PERFORMANCE ISSUES IN MOBILE WIRELESS NETWORKS

A Dissertation

by

P. KRISHNA

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

DOCTOR OF PHILOSOPHY

August 1996

Major Subject: Computer Science

PERFORMANCE ISSUES IN MOBILE WIRELESS NETWORKS

A Dissertation

by

P. KRISHNA

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

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

August 1996

Major Subject: Computer Science

ABSTRACT

Performance Issues in Mobile

Wireless Networks. (August 1996)

P. Krishna, B.S.(Hons.), Regional Engineering College, Rourkela, India;

M.S., Texas A&M University

Co–Chairs of Advisory Committee: Dr. Dhiraj K. Pradhan
Dr. Nitin H. Vaidya

The research presented in this dissertation deals with the following performance issues in mobile wireless networks: recovery, location management and routing.

The mobile wireless environment poses challenging fault-tolerant data management problems due to the mobility of the users, limited bandwidth on the wireless link, and power restrictions on the mobile hosts. Thus, traditional fault-tolerance schemes cannot be directly applied to these systems. To this effect, extensions to existing traditional recovery schemes are presented which suit this environment. Analytical models are built to analyze the performance of these schemes to determine those environments where a particular recovery scheme is best suited. The trade-off parameters to evaluate the recovery scheme are identified. It is determined that in addition to the failure rate of the host, the performance of a recovery scheme depended on the mobility of the hosts and the wireless bandwidth.

In order to communicate with a user, one needs to know their location. The network thus faces a problem of continuously keeping track of the location of every user. An important issue in mobile wireless networks is the design and analysis of location management schemes. This dissertation presents the design and analysis of centralized and distributed location management schemes. Significant performance improvements are obtained over existing protocols.

Dynamic mobile wireless networks consist of mobile hosts which can communicate with each other over the wireless links (direct or indirect) without any static network interaction. In such networks the mobile host has the capability to communicate directly with another mobile host in its vicinity. The mobile hosts also have the capability to forward (relay) packets. The problem in hand is the complexity of updating the routing information in such a dynamic network. The dynamism in the network is due to host mobility, and disconnections. This dissertation presents a cluster-based methodology for routing in such dynamic networks. Algorithms for cluster creation and maintenance are presented and analyzed. Compared to existing and conventional routing protocols, the proposed cluster-based approach incurs lower overhead during topology updates and also provides quicker reconvergence.

To My family.

ACKNOWLEDGMENTS

Many people have participated in making the work reported in this dissertation possible. First and foremost, I deeply thank my thesis advisors, Dr. Dhiraj K. Pradhan and Dr. Nitin H. Vaidya, for giving me the independence to choose a research area of my own. Special thanks go to Dr. Pradhan for his constant encouragement. This dissertation would not have been possible without the guidance of Dr. Vaidya. His enthusiasm and energy level is orders of magnitude higher than anybody I have worked with.

I also wish to thank all of the members of my committee for their time and patience: Dr. Zhao, Dr. Welch, and Dr. Ross. I also wish to thank my GCR, Dr. Paul DuBowy for his cooperation.

Of course, no journey through graduate school would be complete without the interaction and fun that comes from one's fellow graduate students. In particular, I would like to thank Debendra, Akhilesh, Mitrajit, Barun, Bakshi, Paul, and Anil for their help, encouragement, and stimulating conversation. I would also like to thank Suresh, Priya, Srinath, Savita, Fiji, Sindhu, Ashok, Adrian, Mary and Koushik for making my stay at College Station very enjoyable.

This acknowledgment would not be complete without thanking my family back in India. They have always given me all the support and encouragement that is humanly possible.

Lastly, and most importantly, I would like to thank Kavitha without whose support this dissertation may not have been completed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

Mobile wireless network gives users information access regardless of their location. Users of portable computers carry their laptops with them whenever they move from one place to another and would like to maintain transparent network access through a wireless link. With the availability of wireless interface cards, mobile users are no longer required to remain confined within the static network premises to get network access. A host that can move while retaining its network connection is a *mobile host* (*mh*) [31].

Mobility is **not** the same as wirelessness. Mobile host is one who has ability to communicate anytime anywhere. On the other hand, a wireless host is one which is physically untethered by a communication link, which is a capability of the physical media in use. Clearly, wirelessness enables greater mobility than is possible with wired communications. However, wide area network (WAN) mobility does not always require wirelessness. It is easy to conceive ubiquitous internet ports which would support mobility but not require wireless access. One could take a portable computer from place to place, connecting via network taps to send and receive data; this would comprise a mobile capability without involving any wireless technology.

Conversely, a wireless capability in a host does not necessarily imply unlimited mobility. There are a number of wireless local area networks (LANs) [20, 31] which, although free from the physical constraints of cables, cannot be considered to be WANs because of their limited range of operation.

The journal model is *IEEE Transactions on Computers.*

A. Classification of Mobile Wireless Networks

Mobile wireless networks can be classified broadly into two types : *Infrastructure* networks and *Dynamic* networks.

1. Infrastructure Networks

Infrastructure networks are two tiered networks composed of a static backbone network and peripheral wireless networks [9, 51, 58]. The static network comprises of fixed hosts and communication links between them. Some of the fixed hosts, called *mobile support stations* (*MSS*)[1] are augmented with a wireless interface, and, they provide a gateway for communication between the wireless network and the static network. Due to the limited range of wireless transceivers, a mobile host can communicate with a *mobile support station* only within a limited geographical region around it. This region is referred to as a mobile support station's *cell.* A mobile host communicates with one *MSS* at any given time. *MSS* is responsible for forwarding data between the mobile host and static network. Communication to and from the mobile host takes place via the static network. An example of infrastructure networks is the new Personal Communication Systems (PCS) [20, 31].

2. Dynamic Networks

Dynamic networks consist of mobile hosts which can communicate with each other over the wireless links (direct or indirect) without any static network interaction. In such networks the mobile host has the capability to communicate directly with another mobile host in its vicinity. The mobile hosts also have the capability to forward (relay) packets. Examples of such networks are *ad-hoc* wireless local area

[1]Mobile support stations are also called *base stations.*

networks [17, 39, 47, 61] and packet radio networks [16, 40, 42, 59]. The term *ad-hoc* network is in conformance with current usage within the IEEE 802.11 subcommittee [17].

Example applications of such networks range from conference rooms to battlefields. To communicate with each other, each mobile user needs to connect to a static network (wide area network, satellite network). However, there might be situations where connecting each mobile user to a static network may not be possible due to lack of facilities, or it may be expensive. In such situations, it would be preferable for the mobile users to set up communication links between themselves without any static network interaction [39].

B. Limitations and Challenges of Mobile Wireless Networks

The technical challenges that mobile computing must surmount to achieve its potential are far from trivial. Some of the challenges in designing protocols for mobile computing systems are quite different from those involved in the design of protocols for today's "immobile" networked systems. In the following we list the limitations and challenges.

- **Wireless Bandwidth**: The bandwidth on the wireless medium is much lower than the wired medium. Cutting-edge products for portable wireless communications achieve only 2 Mbps for local area networks [20]. The bandwidth availability in wide area wireless networks are much lower; in the order of tens of Kbps (e.g., 19.2 Kbps for Cellular Digital Packet Data (CDPD) networks [15]). Limitations on network bandwidth affect the performance of distributed protocols and applications that require bulk data transfer over the wireless link.

- **Unreliable Wireless Link**: While the wired links on the static network offer a virtually error free transmission medium (*Bit Error Rates* (BER) of the order of 10^{-8} to 10^{-12}), wireless links are much more unreliable. BER in wireless links is of the order of 10^{-2} to 10^{-6}, and they are highly sensitive to the direction of propagation, multipath fading, and other interference [9, 51]. Network protocols such as TCP, ATM do not work well in wireless networks because they were tailored for environments which offer a relatively stable and error free transmission medium, unlike the much more hostile and error prone wireless medium [6, 7, 8, 14].

- **Mobility of Hosts**: Wireless connection enables virtually unrestricted mobility and connectivity from any location within the area of radio coverage. Mobility is an important new component that will have far-reaching consequences for system design. An important issue that stems out of mobility of hosts is design and analysis of location management schemes in *infrastructure* networks [36, 37, 49, 50, 57, 58]. In *dynamic* networks, mobility of hosts causes the network topology to change. This in turn complicates the design of routing protocols in such networks [16, 39, 47, 59, 61]. Mobility also affects the design of applications. In a client-server environment, mobile clients may find themselves far away from their servers; servers may also move further away from their clients. Thus, the system will have to adapt to changing spatial distribution of clients by dynamic replication of data and services (for example [31]).

- **Available Storage on Mobile Computers**: The storage space available on a mobile computer is limited by physical size and power requirements. Traditionally, disks provide large amounts of stable (non-volatile) storage. However, in a mobile computer, disks are a liability. This is because, they consume more power than memory chips. Moreover, they may not really be nonvolatile when subject to the harsh environment that a portable computer faces [3]. This will require any critical data (e.g., database logs, process checkpoint) to be stored elsewhere other than the mobile computer. This in turn will impact performance of protocols (e.g., recovery protocols) that require these critical data [1, 48, 65].

- **Disconnections** : Mobile units run on batteries; with limited capacity [20, 31]. Limitations in battery power and bandwidth make disconnections from the network very frequent. Disconnections have various degrees depending on bandwidth availability. Because of their frequency, disconnections must be treated differently than failures or crashes. The difference between disconnection and failure is its *elective* nature. Disconnections are to be treated as *planned* failures which can be anticipated and prepared for [20, 31, 68]. Disconnections cause the network topology to change in dynamic networks. This is because the disconnected mobile host can no longer be used for forwarding data.

These limitations and challenges have significant effect on the design and performance of distributed algorithms, application recovery protocols, location management protocols, routing protocols, file systems, database systems, and transport protocols, for mobile wireless networks.

C. Overview of the Thesis

As our discussion in the previous section indicates, mobile computing is a large and rapidly expanding area with number of problems yet to be solved. Since it is impossible to study the entire area in any depth in a single dissertation, we limit our scope to only some of the issues. The research presented in this thesis deals with the following issues:

- Application level recovery in infrastructure networks
- Location management in infrastructure networks
- Routing in dynamic networks

1. Recovery Issues

The mobile computing environment poses challenging fault-tolerant data management problems due to the mobility of the users, limited bandwidth on the wireless link, and power restrictions on the mobile hosts. Thus, traditional fault-tolerance schemes cannot be directly applied to these systems. In this work we investigate the limitations of the mobile wireless environment, and its impact on recovery protocols. To this effect, extensions to existing traditional recovery schemes are presented which suit this environment. We build analytical models to analyze the performance of these schemes to determine those environments where a particular recovery scheme is best suited. The trade-off parameters to evaluate the recovery scheme are identified. It is determined that in addition to the failure rate of the host, the performance of a recovery scheme depended on the mobility of the hosts and the wireless bandwidth.

2. Location Management

An important issue in personal communication networks is the design and analysis of location management schemes. We classify the location management schemes into *centralized* and *distributed* schemes.

a. Centralized Location Management

Centralized location management schemes assume that each host has a *home* server named the *home location server* (*HLS*) which maintains the current location of the host. There are existing standards for carrying out location management in a *centralized* manner, e.g., EIA/TIA Interim Standard 41 (IS-41) [58]. However, these schemes are not efficient, due to increase in network load and location management costs. To overcome these drawbacks, we propose forwarding techniques that augment the IS-41 scheme to provide efficient location management. Although, forwarding reduces network load during updates, they may increase search cost due to long chain lengths. We propose heuristics to limit the number of forwarding pointers traversed during a search. We build analytical models to compare the performance of the proposed approach with the IS-41 scheme. We also present a strategy to perform *search-updates*. A search-update occurs after a successful search, when the location information corresponding to the searched mobile host is updated at some hosts. Analysis shows that for some network parameter values, performing search-updates significantly reduces the search costs and total network load.

b. Distributed Location Management

Studies have indicated that in centralized location management schemes such as IS-41, the bottleneck is the HLS [57, 58]. For a typical PCS environment, the HLS is expected to experience a high update rate, and a very high search (or call-delivery) rate [58]. It is thus evident that new network architectures need to be investigated for PCS. In this work we use a network architecture that consists of a hierarchy of location servers so that there is no single bottleneck in the network. We propose *static* and *adaptive* location management schemes for this network architecture. A suite of schemes is proposed, however, it is observed that no scheme outperforms others for all call-mobility patterns. Thus, in order to obtain good performance using static location management, the system designer should *a priori* have a fair idea of the call-mobility pattern of the users. However, the user behavior is not always available to the system designer. Thus, there is a need for adaptive location management. We propose an adaptive scheme that dynamically estimates the future user behavior with the help of past call-mobility patterns. Results indicate that the adaptive scheme performs better than the static schemes for a wide range of call-mobility patterns.

3. Routing Protocol

The design and analysis of routing protocols is an important issue in dynamic networks such as packet radio and ad-hoc wireless networks. The conventional routing protocols were not designed for networks where the topological connectivity is subject to frequent, unpredictable changes. Most protocols exhibit their least desirable behavior for highly dynamic interconnection topology. We propose a new methodology for routing and topology information maintenance in dynamic networks. The basic idea behind the protocol is to divide the graph into number of overlapping clusters. A change in the network topology corresponds to a change in cluster membership. We present algorithms for creation of clusters, as well as algorithms to maintain them in the presence of various network events. Compared to existing and conventional routing protocols, the proposed cluster-based approach incurs lower overhead during topology updates and also has quicker reconvergence. The effectiveness of this approach also lies in the fact that existing routing protocols can be directly applied to the network – replacing the nodes by clusters.

D. Thesis Organization

In Chapter II we address the recovery issues in mobile wireless networks. Chapter III presents the centralized location management scheme. Chapter IV presents the distributed location management scheme. The cluster-based approach for routing in dynamic networks is presented in Chapter V. Chapter VI summarizes the results of this thesis and discusses possible extensions.

CHAPTER II

RECOVERY IN MOBILE ENVIRONMENT

A. Introduction

This chapter deals with design of protocols to recover from a mobile host failure in an *infrastructure* mobile wireless network. The system model of the infrastructure network is the same as explained in Chapter I.

A mobile host may become unavailable due to (i) failure of the mobile host, (ii) disconnection of the mobile host, and (iii) wireless link failure [20, 31, 48, 65]. Loss of battery power make disconnections from the network frequent, and sometimes unpredictable. Because of their frequency, disconnections must be treated differently than failures. The difference between disconnection and failure is its *elective* nature. User initiated disconnections can be treated as *planned* failures, which can be anticipated and prepared for [20, 31, 68]. The wireless link is equivalent to an intermittently faulty link, which transmits correct messages during fault-free conditions, and stops any transmissions upon a failure. Disconnections and weak wireless links primarily delay the system response, whereas a host failure affects the system state.

Strategies are developed in this paper to recover from various *transient* faults in the mobile host. These are faults that are typically caused by environmental upsets (e.g., wireless link failure, power glitches at the mobile host, electromagnetic interference, radiation, etc.) or software errors (e.g., heisenbugs [38], rarely used code paths, etc.). Transient failures are the most common mode of failure [64]. We assume the well known *fail-silent* [64] model. In this model, upon a failure at a mobile host, the host stops executing, and its state is lost. Failure detection is performed by requiring a mobile host to send periodic “I am alive” messages to the base station.

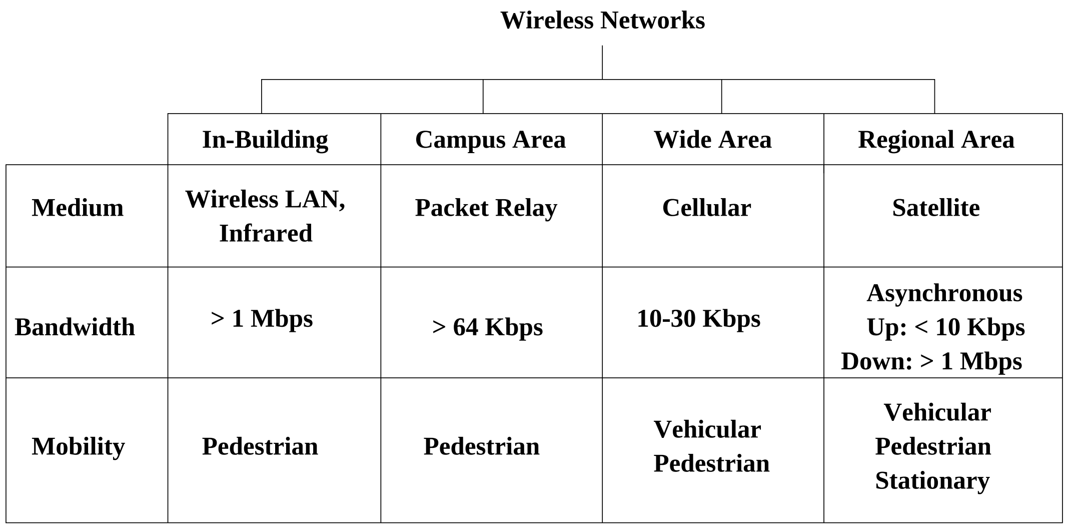
Transient failure recovery requires re-execution from a known good state or a complete restart. Checkpointing and message logging techniques have been proposed earlier for efficient rollback and recovery with minimal loss in performance [38, 64].

