINLAB TR-49, February, 1993.

[38] Imielinski, T., and Badrinath, B. R. Data management for mobile computing.

ACM SIGMOD Record (March '93).

[39] Imielinski, T., Viswanathan, S., and Badrinath, B. R. Power efficient filtering of data on the air. In *EDBT '94* (1994).

[40] Ioannidis, J., Duchamp, D., and Maguire, G. Q. IP-based protocols for mobile internetworking. In *Proc. of ACM SIGCOMM Symposium on Communication, Architectures and Protocols* (September 1991), pp. 235–245.

[41] Jahanian, F., and Jr., W. L. M. Strong, weak and hybrid group membership. In *Proc. of the 2nd workshop on the management of replicated data* (1992).

[42] Johnson, D. B. Scalable and robust internetwork routing for mobile hosts. In *Proc. of the 14th Intl. Conf. on Distributed Computing Systems* (June 1994).

[43] Johnson, D. B., and Zwaenopel, W. Recovery in distributed systems using optimistic message logging and checkpointing. In *7th ACM Symposium on Principles of Distributed Computing* (1988).

[44] Juang, T., and Venkatesan, S. Crash recovery with little overhead. In *11th IEEE Intl. Conf. on Distributed Computing Systems* (1991).

[45] Kaashoek, M. F., and Tanenbaum, A. S. Group communication in the Amoeba distributed operating system. In *Proc. of the 11th Int'l Conf. on Distributed Computing Systems* (1991).

[46] Kistler, J., and Satyanarayanan, M. Disconnected operation in the Coda file system. *ACM Trans. on Computer Systems* 10, 1 (Feb. 1992).

[47] Krishna, P., Vaidya, N. H., and Pradhan, D. K. Recovery in distributed mobile environments. In *IEEE Workshop on Advances in Parallel and Distributed Systems* (October 1993).

[48] Krishna, P., Vaidya, N. H., and Pradhan, D. K. Location management in distributed mobile environments. In *Proc. of the 3rd Intl. Conf. on Parallel and Distributed Information Systems* (October 1994).

[49] Lamport, L. Time, clocks and the ordering of events in a distributed system. *Comm. ACM* 21, 7 (1978), 558–565.

[50] Lann, G. L. Distributed systems, towards a formal approach. *IFIP Congress, Toronto* (1977), 155–160.

[51] Maekawa, M. A \sqrt{N} algorithm for mutual exclusion in decentralized systems.

ACM Transactions on Computer Systems 3, 2 (May 1985).

[52] Marsh, B., Douglass, F., and Caceres, R. Systems issues in mobile computing. Tech. Rep. MITL-TR-50-93, MITL, 1993.

[53] Marsh, B., Douglass, F., and Krishnan, P. Flash memory file caching for mobile computers. Tech. rep., MITL-TR-59-93, June, 1993.

[54] Myles, A., and Skellern, D. Comparison of mobile host protocols for ip. *Journal of Internetworking Research and Experience* 4, 4 (December 1993).

[55] Perkins, C. Providing continuous network access to mobile hosts using TCP/IP. In *Joint European Networking Conference* (May 1993).

[56] Rajagopalan, B. Reliability and scaling issues in multicast communication. In *Proc. ACM SIGCOMM* (1992), pp. 188-197.

[57] Rajagopalan, B., and McKinley, P. K. A token-based protocol for reliable, ordered multicast communication. In *Proc. of the 8th IEEE Symposium on Reliable Distributed Systems* (Oct. 1989).

[58] Spreitzer, M., and Theimer, M. Architectural considerations for scalable, secure, mobile computing with location information. In *Proc. of the 14th Intl. Conf. on Distributed Computing Systems* (June 1994).

[59] Strom, R. E., and Yemini, S. Optimistic recovery in distributed systems. *ACM Trans. on Computer Systems* 3, 3 (1985).

[60] Tait, C. D., and Duchamp, D. Service interface and replica management algorithm for mobile file system clients. In *Proc. First Intl. Conf. on Parallel and Distributed Information Systems* (1991).

[61] Teraoka, F., Yokote, Y., and Tokoro, M. A network architecture providing host migration transparency. *Proc. of ACM SIGCOMM'91* (September, 1991).

[62] Theimer, M., Demers, A., and Welch, B. Operating system issues for PDAs. In *Fourth Workshop on Workstation Operating Systems (WWOS-IV)* (Oct. 1993).

[63] Wada, H., Yozawa, T., Ohnishi, T., and Tanaka, Y. Mobile computing environment based on internet packet forwarding. In *1992 Winter Usenix* (Jan. 1993).

VITA

Arup Acharya

- 1987 B. Tech. (Honours) in Computer Science and Engineering, Indian Institute of Technology (IIT), Kharagapur, India.
- 1987 Chakrabarti, P.P., S. Ghose, Arup Acharya, and S. C. DeSarkar. Tree Searching in Restricted Memory. *Proc. of the Intl. Symposium on Electronic Devices, Circuits and Systems*.
- 1989 Chakrabarti, P.P., S. Ghose, and A. Acharya. Recognition of Occluded Objects with Heuristic Search. **Artificial Intelligence**, Vol. 41, No. 2.
- 1990 Chaudhury, S., A. Acharya, and S. Subramanian. Recognition of Occluded Objects with Heuristic Search. **Pattern Recognition**, Vol. 23, No. 6.
- 1987–90 Teaching Assistant, Department of Computer Science, Rutgers University.
- 1990 M.S. in Computer Science, Rutgers University.
- 1990–91 Research Assistant, Department of Electrical and Computer Engineering, Rutgers University.
- 1991–94 Teaching Assistant, Department of Computer Science, Rutgers University.
- 1991 Arup Acharya, and B.R. Badrinath. An Efficient Protocol for Ordering Broadcast Messages in Distributed Systems. *Proc. of the 3rd IEEE Symposium on Parallel and Distributed Systems*.
- 1992 Arup Acharya, and B.R. Badrinath. Recording Distributed Snapshots based on Causal Order of Message Delivery. **Information Processing Letters**, Vol. 44, No. 6.
- 1993 Badrinath, B.R., Arup Acharya, and T. Imielinski. Impact of Mobility on Distributed Computations. **Operating Systems Review**, Vol. 27, No. 2.
- 1993 Arup Acharya, and B.R. Badrinath. Delivering Multicast Messages in Networks with Mobile Hosts. *Proc. of the 13th Intl. Conf. on Distributed Computing Systems*.
- 1994 Badrinath, B.R., Arup Acharya, and T. Imielinski. Structuring Distributed Algorithms for Mobile Hosts. *Proc. of the 14th Intl. Conf. on Distributed Computing Systems*.

- 1994 Arup Acharya, and B. R. Badrinath. Checkpointing Distributed Applications on Mobile Computers. *Proc. of the 3rd Intl. Conf. on Parallel and Distributed Information Systems*.
- 1995 Ph.D. in Computer Science, Rutgers University.