Table I. Difference Between Static Wired and Mobile Wireless Networks: Recovery Perspective

Category	**Static Wired Networks**	**Mobile Wireless Networks**
Network char.	Uniform, Non-varying	Non-uniform, Varying
Host's local disk	Stable	Unstable
Stable storage location	Static	Mobile
Key perf. parameter	Failure rate	Failure rate, wireless bandwidth, mobility
Perf. metrics	State-saving cost Recovery cost	State-saving cost, Recovery cost, Handoff time

It will now be discussed why traditional fault-tolerance schemes cannot be applied to a mobile wireless environment [48, 65]. Some of the differences between static and mobile networks are enumerated in Table I.

Traditional fault-tolerance schemes like checkpointing and message logging [38, 64] require a stable storage for saving the checkpoint and the logs. It has been pointed out [3] that while the disk storage on a static host is stable, the stability of any storage on a mobile host is questionable, for reasons such as dropping of laptops or effect of airport security systems [1]. Thus, a mobile host's disk storage cannot be considered stable and is vulnerable to failures. Moreover, all mobile hosts are not necessarily equipped with disk storage. Thus, we need the stable storage to be located on a static host. An obvious candidate is the ‘local base station’, that is the base station in charge of the cell in which the mobile host is currently residing. Traditional recovery schemes are not applicable because the mobile hosts move from cell to cell. Thus, a mobile host does not have a fixed base station to communicate with. Also, recovery is complicated because successive checkpoints of a mobile host may be stored at different base stations. This dynamic topological situation warrants formulation of special techniques to recover from failures.



Wireless Networks

	In-Building	**Campus Area**	**Wide Area**	**Regional Area**
Medium	**Wireless LAN, Infrared**	**Packet Relay**	**Cellular**	**Satellite**
Bandwidth	**> 1 Mbps**	**> 64 Kbps**	**10-30 Kbps**	**Asynchronous Up: < 10 Kbps Down: > 1 Mbps**
Mobility	**Pedestrian**	**Pedestrian**	**Vehicular Pedestrian**	**Vehicular Pedestrian Stationary**

Fig. 1. Classification of Wireless Networks

Traditional fault-tolerant schemes do not consider the disparity in the network characteristics (bandwidth, error) of the static network and the wireless network. As shown in Fig. 1, the network and user characteristics (bandwidth, mobility) also vary with the type of wireless network used (infrared, packet relay, satellite, etc.). Over a length of a connection, the mobile host might be employing different types of wireless networks. For example, within a building, infrared will be used; in a campus environment, packet relay will be used; and in a remote region, satellite will be used. Available wireless bandwidth and error conditions will be different in each of these wireless networks. Thus, the appropriate recovery protocol needs to be determined adaptively, based on the characteristics of the underlying wireless network and users.

Performance of traditional recovery schemes primarily depends on the failure rate of the host [63, 77]. However, in a mobile environment, due to mobility of the hosts and limited bandwidth on the wireless links, parameters other than failure rate of the mobile host play a key role in determining the effectiveness of a recovery scheme. A mobile environment is determined by the mobility, wireless bandwidth and the failure rate. This chapter presents the following:

- User transparent recovery from mobile host failure.
- Trade-offs for the recovery schemes proposed.
- Best recovery scheme for an environment.

We propose several schemes for recovery from a failure of a mobile host. These proposed schemes have two major components: a *state-saving* scheme and a *handoff* scheme. We propose two schemes for state-saving, namely, (i) *No Logging* (N) and (ii) *Logging* (L), and three schemes for handoff, namely, (i) *Pessimistic* (P), (ii) *Lazy* (L), and (iii) *Trickle* (T). We denote a recovery scheme that employs a combination of a state-saving scheme, X ($X \in \{N, L\}$), and a handoff scheme, Y ($Y \in \{P, L, T\}$), as XY. For example, LL is a recovery scheme that uses a combination of the Logging scheme for state-saving and the Lazy scheme for handoffs.

Each combination provides some level of availability and requires some amount of resources: network bandwidth, memory, and processing power. Through analysis, we show that there can be no single recovery scheme that performs well for all mobile environments. However, among the recovery schemes considered, we determine the best recovery scheme for each environment, as shown in Fig. 2.

Mobility	Wireless Bandwidth	Failure Rate	Optimal Scheme
High	Low	Low	*LL*
		High	*NT*
	High	All	*LT*
Low	All	All	*LL*

Fig. 2. Best Recovery Scheme

This chapter is organized as follows. Section B overviews related work. Section C presents the recovery strategies. Section D gives the performance analysis of the recovery strategies, and summary is found in Section E.

B. Related Work

Research in mobile computing primarily has focussed on mobility management, database system issues, network protocols, disconnected operation and distributed algorithms for mobile hosts [20, 31]. Work on fault-tolerance issues is very limited.

Alagar et.al. [2], demonstrate schemes to tolerate base station failures by replicating the information stored at a base station, at several "secondary" base stations. Strategies for selecting the secondary base stations were shown. These schemes can easily be integrated with the recovery schemes presented in this chapter, to provide a system that tolerates both base station and mobile host failures.

Rangarajan et.al. [67], present a fault-tolerant protocol for location directory maintenance in mobile networks. The protocol tolerates base station failures and host disconnections. Logical timestamps are used to distinguish between old and new location information. The protocol also tolerates the corruption of these logical timestamps.

Lin [52], studies the recovery of mobility databases at the *visitor location register* (VLR) and *home location register* (HLR) for personal communication networks. Schemes with and without checkpointing are described and analyzed.

Acharya et.al. [1], identify the problems with checkpointing mobile distributed applications, presenting an algorithm for recording global checkpoints for distributed applications running on mobile hosts.

In this research, however, we consider protocols to recover from failure in a mobile host, independent of other hosts in the system. Also, we study the effect of mobility and wirelessness on such recovery protocols.

C. Recovery Strategies

A recovery strategy essentially has two components: a state-saving and a handoff strategy. This Section presents two strategies for saving the state, and three strategies for handoff, to achieve fault-tolerance. Strategies for saving the state are similar to traditional fault-tolerance strategies.

1. State-Saving

State-saving strategies presented in this chapter are based on traditional checkpointing and message-logging techniques. In such strategies, the host periodically saves its state at a stable storage. Thus, upon failure of the host, execution can be restarted from the last-saved checkpoint.

It was indicated earlier [3] that a mobile host's disk storage cannot be considered stable. Thus, our algorithms use the storage available at the base station for the cell in which the mobile host is currently residing, as the stable storage.

Multiple hosts (both static and mobile) will take part in a distributed application. Such applications require messages to be transferred between the hosts, and might also require user inputs at the mobile hosts. While the user inputs may go directly to the mobile host, the messages will first reach the base station in charge of the *cell* in which the mobile host currently resides. The base station then forwards the messages to the corresponding mobile host. Likewise, all messages sent by a mobile host will first be sent to its base station, which will forward them to the destination host (static or mobile).

Two strategies to save the process state [38] will be discussed here: (i) *No Logging* and (ii) *Logging*. It is assumed that the mobile host remains in one cell during the length of the application. This is followed by a discussion of three schemes that address the recovery steps needed because of mobility.

• *No Logging* Approach (denoted as N): The state of the process can get altered, either upon receipt of a message from another host, or upon user input. The messages or inputs that modify the state are called *write* events. (If semantics of the message are not known, in the worst case, we might have to assume that the state gets altered upon receipt of every message or user input). In the *No Logging* approach, the state of the mobile host is saved at the base station upon every *write* event on the mobile host data.

After a failure, when the mobile host restarts, the host sends a message to the base station, which then transfers the latest state to the mobile host. The mobile host then loads the latest state and resumes operation. Importantly, need for frequent transmission of state on the wireless link is a limiting factor for this scheme.

• *Logging* Approach (denoted as L): This approach is rooted in "pessimistic" logging [13], used in static systems. In this scheme, a mobile host checkpoints its state periodically. To facilitate recovery, the *write* events that take place in the interval between checkpoints are also logged. As defined earlier, the messages or inputs that modify the state of the mobile host are called *write* events. If a *write* message is received from another host, the base station first logs it, and then forwards it to the mobile host for execution. Likewise, upon user input (write event), the mobile host first forwards a copy of the user input to the base station, for logging. After logging, the base station sends an acknowledgment back to the mobile host. The mobile host can process the input, while waiting for the acknowledgment, but cannot send a response. Only upon receipt of the acknowledgment does the mobile host send its response.

The above procedure ensures that no messages or user inputs are lost due to a failure of the mobile host. The logging of the write events continues until a new checkpoint is backed up at the base station. The base station then purges the log of the old write events, along with the previous checkpoint.

After a failure, when the mobile host restarts, the host sends a message to the base station, which then transfers both the latest backed-up checkpoint of the host, as well as the log of write events, to the mobile host. The mobile host then loads the latest backed-up checkpoint and restarts executing, by replaying the write events from its logs, thus reaching the state before failure. Below, the recovery steps are considered which are needed, arising due to mobility of the hosts.

2. Handoff

The mobility warrants a special *handoff* process, described below. The key problem to be addressed is how a recovery can be effected if a mobile host moves to a new cell, as illustrated in the following example.

Consider the system in Fig. 3. *BSi* denotes *i*-th base station, and *mhi* denotes *i*-th mobile host. Here, mobile hosts *mh1* and *mh2* are executing a distributed algo-

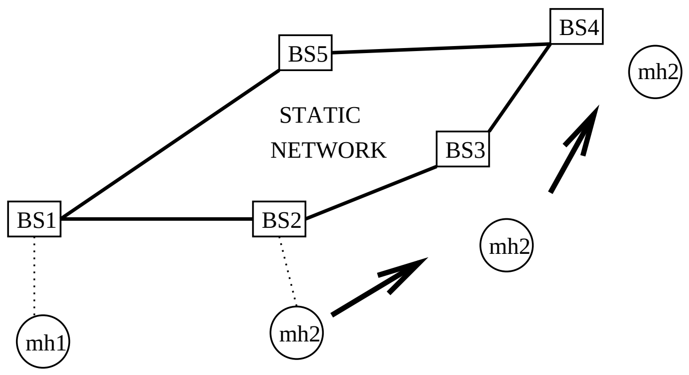


Fig. 3. Handoff in the Middle of an Execution

rithm. The mobile host $mh2$ has saved both its checkpoint and message log at $BS2$. In the middle of the execution, *mh2* moves to the cell of *BS3*, and then to the cell of *BS4*. Handoff occurs at both the boundaries of *BS2* and *BS3*, and *BS3* and *BS4*. Let a failure of the mobile host *mh2* occur upon reaching the cell of *BS4*. Had *mh2* remained in the cell of *BS2*, the system would have recovered because the checkpoint and the logs are saved at *BS2*. But since no state-saving took place at *BS3* or *BS4*, and since *BS4* does not know where the last checkpoint of *mh2* is stored, the recovery procedure will now have to identify the base station where the checkpoint is saved. This will warrant additional steps to identify the base station. Therefore, what is proposed is transferring during the handoff process some *information* regarding the state of the mobile host. The following delineates three ways to transfer this information during the *handoff* process: (i) *Pessimistic*, (ii) *Lazy*, and (iii) *Trickle*.

a. Pessimistic Strategy (P)

When a mobile host moves from one cell to another, the checkpoint is transferred to the new cell's base station during handoff. If Logging strategy is being used, then in addition to the checkpoint, the message log is also transferred to the new cell's base station. Upon receipt of the checkpoint and/or the log, the new cell's base station sends an acknowledgment to the old base station. The old base station, upon receiving the acknowledgment, purges its copy of the checkpoint and the log, since the mobile host is no longer in its cell.

The chief disadvantage to this approach is that it requires a large volume of data to be transferred during each handoff. Potentially, this can cause long disruptions during handoffs. However it can be avoided if we use the *Lazy* or *Trickle* strategy, as explained.

b. Lazy Strategy (*L*)

With *Lazy* strategy, during handoff, there is no transfer of checkpoint and log. Instead, the *Lazy* strategy creates a *linked list* of base stations of the cells visited by the mobile host. The mobile host may be using either one of the state-saving strategies (No Logging or Logging) described earlier. If the mobile host is using the No Logging strategy, the checkpoint is saved at the current cell's base station after every *write event*. On the other hand, if Logging strategy is used, a log of *write* events is maintained, in addition to the last checkpoint of the mobile host at the base station. Upon a handoff, the new cell's base station keeps a record of the preceding cell. Thus, as a mobile host moves from cell to cell, the corresponding base stations effectively form a linked list. One such linked list needs to be maintained at the base station for each mobile host.

This strategy could lead to a problem if the checkpoint and logs of the mobile host are unnecessarily saved at different base stations. To avoid this, upon taking a checkpoint at a base station, a notification is sent to the last cell's base station, to purge the checkpoint and logs of the mobile host, if present. If a checkpoint is not present, this base station forwards the notification to the preceding base station in the linked list. This process continues, until a base station with an old checkpoint of the mobile host is encountered. All base stations receiving the notification purge any state associated with the particular mobile host.

The *Lazy* strategy saves considerable network overhead during handoff, compared to the *Pessimistic* strategy. Recovery, though, is more complicated. Upon a failure, if the base station does not have the process state, it obtains the logs and the checkpoint from the base stations in the linked list. The base station then transfers the checkpoint and the log of write events to the mobile host. The host then loads the checkpoint, and replays the messages from the logs to reach the state just before failure.

c. Trickle Strategy (T)

Importantly, in the *Lazy* strategy, the scattering of logs in different base stations increases as the mobility of the host increases, potentially making recovery time-consuming. Moreover, a failure at any one base station containing the log renders the entire state information useless.

To avoid this, a *Trickle* strategy is proposed. In this strategy, steps are taken to ensure that the logs and the checkpoint are always at a nearby base station (which may not be the current base station). In addition, care is taken so that the handoff time is as low as with *Lazy* strategy.

We make sure that the logs and the checkpoint corresponding to the mobile host are at the "preceding base station" of the current base station[1]. (The preceding base station is the base station of the previous cell visited by the mobile host.) Thus, assuming that neighboring base stations are one hop from each other (on the static network), the checkpoint and the logs are always, at most, one hop from the current base station.

[1]Variations of this scheme are possible where the checkpoint and logs are at a *bounded* distance from current cell.

To achieve the above, during handoff, a control message is sent to the preceding base station to transfer any checkpoint or logs that had been stored for the particular mobile host. Similar to *Lazy* strategy, the current base station also sends a control message to the new cell's base station identifying the preceding cell location of the mobile host. Thus, the new cell's base station, just retains the identification of the mobile host's preceding cell.

If a checkpoint is taken at the current base station, it sends a notification to the preceding base station that has the last checkpoint and logs, to purge the process state of the mobile host. During recovery, if the current base station does not have a checkpoint of the process, it obtains the checkpoint and/or the logs from the preceding base station[2]. The base station then transfers the checkpoint and/or the log to the mobile host. The mobile host then loads the checkpoint and replays the messages from the logs, to reach the state just before failure.

D. Performance Analysis

Basically, six schemes (combinations of state-saving and handoff) are possible. This Section analyzes these schemes, determining which combination is best-suited for a given environment.

1. Terms and Notations

The following terminology is used, the significance of which will be clearer later in this Section.

[2]If No Logging strategy was used for state-saving, the checkpoint will be transferred. On the other hand, if Logging is used, the checkpoint and the log are transferred.

- The term *operation* may refer to one of (i) checkpointing, (ii) logging, (iii) handoff, or (iv) recovery.

- **Cost** of an operation quantifies the network usage of the messages due to the operation. In other words, it is the amount of time the network is busy trasmitting the messages.

- λ: Failure rate of the mobile host. We assume that the time interval between two failures follows an exponential distribution with a mean of $1/\lambda$.

- μ: Handoff rate of the host. We assume that the time interval between two handoffs follows an exponential distribution with a mean of $T = 1/\mu$.

- The time interval between two consecutive write events is assumed to be fixed and equal to $1/\beta$. Write events are comprised of user inputs and messages from other hosts. Since we are only interested in the relative performance of the various schemes proposed, this assumption will not significantly affect the results.

- r: Communication-mobility ratio, defined as the expected number of write events per handoff, equal to β/μ. For a fixed β, a small value of r implies high mobility, and vice-versa.

- ρ: Fraction of write events that are user inputs. If ρ is 1, then all the write events are user inputs. This means that the application is not distributed in nature, and that the mobile host is the only participant in this execution.

- T_c: Checkpoint interval, defined as the time spent between two consecutive checkpoints executing the application. T_c is fixed for all schemes under consideration. Specifically, T_c is $1/\beta$ for No Logging schemes.

- k: Number of write events per checkpoint. For the Logging schemes, $k = \beta T_c$. For the No Logging schemes, k is always equal to 1.

- α: Wireless network factor. This is the ratio of the cost of transferring a message over one hop of a wireless network to the cost of transferring the message over one hop of a wired network. The higher the value of α, the costlier is the wireless transmission relative to the wired transmission.

- $N_c(t)$: Number of checkpoints in t time units.

- $N_l(t)$: Number of messages logged in t time units.

- C_c: Average cost of transferring a checkpoint state over one hop of the wired network.

- C_l: Average cost of transferring an application message over one hop of the wired network.

- γ: Relative logging cost. It is the ratio of the cost of transferring an application message to the cost of transferring a checkpoint state over one hop of the wired network (C_l/C_c).

- C_m: Average cost of transferring a control message over one hop of the wired network. The size of a control message is typically assumed to be much less than the size of an application message.

- ϵ: C_m/C_c = Relative control message cost. It is the ratio of the cost of transferring a control message to the cost of transferring a checkpoint state over one hop of the wired network.

- C_h: Average cost of a handoff operation.

- C_r: Average cost of a recovery operation.
- C_t: Average total cost per handoff.

2. Modeling and Metrics

The interval between two handoffs is referred to as *handoff interval.* A handoff interval can be represented using a 3-state discrete Markov chain [76, 77], as presented in Fig. 4.

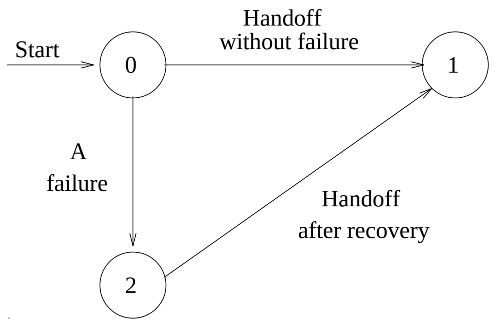


Fig. 4. Markov Chain Representation

State 0 is the initial state when the handoff interval begins. During the handoff interval, the host receives messages and/or user inputs (write events). Depending upon the state-saving scheme, the host either takes a checkpoint or logs the write events. A transition from State 0 to State 1 occurs if the handoff interval is completed without a failure. If a failure occurs during the handoff interval, a transition is made from State 0 to State 2. After State 2 is entered, a transition occurs to State 1 once the handoff interval is completed. To simplify the analysis, we have assumed that, at most, one failure occurs during a handoff interval. This assumption does not significantly affect the results when the average handoff interval is small, compared to the mean time to failure.

Fig. 5 illustrates an example of the state transitions in a handoff interval. Fig.

5(a) is a case when the handoff interval is greater than a checkpoint interval. Thus, multiple checkpoint operations take place within a handoff interval. As seen in Fig. 5(a), the initial state is state 0. A transition to state 2 takes place after a failure. State 2 begins with a recovery operation. A subsequent transition to state 1 takes place after a handoff operation. Fig. 5(b) is a case when the checkpoint interval is greater than a handoff interval. Thus, multiple handoff operations take place within a checkpoint interval. State transitions are similar to the previous case.

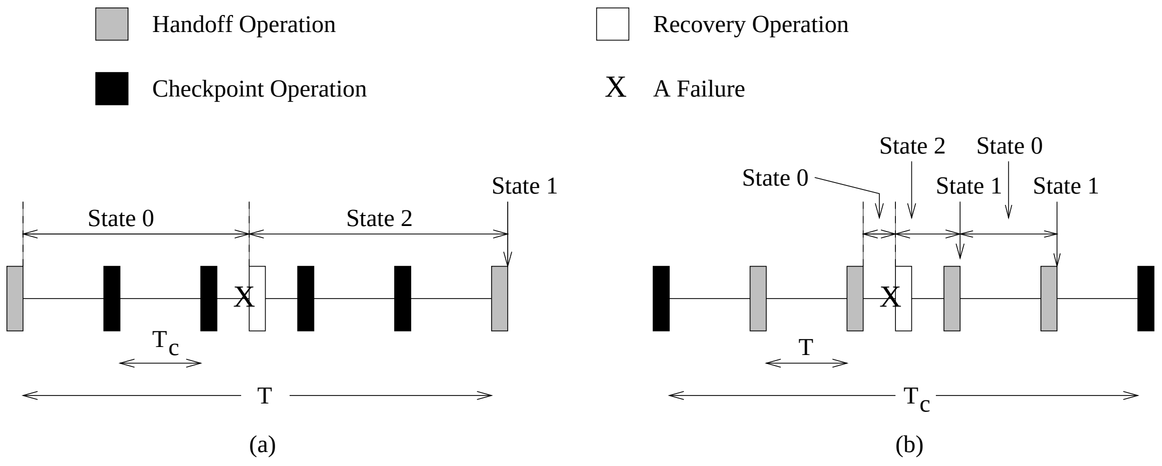


Fig. 5. State Intervals

Each transition (a,b), from state a to state b in the Markov chain, has an associated transition probability P_{ab} and a cost C_{ab}. Cost C_{ab} of a transition (a,b) is the expected total cost of operations that take place during the time spent in state a before making the transition to state b.

The transition probability P_{02} is the probability that a failure occurs within a handoff interval. Let t_f be the time of failure, and t_h be the time of handoff. Then:

$$P_{02} = P(t_f < t_h) = \int_0^\infty \int_{\tau_f}^\infty \lambda\mu e^{-\lambda\tau_f} e^{-\mu\tau_h}\, d\tau_h d\tau_f$$

Solving the above, we get,

$$P_{02} = \frac{\lambda}{\lambda + \mu}$$

The expected duration from the beginning of the checkpoint interval until the time when the failure occurred, given that a failure occurs before the end of the checkpoint interval is,

$$T_{cexp} = \int_0^{T_c} \frac{t\lambda e^{-\lambda t}}{1 - e^{-\lambda T_c}}\,dt = \frac{1}{\lambda} - \frac{T_c e^{-\lambda T_c}}{1 - e^{-\lambda T_c}}$$

As stated earlier, $N_c(t)$ and $N_l(t)$ denote the number of checkpoints and messages logged in t time units, respectively. Cost C_{01} of transition (0,1) is the expected total cost of operations that occur during the time spent in State 0 before making the transition to State 1. C_{01} is as follows: (Recall that T is the mean handoff interval.)

$$C_{01} = (\alpha C_c) * N_c(T) + (\alpha C_l) * N_l(T) + C_h \tag{2.1}$$

Performance metrics for the proposed schemes are:

• **Handoff Time**: The handoff time is the additional time required to transfer the state information from one base station to other, with the overhead of fault-tolerance. Basically it is the difference in the time duration of a handoff operation with fault tolerance and the time duration of a handoff operation without fault tolerance.

• **Recovery Cost**: Upon a failure, this is the expected cost incurred by the recovery scheme, to restore the host to the state just before the failure.

• **Total Cost**: This is the expected cost incurred during a handoff interval with and without failure. The total cost is determined as follows:

$$C_t = C_{01} + P_{02} C_r \tag{2.2}$$

The costs will depend on the state-saving and handoff scheme used. We denote the total cost of a scheme that employs a combination of a state-saving scheme, X ($X \in \{N, L\}$), and a handoff scheme, Y ($Y \in \{P, L, T\}$) as C_{tXY}.

Now, we will derive the costs C_{01}, C_r, and the handoff time for each scheme. The total cost C_t for each scheme can be determined by replacing the costs C_{01} and C_r obtained, in (2.2). Our analysis assumes that the cost of transmitting a message from one node to another depends on the number of hops between the two nodes. We also assume that neighboring base stations are at a distance of one network hop from each other.

3. No Logging-Pessimistic (NP) Scheme

A checkpoint operation takes place upon every write event. Thus, upon every write event, the checkpoint is transferred over the wireless network to the base station, incurring a cost of αC_c, on average. There are r write events during a handoff interval. Since there is no logging operation involved, $N_l(t) = 0, t \geq 0$. During a handoff, the last checkpoint is transferred to the new base station, and in reply, an acknowledgement is sent. Therefore, the cost of handoff $C_h = C_c + C_m$. Thus:

$$C_{01} = (r\alpha + 1)C_c + C_m$$

During recovery, the process state will be present at the current base station. Therefore, the recovery cost is the cost of transmitting a request message from the mobile host to the base station, and the cost of transmitting the state over one hop of the wireless link. Thus:

$$C_r = \alpha(C_c + C_m)$$

4. No Logging-Lazy (NL) Scheme

The checkpoint and logging operations are similar to the NP scheme in Section D.3. However, upon the first checkpoint operation at the current base station, a control message is sent to the base station that has the last checkpoint, requesting it to purge that checkpoint. Let that base station be, on average, N_h hops from that current base station. Thus, the average cost of purging is N_hC_m. A handoff operation includes setting a pointer at the current base station, and transferring a control message between the current and the new base stations. Since setting a pointer does not involve any network usage, the cost of handoff, C_h, is equal to the cost, C_m, of transferring a control message between the two base stations. Thus:

$$C_{01} = r\alpha C_c + N_hC_m + C_m$$

Since a checkpoint operation takes place upon every write event, and the checkpoint is not transferred to the new base station upon a handoff, the location of the last checkpoint will depend on the number of handoffs since the last write event. The upper bound on the number of hops traversed, to transfer the last checkpoint to the current base station, will be the number of handoffs between two write events (or, in this case, checkpoints). In addition to this, the cost of transferring the checkpoint over the wireless link is incurred: αC_c. The average number of handoff operations completed since the last write event (or checkpoint event) until the time of failure is N_h, where:

$$N_h = \mu T_{cexp} \tag{2.3}$$

A cost is also incurred due to the request message from the mobile host for the checkpoint. The cost is $(\alpha + N_h)C_m$. Thus, an upper bound on the recovery cost is

$$C_r = (N_h + \alpha)(C_c + C_m)$$

We will use this C_r to evaluate C_{tNL}. As this C_r estimated is an upper bound, C_{tNL} estimated here is somewhat pessimistic.

5. No Logging-Trickle (NT) Scheme

The checkpoint and logging operations are the same as for the NP and NL schemes described in Sections D.3 and D.4. As in the NL scheme, the handoff cost is the cost of transferring a control message from the current to the new base station. In addition to this, a control message is sent to the previous base station, requesting it to transfer any state corresponding to the mobile host. This ensures that the maximum number of hops traversed, to transfer the state during recovery, is one. The cost of the handoff operation is, thus, the sum of the cost of transferring the state over one hop of wired network, and the cost of sending two control messages. Thus, $C_h = C_c + 2C_m$. It should be noted, however, that the handoff time is only determined by C_m, for the transfer of a control message between the current and the new base station. The time spent due to the transfer of state is transparent to the user.

Upon the first checkpoint operation at the current base station, a control message is sent to the base station that has the last checkpoint, requesting it to purge that checkpoint. Let that base station be, on average, N'_h hops from the current base station. Therefore, the cost of purging is $N'_h C_m$. Thus:

$$C_{01} = (r\alpha + 1)C_c + 2C_m + N'_h C_m$$

As stated earlier, during the recovery operation, the number of hops traversed to transfer the state is, at most, one. Thus:

$$C_r = (N'_h + \alpha)(C_c + C_m) \text{ , where:}$$

$$N'_h = 1(1 - e^{-\mu T_c}) + 0(e^{-\mu T_c}) = (1 - e^{-\mu T_c}) \text{ ,} \tag{2.4}$$

where $e^{-\mu T_c}$ is the probability that the last checkpoint took place at the current base station.

6. Logging-Pessimistic (LP) Scheme

For this scheme, the state of the process will contain a checkpoint and a log of write events. The message log will contain the write events that have been processed since the last checkpoint. The logging cost will involve only those write events that have to traverse the wireless network to be logged at the base station. Only the user inputs need to traverse the wireless network to be logged. On the other hand, write events received from other hosts in the network come via the base station anyway, so they get logged first, and then forwarded to the mobile host. Thus, no cost is incurred due to logging of write events from other hosts. As stated earlier, ρ is the fraction of write events that are user inputs. Thus, ρr is the number of user inputs between two handoffs. This is also the number of logging operations in a handoff interval. For each logging operation, there is a cost for the acknowledgment message sent by the base station over the wireless network. The cost of each acknowledgment message is αC_m.

The handoff cost will now include the cost of transferring the state as well as the message log, and the cost of transferring an acknowledgment. Let ν denote the average log size during handoff. Then, the average handoff cost will be $(\nu C_l + C_c + C_m)$. Under the assumption of handoffs being a Poisson process, $\nu = \frac{k-1}{2}$. (Recall that k is the number of write events per checkpoint.) Thus:

$$C_{01} = \frac{r\alpha C_c}{k} + \rho r\alpha C_l + \rho r\alpha C_m + \nu C_l + C_c + C_m$$

During recovery, the checkpoint and the log are present at the current base station. Therefore, the recovery cost is the cost of transmitting a request message from the mobile host to the base station, and the cost of transmitting the checkpoint and log over one hop of the wireless network. The expected size of the log at the time of failure is ν'. For Poisson failure arrivals, $\nu' = \frac{k-1}{2}$. Therefore:

$$C_r = \alpha(\nu' C_l + C_c + C_m)$$

7. Logging-Lazy (LL) Scheme

The checkpoint and logging operations are the same as for the LP scheme described in Section D.6. When a checkpoint takes place, the old checkpoint and logs at the different base stations are purged. As also determined earlier in Section D.4, the purging cost is $N_h C_m$, and the handoff cost is C_m.

$$C_{01} = \frac{r\alpha C_c}{k} + \rho r\alpha C_l + \rho r\alpha C_m + N_h C_m + C_m$$

As determined earlier, the expected number of write events completed until the time of failure since the last checkpoint is $\nu' = \frac{k-1}{2}$. This is distributed over different base stations. The last checkpoint and the logs have to traverse, on an average, N_h (2.3) hops on the wired network to reach the current base station, and an additional wireless hop to reach the mobile host. A cost of $(N_h + \alpha)C_m$ is also incurred due to the request message for the checkpoint and the logs (same as for NL scheme). Therefore,

$$C_r = (N_h + \alpha)(\nu' C_l + C_c + C_m)$$

8. Logging-Trickle (LT) Scheme

The checkpoint and logging operations are the same as in LP and LL. The cost of handoff operation is, thus, the sum of the cost of sending two control messages (same as for NT scheme), and the cost of transferring checkpoint and logs over one hop of wired network. Thus, $C_h = \nu C_l + C_c + 2C_m$. The cost of purging is $N'_h C_m$. Thus:

$$C_{01} = \frac{r\alpha C_c}{k} + \rho r\alpha C_l + \rho r\alpha C_m + \nu C_l + C_c + 2C_m + N'_h C_m$$

$$C_r = (N'_h + \alpha)(\nu' C_l + C_c + C_m)$$

9. Results

The above equations have been normalized with respect to C_c. Recall that γ is the relative logging cost and is equal to C_l/C_c. Thus, $C_l = \gamma C_c$. Recall that ϵ is the relative control message cost and is equal to C_m/C_c. We assume that $C_m \ll C_c$ (which is the case, in practice). We replace $C_c = 1$, $C_l = \gamma$, and $C_m = \epsilon$ in the above equations and determine the handoff time, recovery cost and the total cost. The rate of writes β is set to 1.

For our analysis, we assume that $\rho = 0.5$. (Recall that ρ is a fraction of write events that are user inputs.) This means that the write events comprise an equal percentage of user inputs and messages from other hosts. For our analysis, we fix the relative control message cost, $\epsilon = 10^{-4}$.

a. Optimum Checkpoint Interval

An optimum checkpoint interval is required to be determined only for the Logging schemes. Recall that for a No Logging scheme, a checkpoint takes place upon every write event. However, for a Logging scheme, a checkpoint takes place periodically every T_c units of time. Since the rate of writes β is equal to 1, the number of write events per checkpoint (k) is equal to T_c. A "good" value for k needs to be chosen for the Logging schemes. We define a good value of k to be the one that offers the minimum total cost. This value of k (say, $k_{opt_{LY}}$, for a Logging scheme that uses scheme Y for handoffs: $Y \in \{P, L, T\}$) is a function of the failure rate λ, relative logging cost γ, wireless network factor α and communication-mobility ratio r. Let us consider the LL scheme as an example. The value of $k_{opt_{LL}}$ for the LL scheme is obtained as a solution of:

$$\frac{\partial C_{tLL}}{\partial k} = 0 \text{ and } \frac{\partial^2 C_{tLL}}{\partial^2 k} < 0$$

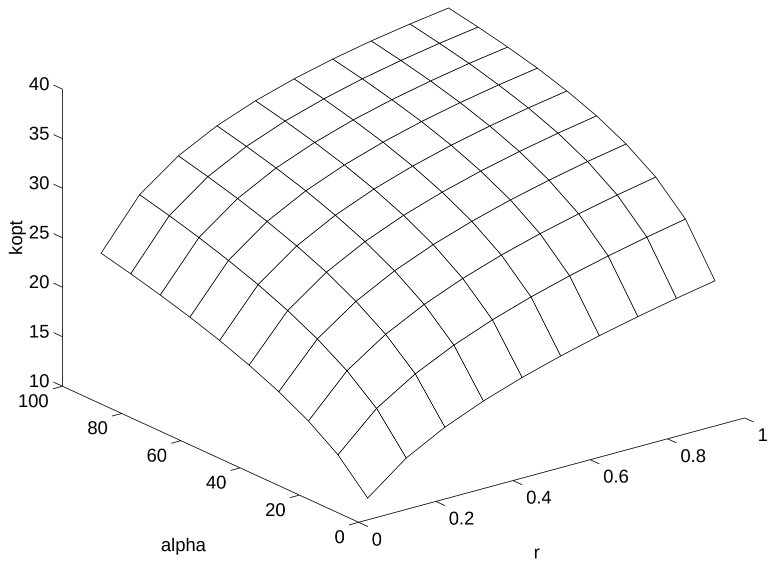


Fig. 6. $k_{opt_{LL}}$ vs. r and α: $\lambda = 10^{-2}$, $\gamma = 0.1$

Fig. 6 illustrates the variation of $k_{opt_{LL}}$ with r and α for $\lambda = 10^{-2}$ and $\gamma = 0.1$. Note that $k_{opt_{LL}}$ increases as r and α increase. For a given k, as r increases, the number of checkpoints per handoff increases. This increases the total cost. As α increases, the cost due to a checkpoint increases. Thus, to lower the total cost, k should also increase. Therefore, as r and/or α increases, $k_{opt_{LL}}$ also increases.

Fig. 7 illustrates the variation of $k_{opt_{LL}}$, with γ and λ for $r = 0.1$ and $\alpha = 10$. Note that $k_{opt_{LL}}$ decreases as γ and λ increase. As γ increases, the cost of the logging operation increases. Thus, checkpoint interval size has to be reduced to decrease total cost. Therefore, $k_{opt_{LL}}$ decreases as γ increases. As λ increases, the probability of failure increases. Thereby, the fraction of recovery cost in the total cost increases. The recovery cost for the Logging schemes depends on the average log size during failure. The average log size, in turn, depends on checkpoint interval size. To decrease recovery cost, we need to reduce checkpoint interval size. Thus, as λ increases, $k_{opt_{LL}}$ decreases.

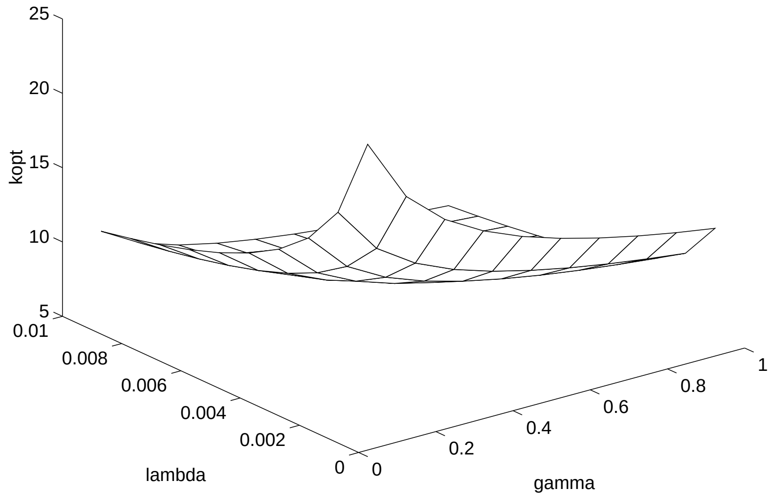


Fig. 7. $k_{opt_{LL}}$ vs. γ and λ: $\alpha = 10$, $r = 0.1$

Similar behavior was observed for the LP and LT schemes. We used $k = k_{opt_{LY}}$ for the analysis of the Logging scheme which uses scheme Y for handoffs, where $Y \in \{L, P, T\}$. We assume that relative logging cost $\gamma = 0.1$. We vary α to represent different classes of wireless networks. We vary λ to represent different failure rates. We vary the value of r to represent different user mobility patterns. We will now illustrate the performance of each of the proposed schemes.

b. Handoff Time

Recall that the handoff time is the additional time required, due to the transfer of state information by the fault tolerance scheme during handoff operation. Let BW be the bandwidth of a link on the wired network. Table II illustrates the handoff cost and (handoff time $\times BW$) of the various schemes. The Pessimistic handoff schemes incur a very high handoff *time* compared to the Lazy and Trickle handoff schemes. This is because in the Lazy scheme, there is no state transfer during handoff. In the Trickle scheme, the state transfer is performed separately from the handoff. Note, however, that for a given state-saving scheme, the handoff *cost* of the Trickle handoff scheme is almost equal to the Pessimistic handoff scheme.

Table II. Handoff Cost and (Handoff Time $\times BW$)

Scheme	Handoff Cost	(Handoff Time $\times BW$)
NP	$1+\epsilon$	$1+\epsilon$
NL	ϵ	ϵ
NT	$1+2\epsilon$	ϵ
LP	$1+\nu\gamma+\epsilon$	$1+\nu\gamma+\epsilon$
LL	ϵ	ϵ
LT	$1+2\epsilon+\nu\gamma$	ϵ

c. Recovery Cost

In Fig. 8, we plot the recovery cost for all the schemes for $\alpha = 10$, and $\lambda = 10^{-2}$. Similar behavior was observed for other values. As expected, the recovery cost of the Logging schemes is more than the No Logging schemes. The recovery cost of the

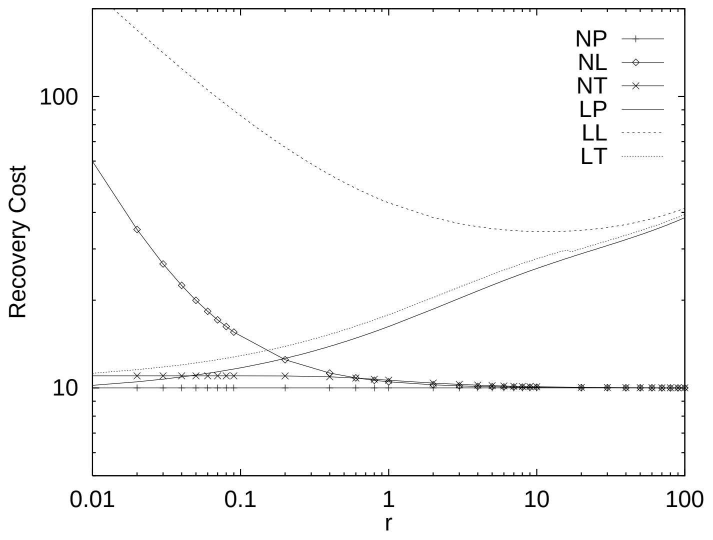


Fig. 8. Recovery Cost: $\lambda = 10^{-2}$, $\alpha = 10$

NP scheme is independent of r. The NP scheme incurs the lowest cost for all values of r. This is because the last checkpoint state is always present at the current base station. The recovery cost of the NT scheme is a constant for low r ($r < 1$), and slightly more than the NP scheme. This is because the last checkpoint of the host is always available one hop from the current base station. As stated earlier, β is fixed for the analysis. For a fixed β, as μ (i.e.; mobility) decreases, r ($= \beta/\mu$) increases, and the probability of the last checkpoint being available at the current base station increases. Therefore, at high values of r ($r > 1$), the costs of NT and NP converge.

The recovery cost of the LP and the LT schemes is proportional to the size of the log before failure. The size of the log depends on k. Since k ($= k_{opt_{LP}}$ or $k_{opt_{LT}}$) increases with r, the recovery cost also increases. Similar to NP and NT schemes, at low values of r ($r < 1$), the recovery cost of the LT scheme is slightly higher than LP scheme. However, at high values of r, the costs of LP and LT schemes become similar.

For low values of r ($r < 1$), note that the recovery cost of the Lazy handoff (LL

and NL) schemes are much larger than for the Pessimistic and the Trickle handoff schemes. This is because the checkpoint state might not be at the current base station. Secondly, the log of write events might be distributed at different base stations. Thus, the cost of recovery will include the cost of transferring the checkpoint state and the log from the various base stations to the current base station, and then forwarding them to the mobile host over the wireless link. The LL scheme incurs a very high recovery cost for low r. The lower the value of r, the greater the amount of scatter of recovery information. As r increases, the possibility of a checkpoint operation taking place at the current base station increases. Thus, the recovery cost decreases as r increases. However, as r increases, k $(= k_{opt_{LL}})$ also increases. Thus, after some value of r, the recovery cost starts increasing. On the other hand, the recovery cost of the NL scheme continues to decrease as r increases. At high values of r $(r > 1)$, the cost of NL converges to NP and NT. Similarly, the cost of the LL scheme becomes similar to LP and LT.

As expected, at high values of r (i.e., low mobility), the recovery cost becomes almost independent of the handoff scheme used – the state-saving scheme determining the recovery cost.

d. Total Cost

Fig. 9 illustrates the variation of total cost of various schemes with r, for $\lambda = 10^{-2}$ and $\alpha = 10$. The total cost is comprised of the failure-free cost and the recovery cost. The total cost of the Pessimistic handoff scheme and the Trickle handoff scheme are almost equal ($NP \approx NT$, and, $LP \approx LT$). The Lazy handoff scheme incurs a lower total cost at low values of r $(r < 1)$. At high values of r, the total cost of the different handoff schemes converge. However, the difference in the total costs of the Logging and No Logging schemes remains. The total cost of No Logging scheme is higher than

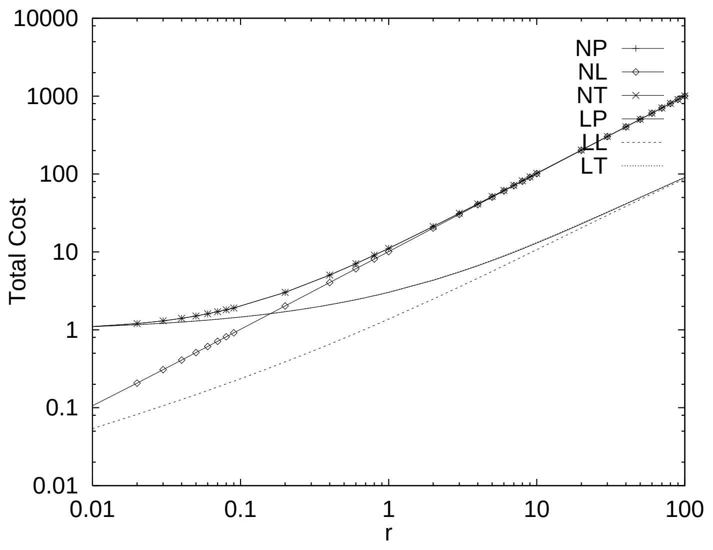


Fig. 9. Total Cost: $\lambda = 10^{-2}$, $\alpha = 10$

the Logging scheme for all values of r. The LL scheme incurs the lowest total cost for all r.

Fig. 10 illustrates the variation of the total cost with r, for $\lambda = 10^{-5}$. Comparison of Figures 9 and 10 indicates that, for the same α, as λ decreases, the cost difference between the handoff schemes for the Logging state-saving scheme increases. As the probability of failure decreases, the Lazy handoff scheme becomes more justified. The total costs of the Trickle and the Pessimistic handoff schemes are almost always equal, and both are higher than the Lazy scheme.

Fig. 11 illustrates the variation of the total cost with r, for $\alpha = 500$. The total cost increases with α. Comparison of Figures 9 and 11 indicates that, for the same λ, as α increases, the cost difference between the handoff schemes reduces. Thus, the performance of a scheme becomes more dependent on the state-saving scheme used than on the handoff scheme.

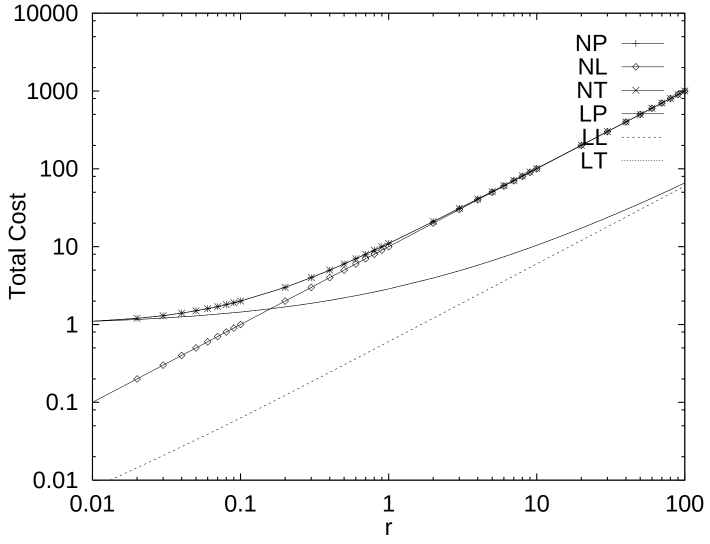


Fig. 10. Total Cost: $\lambda = 10^{-5}$, $\alpha = 10$

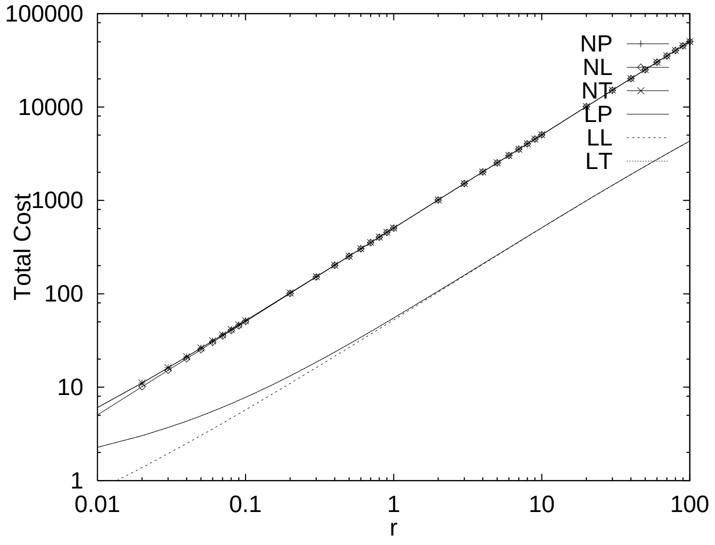


Fig. 11. Total Cost: $\lambda = 10^{-2}$, $\alpha = 500$

10. Discussion

Handoff time of Pessimistic handoff schemes is very high, and unacceptable for applications that require connection-oriented services. During a handoff period, there are no packets sent or received by the mobile host. Thus, if handoff time is very high, the communication protocols used for these connection-oriented services might timeout and/or the mobile host might notice long disruption in service during handoffs [14].

Some applications might require a very quick recovery, and some other applications might require a very low total cost to be incurred by the recovery schemes. Some hosts might be running the application in a high failure rate environment, and some in a very low failure rate environment. As can be observed from the results, among the schemes considered, there is no single recovery scheme that performs best (lowest total cost, lowest recovery cost and lowest handoff time) for all environments.

We will now determine the environments where a particular recovery scheme is best suited. For the sake of convenience, we divide the communication-mobility ratio (r) space into two regions: *low* ($r \leq 1$), and *high* ($r > 1$). We divide the wireless network factor (α) space into two regions: *low* ($\alpha \leq 50$), and *high* ($\alpha > 50$). We also divide the failure rate (λ) space into two regions: *low* ($\lambda \leq 10^{-3}$), and *high* ($\lambda > 10^{-3}$). In our discussions, we will refer to the regions instead of actual values. The summary of the results are presented in Fig. 2.

In a low failure rate environment, failures occur very infrequently. The primary goal of a recovery scheme in such an environment is to incur low failure-free cost. The LL scheme incurs low failure-free cost for all values of r. However, for high α values, the difference in the failure-free costs of the LL and LT schemes reduces. Since the recovery time (as determined by recovery cost) of the LT scheme is much lower than for the LL scheme for low values of r, it is preferable to choose LT for high α values.

In a high failure rate environment, failures occur very frequently. The primary goal of a recovery scheme is to incur low failure-free cost and low recovery cost. For low r values, the recovery cost of the LL and NL schemes is very high. Thus, we need to choose between NT or LT. When α is low, NT incurs a low failure-free cost (slightly more than LT), and provides a quicker recovery than LT. However, when α is high, LT becomes preferable. For high r values, LL is preferable over other schemes.

E. Summary

The new mobile wireless environment presents many challenges due to the mobile nature of the hosts and the limited bandwidth on the wireless network. Presented in this chapter are recovery schemes for a mobile wireless environment. The recovery schemes are a combination of a state-saving strategy and a handoff strategy. Two strategies for state-saving, namely, (i) *No Logging* and (ii) *Logging*, and three strategies for handoff, namely, (i) *Pessimistic*, (ii) *Lazy*, and (iii) *Trickle* are discussed.

Our main goal here is to present the limitations of the new mobile computing environment, and its effects on recovery protocols. The trade-off parameters to evaluate the recovery scheme were identified. It was determined that, in addition to the failure rate of the host, the performance of a recovery scheme depended on the mobility of the hosts and the wireless bandwidth. We analyzed the performance of the various recovery schemes proposed in this chapter, and determined those mobile environments where a particular recovery scheme is best-suited.

CHAPTER III

CENTRALIZED LOCATION MANAGEMENT

A. Introduction

It is expected that existing wireless cellular networks will be upgraded for personal communication services (PCS) [20, 31, 58]. An important issue in mobile wireless networks is the design and analysis of location management schemes. In order to communicate with any particular user, it is first necessary to locate the user in the network. This is due to the fact that the users are mobile and they could be anywhere in the area covered by the network.

An analysis in [57] shows that if location updates are to occur on each cell crossing the resulting signalling load will have a major impact on the load of the network. The additional signalling traffic on the SS7 signalling system (capacity of 56 Kbps) is expected to be 4-11 times greater for cellular networks than for ISDN and 3-4 times greater for future personal communication networks (PCN) than for cellular networks. The signalling load due to updates alone increases network load by 70%. Thus location updates will become a major bottleneck at the switches such as SS7, and hence mechanisms to control the cost of location update are needed. It is thus evident that new efficient location management strategies need to be investigated for personal communication services (PCS).

Location management consists of location *searches* and *updates*. A *search* occurs when a host wants to communicate with a mobile host whose location is unknown to the requesting host. An *update* occurs when a mobile host changes location.

This chapter attempts to study the effect of forwarding pointers and search-updates on location management in Personal Communication Networks (PCN). Location management for PCS utilize the fact that there is a home location server (HLS) for every mobile host. There are existing standards for carrying out location management using home location servers (HLS), e.g., (EIA/TIA) Interim Standard 41 (IS-41), and Global System for Mobile Communications (GSM) in Europe [58]. However, these schemes are not efficient, due to increase in network load and location management costs. To overcome these drawbacks, this chapter presents location management strategies using forwarding pointers in addition to HLS. We present two heuristics to limit the number of forwarding pointers traversed during a search, namely, *movement-based* and *search-based*. We show that significant improvement can be obtained using forwarding pointers in addition to HLS. The chapter also presents a strategy to perform *search-updates*. A search-update occurs after a successful search, when the location information corresponding to the searched mobile host is updated at some hosts. Analysis shows that performing search-updates significantly reduces the search costs. However, search-updates also increase the total network load. This extra network load is due to forwarding pointer maintenance.

The performance of the proposed heuristics depend on (i) the relative cost of setting and traversing the forwarding pointers (parameter α), and, (ii) call and mobility patterns of the user (parameter r). Through analysis, we show that among the schemes considered, there is no single location management scheme that performs well for all values of r and α. However, among the schemes considered, we determine the best location management scheme for each environment, as shown in Fig. 12. $\mathcal{M}$ is a parameter that determines the performance of the movement-based heuristic scheme. It will be explained in detail in Section D.2.b.

On the downside, forwarding pointers are prone to failures anywhere along the chain of pointers. Omission/corruption of a forwarding pointer could cause the host to be intractable. This chapter proposes cost-effective techniques for fault tolerance

r \ α	Low	High
Low	Search-based with Search-Updates	Movement-based (M<5)
Moderate	Movement-based with Search-Updates (M<5)	Movement-based (M<3)
High		Search-based

Fig. 12. Best Location Management Scheme

and automatic recovery of the network, and also for forwarding pointer maintenance. This chapter then shows that the memory overhead due to forwarding pointers is insignificant when compared to savings in terms of network load.

This chapter is organized as follows. Review of related literature in location management is presented in Section B. Section C presents a network architecture for a distributed system with mobile hosts. Section D presents the proposed location management scheme using forwarding pointers, and the corresponding performance analysis. Section E presents the search-update strategy and the corresponding performance analysis. Section F presents schemes to make the proposed location management scheme robust. Memory overhead analysis is presented in Section G. Section H discusses algorithms to determine the call-mobility ratio of the users. The work presented in this chapter is compared with other centralized approaches in Section I. Summary is presented in Section J.

B. Related Work in Location Management

Numerous location strategies have been proposed in the recent years. One of the earlier works which dealt with object tracking was done in 1986 by Fowler [21]. Fowler deals with techniques to efficiently use forwarding addresses for finding decentralized objects. Our research borrows the idea of manipulating forwarding pointers upon a successful search.

Awerbuch et.al. proposed a theoretical model for online tracking of mobile hosts [4]. Their architecture consists of a hierarchy of "m-regional matching directories". In their scheme, forwarding pointers and regional matching directories are used to enable localized updates and searches.

Spreitzer et.al. proposed a network architecture which consists of *user agents* and a *location query service* (*LQS*) [72]. There is a dedicated user agent per user and these user agents are responsible to forward any communication to or from the user. This scheme is mainly aimed for local networks – the kind used within building premises. As the number of hosts in a network increase, it might not be efficient to have a dedicated user agent per user.

Wu et. al. dealt with the idea of caching location data at the Internet Access Point(IAP) [79]. Here, the IAP will maintain location data of some of the hosts. This becomes useful when optimal routing decisions are to be taken. If the IAP does not have an entry for a host, the message is forwarded to the *Mobile Router* (*MR*) which maintains information of all the hosts. It is a very simple idea that will be effective for local networks. However, when the network sizes increase, the *MR* will become a serious bottleneck, and one has to resort to more efficient location management techniques.

Ioannidis et.al. proposed IP-based protocols for providing continuous networks access for mobile computers using caching and forwarding technique [32]. However, these protocols are primarily proposed for a campus environment with mobile computers.

Badrinath et.al. examine strategies that reduce search costs and control the volume of location updates by employing user profiles [5]. Their architecture consists of a hierarchy of location servers which are connected to themselves and to the base stations (or mobile support stations) by a static network. User profiles are used to create partitions. It is only when the user crosses partitions that the update takes place. However, in most cases, user profiles are not always available a priori.

A modified tree structure for location management was proposed in [19]. The root and the some of the higher levels of the tree was replaced by a *set-ary* butterfly network. This helped in balancing the search requests at the nodes. Also proposed were schemes to make the protocol self-stabilizing.

The idea of location management with *home location servers* (*HLS*) has also been proposed in [36, 37]. Caching [36] and forwarding [37] have been independently proposed for Personal Communication Services (PCS). Similar to our work in this chapter, these strategies augment the basic location strategy which uses HLS. Section I compares them with the work presented in this chapter. Other techniques like user profile replication [71] and local anchoring [29] have also been proposed to reduce network load due to location management. In the next section we will present the network architecture typically used in centralized location management schemes.

C. Network Architecture

Cells are grouped into *registration areas*. There is a location server in each registration area. Each mobile host is assumed to be permanently registered to a particular registration area. The location server of that registration area is called the *home location server* (*HLS*) for the mobile host. This association of a host with a particular *home location server* is fully replicated across the whole network. The *home location server* is responsible for keeping track of the location information of the mobile host. A location server is also responsible for maintaining location information of the mobile hosts currently residing (visitors) in all cells within its registration area[1].

The events that can cause change in the location information of a mobile host are when the mobile host (i) *Switches ON*, (ii) *Switches OFF*, (iii) *Handoffs*, and (iv) *Crosses registration areas.* The details of the events (i), (ii) and (iii) are discussed in Appendix A. In this research we are primarily interested in analysing strategies for location management due to registration area crossings, because these strategies involve updates at the home location servers and thus increasing the network traffic. In the next section we will discuss the location management strategies for registration area crossings.

D. Location Management

In this section, we first present the scheme for updates and searches that is being currently used in IS-41. We will then present the drawbacks of these schemes. Later in this section we will present the proposed location management scheme using forwarding pointers, and the corresponding performance analysis.

1. Overview of the Basic IS-41 Scheme

Location management with home location servers is being used in current personal communication systems (PCS) standards proposals such as IS-41 [58].

[1]The terminology used in IS-41 literature is slightly different. IS-41 uses home location register (HLR) and visitor location register (VLR) databases. However, the information maintained in the HLR is same as what is maintained in HLS, the information maintained in the VLR is maintained in the various location servers.

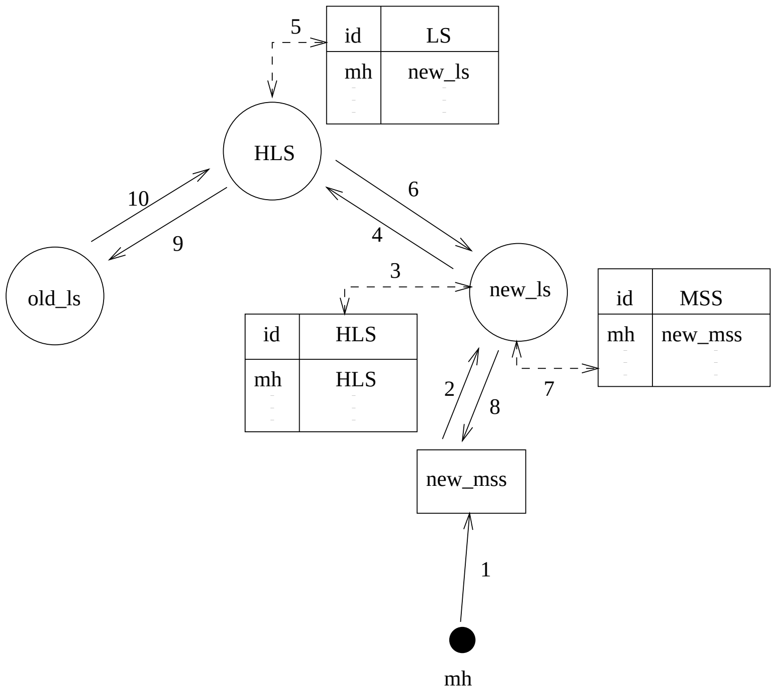


Fig. 13. Update in the Basic Scheme (IS-41)

a. Updates in the Basic IS-41 Scheme

Updates take place only when the mobile host (mh) enters a new registration area. Let the old registration area be *old*, and the new registration area be *new*. Please refer Fig. 13 for this discussion. Upon entering *new*, the mobile host (mh) informs the new mobile support station (new_mss), which forwards the information to the location server of registration area *new*, named new_ls (steps 1-2 in Fig. 13). The new_ls looks up its database to determine the home location server (HLS) of mh (step 3). The new_ls then sends a message to HLS notifying it about the new location of mh (step 4). The HLS updates the location information of mh (i.e., changes the current location server to new_ls), and sends the host information (user profile, etc.) to new_ls (steps 5-6). The location server new_ls stores the user profiles, updates its host database and sends an acknowledgement to new_mss (steps 7-8). The HLS

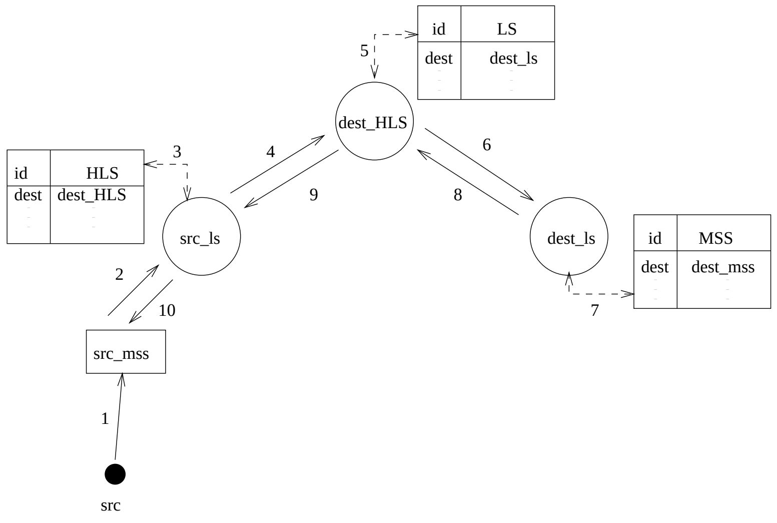


Fig. 14. Search in the Basic Scheme (IS-41)

also sends a message to the location server of registration area *old*, named *old_ls*, requesting it to delete any host information stored at *old_ls* (step 9). The location information of the mobile host is then purged from *old_ls*, after that, *old_ls* sends a confirmation message to *HLS* (step 10).

b. Searches in the Basic IS-41 Scheme

Let us suppose that a mobile host *src* wants to communicate with another mobile host *dest*. Let *src_mss* be the *MSS* of the cell in which host *src* currently resides. When *src* wants to communicate with a mobile host *dest*, it has to first determine the location of *dest*. Host *src* sends a location query message to the mobile support station for its cell, *src_mss* (step 1 in Fig. 14). The *src_mss*, forwards the query to the location server (say *src_ls*) in its registration area (step 2 in Fig. 14). The location server *src_ls* checks whether *dest* is in its registration area[2]. If *dest* is in its registration

[2]In IS-41 standard, every call results in a query to the *HLS* of *dest*. In this chapter we have altered the search procedure slightly to search the registration area of the caller first. Thus, the search procedure of IS-41 standard will in fact result in an even

area, then it returns the location information. Otherwise, *src_ls* looks up its database to determine the *HLS* of *dest* (say *dest_HLS*) (step 3). The location server *src_ls* then forwards the location query message to *dest_HLS* (step 4). The *dest_HLS* maintains the identification of the location server (say *dest_ls*) of the registration area in which the last update of the location of host *dest* took place (step 5). The *dest_HLS* queries the *dest_ls* for the cell location of *dest* (step 6). In reply, *dest_ls* sends the identifier of the mobile support station *dest_mss*. Upon getting the reply from *dest_ls* (steps 7 and 8), *dest_HLS* returns the current location of *dest* to *src_ls*, which, in turn forwards it to *src_mss* (steps 9 and 10). The location information is nothing but the address of the *MSS* (named *dest_mss*) of the cell in which the mobile host *dest* currently resides. Thereafter, the call is set up between *src* and *dest* via *src_mss* and *dest_mss*.

c. Drawbacks

- Increase in network traffic when the host crosses registration areas very frequently: This can be due to a very mobile host going in some specific direction, or because the host moves in and out of a registration area such that there are lot of registration area boundary crossings. As every registration area crossing causes an update at the host's *HLS*, this scheme increases the network traffic.

- Inefficient location management: Updates on each registration area crossing is useful to reduce the search costs (i.e., *call set-up* time), only if the user is being called frequently. However, if the user is not being called frequently, updates on each crossing are not necessary. In such scenarios, updates on each crossing lead to inefficient location management.

---

higher network load than the search procedure presented here.

2. Proposed Scheme

In this section we will discuss the proposed scheme that tries to avoid the drawbacks of the above location management strategy. We use forwarding pointers in addition of HLS to assist in location management. Our main goal is to avoid the increase in network traffic due to updates at the HLS. We achieve this by not updating the location information after every registration area crossing. Instead, forwarding pointers are maintained at the location servers of the registration areas visited as described below.

We classify location *updates* further into two types, namely, *updates* and *search-updates*. A *update* occurs when a mobile host changes location. A *search-update* (also known as caching [36]) occurs after a successful *search*, when some hosts update the location information corresponding to the searched mobile host. We discuss more about search-updates in Section E.

a. Forwarding Pointers

A forwarding pointer maintained at some location server for a mobile host mh is a data structure that contains the following: (i) identifier of mh, (ii) *location* of the host mh. The key for the forwarding pointer database is the identifier of mh, i.e., there can be only one forwarding pointer per mobile host.

The forwarding pointers for a mobile host can be interpreted as forming a directed graph with the location servers as vertices and with an edge from location server ls_1 to location server ls_2 if and only if ls_1 has a forwarding pointer that points to ls_2. The resulting graph will be a tree with edges directed towards the root, the current location of the mobile host [21].

Example D.1: For an easier understanding we will illustrate with an example. Let the notation *move*(a,b) represent a move of the mobile host from registration area *a* to registration area *b*. Let a mobile host make the following moves: *move*(1,2), *move*(2,3), *move*(3,2), *move*(2,4), and *move*(4,5). The resulting graph due to the forwarding pointers is shown in Fig. 15(a). Now, the mobile host makes the following move, *move*(5,2). The mutation of the graph due to *move*(5,2) is shown in Fig. 15(b).

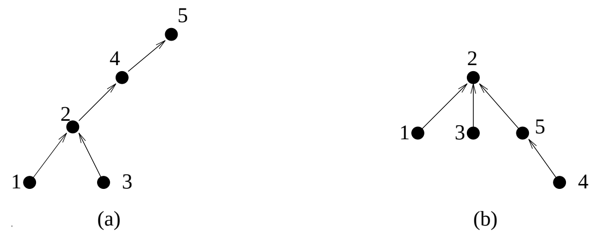


Fig. 15. An Example of Forwarding Tree

The following obervations can be made [21]:

1) The graph due to the forwarding pointers will be a tree with edges directed towards the root, the current location of the mobile host. □

2) The resulting graph after mutation due to a move of the mobile host still remains a tree, however, with a different root, the new location of the mobile host. □

3) It follows from (1) and (2) that there are no loops in the graph formed by the forwarding pointers. □

Transient Loops: Transient loops are unavoidable. Transient loops could be caused due the following reason: A call for host h' is set up based on a location of h'; however, before the call reaches h', host h' moves to some other location. This will require the call to traverse a path which might have loops. For example, in Fig. 15(a), let a call be set-up from a host h in registration area 2 to a host h' which is currently in registration area 5. Thus, it will take the path along the chain of forwarding pointers, $2 \rightarrow 4 \rightarrow 5$. Let the host h' move to registration area 4 when the "*call set-up*" message was in transit from 4 to 5. Thus, the "*call set up*" message has to traverse the forwarding pointer $5 \rightarrow 4$, thereby visiting registration area 4 twice. However, these loops are short-lived, and do not exist under stable conditions.

b. Updates Using Forwarding Pointers

Whenever a mobile host leaves a registration area, say *old*, and enters a new registration area, say *new*, the host information (e.g., user profile, number of forwarding pointers created due to the moves, etc.) is transferred from *old_ls* to *new_ls*, where, *old_ls* (*new_ls*) is the location server for *old* (*new*) registration area. In addition, a forwarding pointer is created at *old_ls* to point to *new_ls* to indicate that the host has moved to *new*.

Update at the *HLS* does not take place for every registration area crossing. When an update takes place is determined using one of the heuristics stated below:

- Movement-based heuristic: The location information of a host at the *HLS* is updated when the number of registration area crossing by the host is $\mathcal{M}$, where $\mathcal{M}$ is a constant parameter that determines the performance of this heuristic.
- Search-based heuristic: The location information of a host at the *HLS* is updated when the number of search requests for the host is $\mathcal{S}$, where $\mathcal{S}$ is a constant parameter that determines the performance of this heuristic.

The procedure to update the *HLS* in the proposed scheme is same as the update procedure in Section D.1.a. After such an update, the *HLS* will know the present location of the mobile host, and no forwarding pointers need to be traversed (until the mobile host moves from its present registration area).

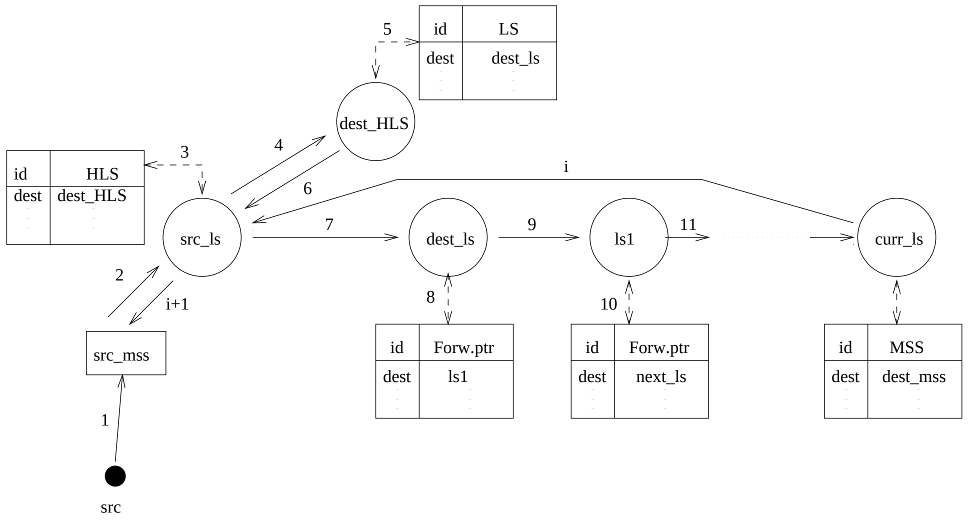


Fig. 16. Search using HLS and Forwarding Pointers

c. Location Searches Using Forwarding Pointers

Let us suppose that a mobile host *src* wants to communicate with another mobile host *dest*. Please refer Fig. 16 for this discussion. The first four steps of the search procedure are identical to steps 1-4 in Fig. 14, until the location query is forwarded to the home location server of *dest*, *dest_HLS*. *Dest_HLS* returns the address of the location server (*dest_ls*) of the registration area in which the mobile host was residing when the last update at *dest_HLS* took place (steps 5 and 6). Upon receiving the address of *dest_ls*, *src_ls* sends a message query that traverses the chain of forwarding pointers originating at *dest_ls* (step 7). Each location server on the chain looks up its forwarding pointer database to determine the next location server on the chain, and forwards the location query to it (e.g., steps 8 and 9 for *dest_ls*). This continues till the end of the chain is reached (say *curr_ls* is the location server). The host *dest* will be located in a cell in the registration area of *curr_ls*. The location server *curr_ls* returns the current cell location to *src_ls*, which in turn forwards it to *src_mss* (steps i and $i+1$ in Fig. 16). Thereafter, the call is set up between *src* and *dest* via the

src_mss and $dest\_mss$.

d. Forwarding Pointer Maintenance

Forwarding pointers for a host may remain at the location servers for a long time containing stale information and also increasing the storage requirements at the location servers. A forwarding pointer can potentially be purged if it has not been used for a "long" time. To avoid any inconsistencies in the location information due to purging, we require the location servers to maintain a timestamp associated with each forwarding pointer. While creating a forwarding pointer, the timestamp is the current time at the location server when the pointer is created. Along with the timestamp, the purge time interval t_p for that pointer is also maintained. Every location server purges the forwarding pointers which are older than purge time interval t_p units of time. Through this process, we avoid maintaining any stale forwarding pointers at the location servers.

A Note: The above data structure contains a field to store time. The time entry for a data structure on a location server, say ls, contains the *local* time at location server ls when the data structure was last modified. It should be noted that the correctness of the algorithms does not require the clocks at various location servers to be tightly synchronized.

However, there might be "dangling" pointers due to inconsistent purging. For example, let there be a chain $5 \rightarrow 4 \rightarrow 2$. Thus, the current location of the host is 2. Suppose, 4 purges its pointer to 2 before 5 purges its pointer to 4. Then, $5 \rightarrow 4$ is a "dangling" pointer because it does not lead to the location of the host i.e., 2. To avoid this, a trivial solution is to keep a constant value of t_p for all pointers[3]. Let

[3]This requires that clocks of all the location servers progress at an identical rate.

it be T_{purge}. Let us consider the example again. Since, the pointer $4 \rightarrow 2$ is created after pointer $5 \rightarrow 4$, the pointer $5 \rightarrow 4$ will be purged before pointer $4 \rightarrow 2$. Thus, there will not be any "dangling" pointers.

We also need to ensure that the forwarding pointers do not get purged before an HLS update takes place. Thus, the mobile host should update its location information at its HLS at least once every T_{purge} units of time. This will ensure that even if the forwarding pointers are purged, the HLS has the current location information of the host. T_{purge} will be a system design parameter. Larger the value of T_{purge}, lower will be the volume of HLS updates due to purging. However, larger the value of T_{purge}, longer will the forwarding pointers be in the system, thus increasing the storage requirements at the location server. In a later section we derive the relation between T_{purge} and the memory overhead at the location servers.

In the analysis presented in the next section, we assume that T_{purge} is large enough (as compared to the time period between two HLS updates due to heuristics) such that HLS updates due to timeout form a small fraction of all HLS updates. So, we will ignore those cases. We now analyze the performance of the search-based heuristic and the movement-based heuristic.

3. Performance Analysis

In this section we analyze the proposed location management scheme using forwarding pointers and compare it with the basic scheme (IS-41). We will then compare the

---

This may not be valid in practice, because, the clocks will have different rates. However, we can tackle this by having different purge time intervals for the location servers. The difference between the purge time intervals at any two location servers will mainly depend on the relative clock rate of the two location servers. For simplicity of explanation we assume that clocks at all the location servers progress at an identical rate.

performance of the two heuristics presented in this chapter.

Calls between two hosts within the same registration area will not involve the home location servers. Moves within the registration area will also not involve any home location server interaction. Therefore, we analyze the performance of the location management schemes for calls which arrive from outside the registration area of a host (hereafter, we refer them simply as call arrivals), and for moves which are registration area crossings. We define "cost" of sending a message as the amount of network usage due to sending the message. In other words, it is the amount of time the network is busy trasmitting the message.

The following terminology is used in the analysis:

- r = average number of calls per registration area crossing. We call this the call-mobility ratio.

- C_u = cost of an update per registration area crossing using IS-41. We will derive an expression for C_u later.

- C_s = cost of a location search using IS-41. We will derive an expression for C_s below.

- C'_u, C'_s are the update and search cost, respectively, using forwarding pointers.

- cost$(x \rightarrow y)$ = cost of sending a message from x to y.

- A = average cost of sending a location query/reply message between MSS and the LS within a registration area.

- B = average cost of sending a location query/reply message between a location server and a home location server. This is thus the average cost of a HLS interaction.

- k = For a caller host, this is the number of forwarding pointers traversed before locating the destination host.

- F = cost of *setting* a forwarding pointer between two location servers. This cost includes the cost of sending a message between the two location servers. The cost of *traversing* a forwarding pointer also includes the cost of sending a message between two location servers. The cost of traversing a forwarding pointer is also assumed to be F.

- α = Relative forwarding cost $= \frac{F}{C_u}$.

In this chapter, our goal is to analyze the load on the static network. Therefore, we do not consider the cost of message transfers over wireless links.

a. Performance Metrics

The performance metrics of the schemes are **search cost** (C_s), and **total cost** (C_t) where,

$$C_t = \mathcal{X} C_u + \mathcal{Y} C_s$$

We need to choose $\mathcal{X}$ and $\mathcal{Y}$ such that the following holds:

- For users with low r (i.e., moves are more frequent than calls received), the update cost is the governing factor in the total cost. Thus, in this case, the search cost should be dampened.

- For users with high r (i.e., calls received are more frequent than moves), the search cost is the governing factor in the total cost. In this case, the update cost should be dampened.

To achieve the above, we choose $\mathcal{X} = \frac{1}{r}$, and $\mathcal{Y} = r$. Thus, the performance metric of the schemes is:

$$C_t = \frac{1}{r}C_u + rC_s$$

Let, C_t $(= \frac{1}{r}C_u + rC_s)$ be the total cost for the IS-41 scheme, and C'_t $(= \frac{1}{r}C'_u + rC'_s)$ be the total cost for the proposed scheme using forwarding pointers. For the proposed scheme to perform better than the IS-41 scheme, we require the following conditions to be true:

- **Condition 1**: $C'_t \leq C_t$, and
- **Condition 2**: $C'_s \leq \mathcal{R}C_s$.

If the proposed scheme satisfies both the conditions, the network load will be reduced without considerably increasing the call set-up time for the user. The value of R will depend on the quality of service of requirements of the user. We choose $\mathcal{R} = 2$.

For the sake of convenience, we divide the r space into three regions: *low* $(r < 0.1)$, *moderate* $(0.1 \leq r < 5)$ and *high* $(r \geq 5)$. We also divide the α space into two regions: *low* $(\alpha < 0.5)$, and *high* $(\alpha \geq 0.5)$. In our discussions, we will refer to the regions instead of actual values.

b. IS-41 Scheme

Let a mobile host mh move from a registration area to a new registration area, and, let the corresponding location servers be *old_ls* and *new_ls* respectively. Let the new mobile support station be *new_mss*. Let the home location server of mh be HLS. From the update scheme presented in Section D.1.a, and Fig. 13 we can obtain,

$$\begin{aligned} C_u \quad = \quad & \text{cost}(new\_mss \rightarrow new\_ls) + \text{cost}(new\_ls \rightarrow HLS) + \text{cost}(HLS \rightarrow new\_ls) \\ & + \text{cost}(new\_ls \rightarrow new\_mss) + \text{cost}(HLS \rightarrow old\_ls) + \text{cost}(old\_ls \rightarrow HLS) \end{aligned}$$

$$\begin{aligned} &= A + B + B + A + B + B \\ &= 2A + 4B \end{aligned}$$

Step 1 involves message transfer over the wireless link, and hence is not considered in the cost analysis. Steps 3 and 5 in Fig. 13 are database lookup and update operations, and do not incur any network load. Thus, they too are not considered in the cost analysis.

Let a mobile host *src* call another mobile host *dest*. Let the location servers of the registration areas in which *src* and *dest* are currently residing be *src_ls* and *dest_ls* respectively. Let the home location server of *dest* be *dest_HLS*. From Section D.1.b and Fig. 14, we can obtain,

$$\begin{aligned} C_s &= \text{cost}(src\_mss \rightarrow src\_ls) + \text{cost}(src\_ls \rightarrow dest\_HLS) \\ &\quad +\text{cost}(dest\_HLS \rightarrow dest\_ls) + \text{cost}(dest\_ls \rightarrow dest\_HLS) \\ &\quad +\text{cost}(HLS \rightarrow src\_ls) + \text{cost}(src\_ls \rightarrow src\_mss) \\ &= A + B + B + B + B + A \\ &= 2A + 4B \qquad (3.1) \end{aligned}$$

Step 1, 3, 5, 7 do not incur any network load. Thus, they are also not considered in the cost analysis.

c. Movement-based Heuristic

As stated earlier, the number of registration areas crossed by a host when the movement-based heuristic scheme performs an update for that host is $\mathcal{M}$. Thus, the number of forwarding pointers due to moves by the host when an update takes place for that host is $(\mathcal{M} - 1)$. Since the number of forwarding pointers for a host when an update takes place is $(\mathcal{M} - 1)$, $(\mathcal{M} - 1)F$ is the total cost of creating forwarding pointers before an update takes place for that host. Therefore the update cost per registration area crossing for the movement-based heuristic scheme is,

$$C'_u = \frac{1}{\mathcal{M}}((\mathcal{M}-1)F + C_u)$$

From Fig. 16 we can obtain the search cost for the movement-based scheme as follows:

$$\begin{aligned} C'_s &= \text{cost}(src\_mss \rightarrow src\_ls) + \text{cost}(src\_ls \rightarrow dest\_HLS) \\ &\quad +\text{cost}(dest\_HLS \rightarrow src\_ls) + \text{cost}(\text{traversing the} \\ &\quad \text{forwarding pointers to } curr\_ls) + \text{cost}(curr\_ls \rightarrow src\_ls) \\ &\quad +\text{cost}(src\_ls \rightarrow src\_mss) \end{aligned}$$

Step 1 in Fig. 16 involves message transfer over the wireless link, and hence is not considered in the cost analysis.

We make a conservative estimate of the cost $(curr\_ls \rightarrow src\_ls)$. In the worst case, this cost will be equal to kF (the cost of traversing the forwarding pointers). Therefore,

$$\begin{aligned} C'_s &= A + B + B + kF + kF + A \\ &= 2A + 2B + 2kF \end{aligned} \tag{3.2}$$

From equations (3.1) and (3.2), we get,

$$C'_s = A + \frac{C_s}{2} + 2kF$$

The average value of k is upper bounded by $(\mathcal{M}-1)/2$. Thus, the average search cost is as follows:

$$C'_s = A + \frac{C_s}{2} + (\mathcal{M}-1)F$$

We will now determine the relative total cost (C'_t/C_t) and relative search cost (C'_s/C_s).

The relative total cost is given as follows:

$$\begin{aligned}\frac{C_t'}{C_t} &= \frac{\frac{1}{r}C_u' + rC_s'}{\frac{1}{r}C_u + rC_s} \\ &= \frac{\frac{1}{r\mathcal{M}}(C_u + (\mathcal{M}-1)F) + r(A + \frac{1}{2}(C_s + 2(\mathcal{M}-1)F))}{\frac{1}{r}C_u + rC_s} \qquad (3.3)\end{aligned}$$

We assume that the cost B of HLS interactions is the dominant cost in C_s and C_u. This may not be true for all network architectures. This assumption was mainly motivated by the studies which indicate HLS to be the bottleneck in PCNs. The forwarding cost F is assumed to be a fraction of the cost of update and searches, i.e., $F = \alpha C_u$, where, $\alpha \leq 1$. Substituting $F = \alpha C_u$, $A \ll C_u$, and $C_s = C_u$ in (3.3) yields,

$$\frac{C_t'}{C_t} = \frac{1}{2\mathcal{M}}\left(\frac{(1+\mathcal{M}r^2)(1+2\alpha(\mathcal{M}-1))+1}{1+r^2}\right) \qquad (3.4)$$

The relative search cost is given as follows:

$$\frac{C_s'}{C_s} = \frac{1}{2}(1+2\alpha(\mathcal{M}-1)) \qquad (3.5)$$

Using the above equations we will now determine the values of $\mathcal{M}$ that satisfies the conditions stated in Section D.3.a. Condition 2 requires the search cost of the proposed scheme to be less than or equal to twice than the search cost of the IS-41 scheme. For this condition to be satisified, following should hold:

$$\mathcal{M} \leq \frac{3+2\alpha}{2\alpha}$$

Fig. 17 illustrates the variation of maximum allowable $\mathcal{M}$ with α. Note, that for low α values, $\mathcal{M} > 5$ is allowable. Figs. 18-20 illustrate the variation of relative total cost with r for different values of $\mathcal{M}$, for $\alpha = 0.2, 0.5$ and 0.8 respectively. It can be observed that at high values of r, Condition 1 is violated for high values of $\mathcal{M}$. In other words, long chain lengths are not suitable at high values of r. Thus, a small

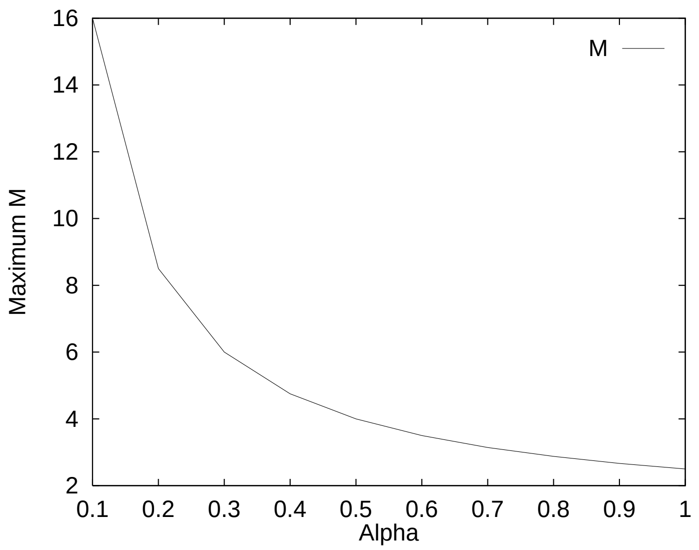


Fig. 17. Maximum Allowable Chain Length

$\mathcal{M}$ has to be chosen in order to obtain good performance for a wide range of r and α. We keep $\mathcal{M} < 5$ in our analysis. We analyze the performance of movement-based heuristic scheme for $\mathcal{M} = 2$, 3 and 4. Note that in Figs. 18-20, the relative total cost of the movement-based scheme increases as r and α increases. Longer chain lengths (i.e., larger $\mathcal{M}$) are preferred at low values of r. However, as r increases, schemes using smaller chain lengths (smaller $\mathcal{M}$) perform better. It can be seen in Fig. 19 and Fig. 20 that for high α values, the performance of movement-based heuristic with large $\mathcal{M}$ performs worse than the IS-41 scheme at high values of r.

Figs. 21-23 illustrate the variation of the relative search cost of the movement-based heuristic scheme with r for $\alpha = 0.2, 0.5$ and 0.8 respectively. It can be observed that the relative search cost of the movement-based heuristic scheme is independent of r. However, the relative search cost increases with α. At high values of α, Condition 2 is violated for large values $\mathcal{M}$.

Thus, it is recommended that the chain lengths be kept small (< 5). This is because, schemes with long chain lengths perform poorly in terms of search cost as

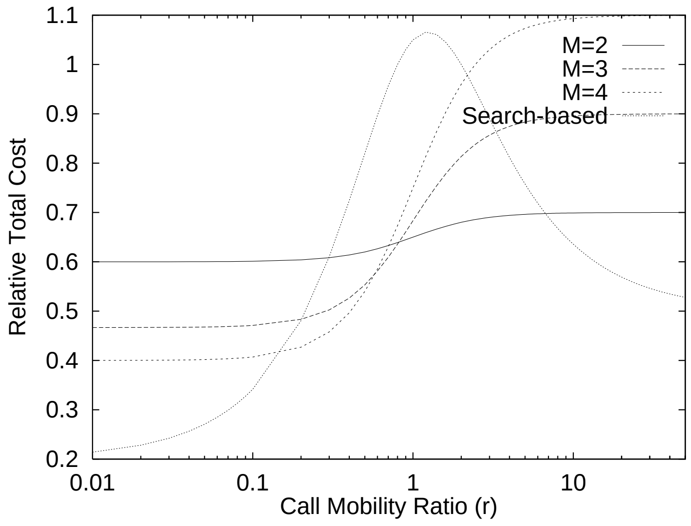


Fig. 18. Relative Total Costs: $\alpha = 0.2$

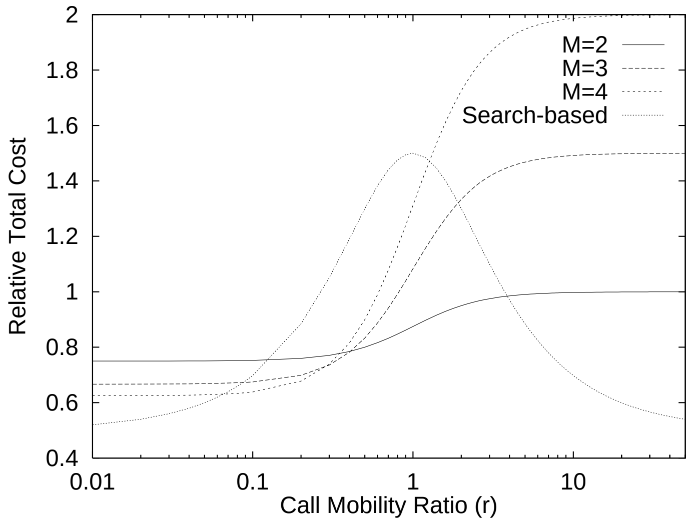


Fig. 19. Relative Total Costs: $\alpha = 0.5$

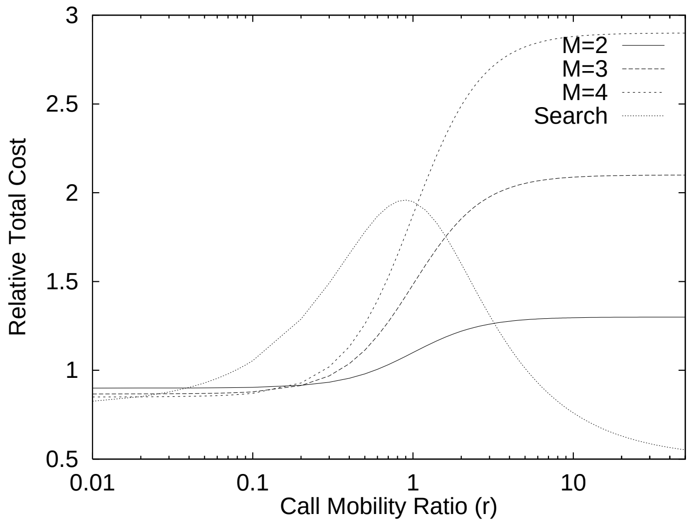


Fig. 20. Relative Total Costs: $\alpha = 0.8$

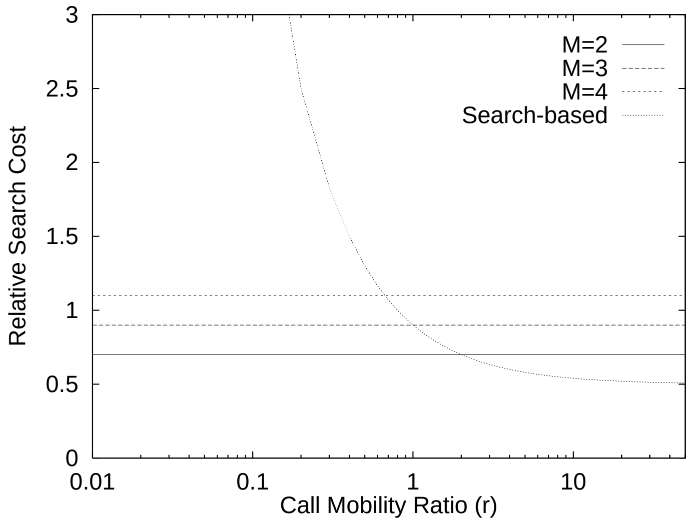


Fig. 21. Relative Search Costs: $\alpha = 0.2$

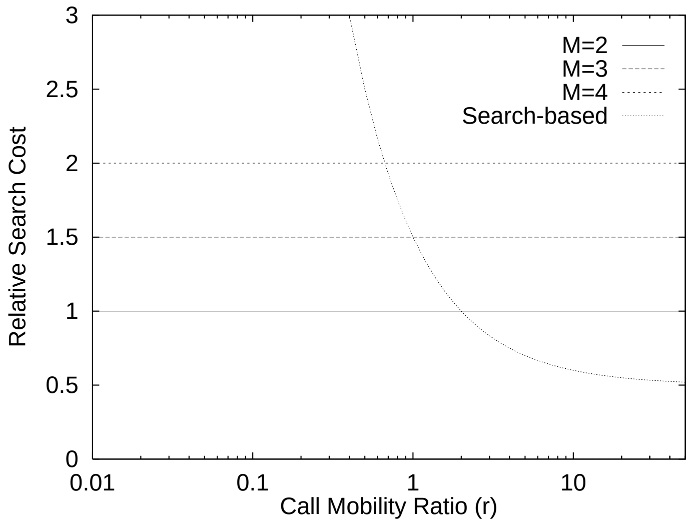


Fig. 22. Relative Search Costs: $\alpha = 0.5$

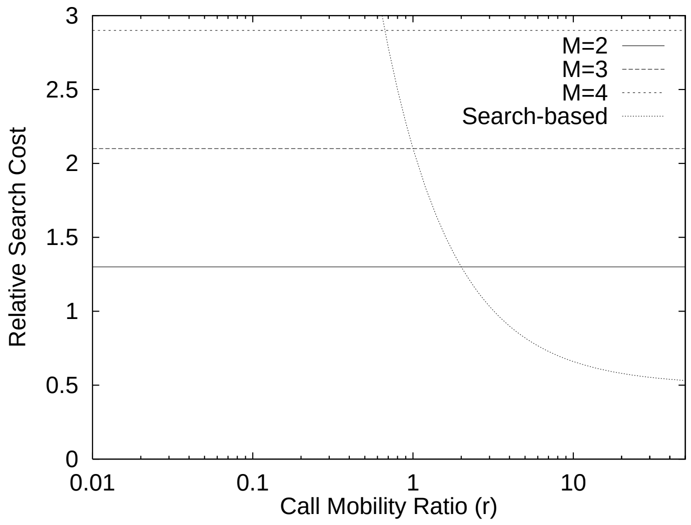


Fig. 23. Relative Search Costs: $\alpha = 0.8$

well as total cost for high values of α and r.

d. Search-based Heuristic

In this scheme, updates at the HLS occur once every $\mathcal{S}$ searches. Thus, the average number of forwarding pointers traversed during a search is $\frac{1}{\mathcal{S}}\sum_{i=0}^{\mathcal{S}}\frac{i}{r} = \frac{\mathcal{S}+1}{2r}$. The cost of update is only F (cost of setting forwarding pointer). The search cost for search-based heuristic scheme is,

$$C'_s = 2A + 2B + 2(\frac{\mathcal{S}+1}{2r})F = A + \frac{C_s}{2} + 2(\frac{\mathcal{S}+1}{2r})F$$

The update cost for search-based heuristic scheme is,

$$C'_u = F + \frac{r}{\mathcal{S}}C_u$$

Thus, the total cost is,

$$C'_t = \frac{1}{r}C'_u + rC'_s = \frac{1}{r}(F + \frac{r}{\mathcal{S}}C_u) + r(A + \frac{C_s}{2} + 2(\frac{\mathcal{S}+1}{2r})F) \tag{3.6}$$

After substitutions similar to that in Section D.3.c, the relative total cost and relative search cost for the search-based heuristic are as follows:

$$\frac{C'_t}{C_t} = \frac{2\mathcal{S}\alpha(1 + r(\mathcal{S}+1)) + r(2 + \mathcal{S}r)}{2\mathcal{S}(1+r^2)} \tag{3.7}$$

$$\frac{C'_s}{C_s} = \frac{1}{2r}(r + 2(\mathcal{S}+1)\alpha) \tag{3.8}$$

Using the above equations, we will now determine a 'good' value of $\mathcal{S}$ that satisfies the conditions stated in Section D.3.a. For Condition 2 to hold, the following should hold:

$$\mathcal{S} \leq \frac{3r - 2\alpha}{2}$$

Fig. 24 illustrates the variation of maximum allowable $\mathcal{S}$ with r for $\alpha = 0.2$. Note, that for low values of r, the maximum allowable $\mathcal{S}$ is less than 1. This suggests that HLS updates should take place more than once between two consecutive searches. The maximum allowable $\mathcal{S}$ increases with r. For our analysis, we fix $\mathcal{S} = 1$. This is because, we observed that the performance of search-based heuristic scheme with $\mathcal{S} = 1$ is quite close to that with 'optimal' $\mathcal{S}$, in terms of relative total cost, where the optimal $\mathcal{S}$ is the value of $\mathcal{S}$ that provides the minimum relative total cost.

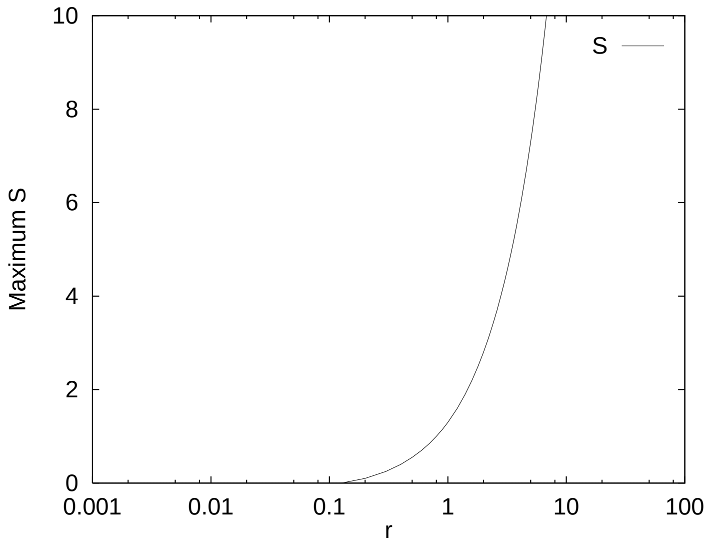


Fig. 24. Maximum Allowable Searches per HLS Update: $\alpha = 0.2$

The variation of the relative total cost with r for the search-based heuristic scheme is illustrated in Figs. 18-20. It can be observed that the relative total cost is very low for low values and r and high values of r. At low values of r, HLS updates occur very rarely (i.e., only during searches). Since the update cost is the dominating factor in the total cost at low values of r, the relative total cost of the search-based update heuristic is very low. On the other hand, HLS updates occur very often at high values of r. This causes the search cost to be very low. Since the search cost is the dominating factor in the total cost at high values of r, the relative total cost of the search-based heuristic is very low. However at moderate values of r, the performance of search-based heuristic scheme is worse than the IS-41 scheme.

Figs. 21-23 illustrate the variation of relative search costs with r for $\alpha = 0.2,\ 0.5$ and 0.8 respectively. It can be observed that the search cost is very high for low values of r. This is because for small r, the HLS updates occur infrequently, and the moves frequently. Thus, each search has to traverse a long chain of forwarding pointers. Moreover, there is a rapid decrease in search costs as r increases. This is because the length of chain of forwarding pointers reduces as calls arrive more often.

e. Observations

- For movement-based heuristic to perform well over a wide range of r and α, a small value of $\mathcal{M}$ should be chosen.

- The relative total cost of the search-based heuristic is much lower than the IS-41 scheme and the movement-based heuristic scheme for low and high values of r ($r < 0.1$ and $r > 5$). This is true for a wide range of α. However, the search cost of the search-based heuristic is unacceptably high for low values of r.

- For moderate values of r ($0.1 < r < 5$), the relative total cost of the movement-based heuristic scheme is lower than the IS-41 scheme and the search-based heuristic scheme. However, this is true only for low values of α (≤ 0.5). At high values of α, a small value $\mathcal{M}$ ($\mathcal{M} < 3$) should be chosen for movement-based heuristic scheme to perform well for moderate r. Also search cost of movement-based heuristic is unacceptably high for high values of α.

Thus, it can be concluded that although forwarding pointers do reduce the total network load, the search cost (i.e., *call set-up* time) increases. In the next section we present a search-update strategy to reduce the search cost.

E. A Search-Update Strategy

The search cost could be reduced by eliminating the cost of querying the *HLS* for the destination host. This can be achieved by having a location entry at the *src_ls* (in Fig. 16) pointing to the start of the chain of forwarding pointers. This is called a "search-update". A search-update occurs after a successful *search*, when the location server of the requesting host updates the location information corresponding to the mobile host. Search-updates might reduce the search cost (call set-up time). In this work, we will use a particular search-update strategy called *jump update*.

1. Jump Update

In *jump update* a location entry is created at the location server (*src_ls*) of the caller (*src*) for the destination host (*dest*) after a successful search. As stated earlier, for search-based heuristic, an update at the *HLS* take place only after a search. For *jump update*, in addition to updating the *HLS*, the location information of *dest* at *src_ls* is also updated. If there were no location entries for *dest* at *src_ls*, a location entry is created. The cost of *jump update* is the cost of setting a forwarding pointer. The motivation behind this kind of update is based on the assumption that *src* communicates frequently with *dest*. Therefore, the subsequent search by *src* for *dest* will potentially be lower.

2. Altered Search Procedure for Search-Updates

The search procedure needs to be altered to make use of the location information (if available) at the location server. If *dest* is not in the registration area of *src_ls*, check if *src_ls* has a location entry for *dest*. The location entry for *dest* at *src_ls* might be either the current location of *dest*, or the start of a chain of forwarding pointers to the current location of $dest$. Thus, when the location entry can be found in src_ls, a HLS query could be avoided. The chain of forwarding pointers is traversed to get to the current location of $dest$. Else, if the location entry for $dest$ is not found in src_ls, follow the previously explained search procedure (Section D.2.c).

3. Forwarding Pointer Maintenance for Search-Updates

In the forwarding pointer maintenance procedure described in Section D.2.d, we were not concerned about the stale forwarding pointers at the location servers, because all the searches (Section D.2.c) went to the HLS first. We ensure that the HLS does not have stale location information by updating the HLS at least once every T_{purge} units of time. The location servers also purge forwarding pointers that are older than T_{purge} units of time, thus, ensuring that stale forwarding pointers are not encountered during a search. However, if the search procedure of a host is altered as in Section E.2 it should be ensured that the forwarding pointers at the location server are not stale at any point of time.

A location information at a location server is purged or updated (when search-update scheme is used) in the following events:

- During search-updates.
- When an HLS update takes place.
- When a mobile host disconnects.

Search-updates: The timestamp of a forwarding pointer for the host is updated with the current time at the location server whenever a search-update for the host takes place at the location server.

HLS update: For the movement-based heuristic, HLS update takes place after $\mathcal{M}$ registration area crossings. Apart from the periodic purging (due to timeouts), we have to purge the forwarding pointers which become stale after an update at the HLS. This will bound the maximum length of forwarding pointers traversed during a search to $(\mathcal{M}-1)$. Such purging is possible, if the pointers are *bi-directional*, i.e., there are backward pointers in addition to the forwarding pointers. Therefore, after an HLS update, purge messages are sent to the location servers containing stale forwarding pointers for the host. These location servers are determined using the backward pointers. If forwarding pointers are purged after an HLS update, then the average number of pointers for a host in the network at any time is upper bounded by $(\mathcal{M}-1)+r\mathcal{M}$. The explanation is as follows: The maximum length of the chain before an update at the HLS is $(\mathcal{M}-1)$, i.e, there is a maximum of $(\mathcal{M}-1)$ location servers which have a forwarding pointer for a host due to moves before an HLS update takes place for the host. If search-updates are used, then each call for the host from a location server will lead to a forwarding pointer for the host at the location server. The average number of calls that can occur before an HLS update is $r\mathcal{M}$. Therefore, the average number of location servers that have forwarding pointers created due to search-updates is upper bounded by $r\mathcal{M}$. This is because there can be more than one call from the area served by a location server. Therefore, for the movement-based heuristic, the average number of purge messages sent during an HLS update is upper bounded by $((\mathcal{M}-1)+r\mathcal{M})$.

If search-based heuristic is used, both HLS update and search-update take place during a search. The average number of moves made by the user between two searches is $\frac{1}{r}$. Therefore, due to moves, the average number of location servers with forwarding pointers for the host is upper bounded by $\frac{1}{r}$. This is because the host may visit the same cell. In addition, there is at most one location server that has a forwarding pointer for the host due to the previous search-update. Therefore, for the search-based heuristic, the average number of purge messages sent during an HLS update is upper bounded by $(1 + \frac{1}{r})$.

Disconnection: Disconnections can also cause inconsistency in location information. If a host disconnects, its location information should be purged from all location servers. The HLS of the host will know the last location of the host before it disconnects (Appendix A). The HLS will also know the location of the host whenever it reconnects/switches ON (Appendix A). However, if the host switches ON at a location different from the location where it had earlier disconnected, the location servers which had forwarding pointers to the old location, have stale and incorrect information. Thus, if the altered search procedure (Section E.2) is used, we need to purge the stale and incorrect location information at these location servers. This will require purge messages to be sent to the location servers which have the forwarding pointers for the disconnected host. Subsequent search for the host initiated from these location servers will be forwarded to the HLS of the host. The HLS will not have incorrect location information of the host. Any search query for a disconnected host initiated from these location servers while the purge is in progress will lead to a location server ls_i which has no information of the host. The location server ls_i sends back an error message to the location server which initiated the search. Upon receiving an error message, the location server will purge its forwarding pointer, and forward the search query to the HLS of the host.

Forwarding pointer maintenance for search-updates require messages to be sent over the network. We include the purge costs in the update cost during our analysis of the search-update scheme. Since we are mainly concerned with the network load due to host movement, we do not consider the network load due to disconnections in our analysis.

4. Performance Analysis of Search-Update Scheme

In the analysis presented here, we assume that T_{purge} is large enough (as compared to the time period between two HLS updates due to heuristics) such that HLS updates due to timeout form a small fraction of all HLS updates. Therefore, we assume that updates at the HLS take place only upon searches (with search-based heuristic) or upon moves (with movement-based heuristic). Let:

- s = *Hit probability.* It is the probability that a forwarding pointer to the destination host exists at the location server of the caller's registration area.
- $C''_u, C''_s, C''_{su}, C''_t$ are the update, search, search-update and total cost using forwarding pointers and search-updates. $C''_t = \frac{1}{r}C''_u + r(C''_s + C''_{su})$.
- k' = For a caller host, this is the average number of forwarding pointers traversed before locating the destination host when search-updates are used.

a. Movement-based Heuristic and Search-Updates

The update cost for schemes using search-updates will also include the purge costs incurred due to forwarding pointer maintenance.

$$C''_u = C'_u + \frac{1}{\mathcal{M}}(\text{purge cost})$$

Purge cost is the cost of sending purge messages to all the location servers containing stale forwarding pointers for the host. As stated earlier, the average number of forwarding pointers for a host in the network when search-updates are used is upper bounded by $((\mathcal{M} - 1) + r\mathcal{M})$. If we assume that the cost of sending a purge message is equal to the cost of setting or traversing a forwarding pointer, then the average purge cost is upper bounded by $((\mathcal{M}-1)+r\mathcal{M})F$. Therefore,

$$C''_u = C'_u + \frac{F}{\mathcal{M}}((\mathcal{M}-1)+r\mathcal{M})$$

$$\begin{aligned}
C''_s &= s(\text{cost}(\textit{src\_mss} \rightarrow \textit{src\_ls}) + \text{cost}(\text{traversing the forwarding pointers from} \\
&\quad \textit{src\_ls} \text{ to } \textit{curr\_ls}) + \text{cost}(\textit{curr\_ls} \rightarrow \textit{src\_ls}) + \\
&\quad \text{cost}(\textit{src\_ls} \rightarrow \textit{src\_mss})) + (1-s)(C'_s) \\
&= s(A + k'F + k'F + A) + (1-s)(2A + 2B + 2k'F) \\
&= 2A + 2k'F + 2(1-s)B \\
&= C'_s - \left(s(\frac{C_s}{2} - A) + F((\mathcal{M}-1) - 2k')\right) \text{ where,}
\end{aligned}$$

C'_s is the search cost for the movement-based heuristic scheme without search-udpates. As stated earlier, the cost of search-update, C''_{su}, is the cost of sending a message to *src_ls* to create a location entry. Therefore, C''_{su} is equal to F.

$$\begin{aligned}
C''_t &= \frac{1}{r}C''_u + r(C''_s + C''_{su}) \\
&= \frac{1}{r}\left(C'_u + \frac{F}{\mathcal{M}}((\mathcal{M}-1)+r\mathcal{M})\right) + \\
&\quad r\left(C'_s - (s(\frac{C_s}{2} - A) + F((\mathcal{M}-1) - 2k')) + F\right)
\end{aligned}$$

We substitute $F = \alpha C_u$, $A \ll C_u$, $C_s = C_u$, $C'_t = \frac{1}{r}C'_u + rC'_s$. The expressions for $\frac{C''_t}{C_t}$ and $\frac{C''_s}{C_s}$ are as follows:

$$\frac{C''_t}{C_t} = \frac{C'_t}{C_t} - \frac{r}{1+r^2}\left(\frac{rs}{2} + r\alpha((\mathcal{M}-1) - 2k') - \frac{\alpha((r+1)\mathcal{M} - 1)}{r\mathcal{M}} - r\alpha)\right) \tag{3.9}$$

$$\frac{C''_s}{C_s} = \frac{C'_s}{C_s} - \left(\frac{s}{2} + \alpha(\mathcal{M} - 1 - 2k')\right) \tag{3.10}$$

For movement-based heuristic scheme with search-updates to be beneficial, we require $\frac{C_t''}{C_t} \leq \frac{C_t'}{C_t}$ in (3.9). The value of s for movement-based heuristic scheme with search-updates to be beneficial is given as,

$$s \geq 2\alpha \left(\frac{\mathcal{M}(r+1)-1}{\mathcal{M}r^2} - (\mathcal{M} - 2 - 2k') \right)$$

We determine the value of s and k' for different values of $\mathcal{M}$ using the markov state analysis presented in Appendix B. Using the s and k' value obtained from Appendix, we determine the relative total cost (in (3.9)) and the relative search cost (in (3.10)). Figs. 25-27 illustrate the variation of relative total cost with r for $\alpha = 0.2, 0.5$ and 0.8 respectively. Similarly, Figs. 28-30 illustrate the variation of relative search cost with r for $\alpha = 0.2, 0.5$ and 0.8 respectively. It can be observed that the relative search cost has considerably reduced as compared to the schemes without search-updates. Moreover, the relative search cost is less than $\mathcal{R}$ (equal to 2) for a wide range of α and r. However, for $r < 1.0$, the relative total cost is greater than the relative total cost for movement-based heuristic without search-updates. This is due to the additional purging costs.

As r increases, the hit probability (s) increases, thus decreasing the search cost. Therefore, for higher values of r, the relative total cost of movement-based heuristic using search-updates is much lower than the movement-based heuristic without search-updates.

It should be noted that for high α values, the movement-based heuristic using search-updates performs worse than the IS-41 scheme. This is due to the high purge costs involved.

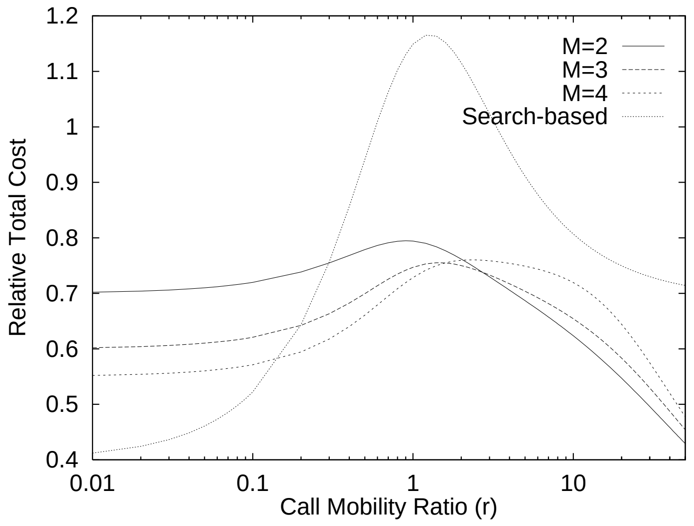


Fig. 25. Relative Total Costs (with Search Updates): $\alpha = 0.2$

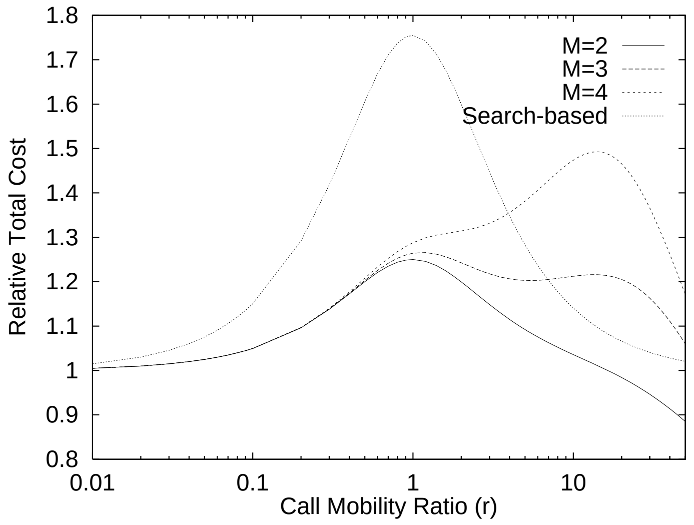


Fig. 26. Relative Total Costs (with Search Updates): $\alpha = 0.5$

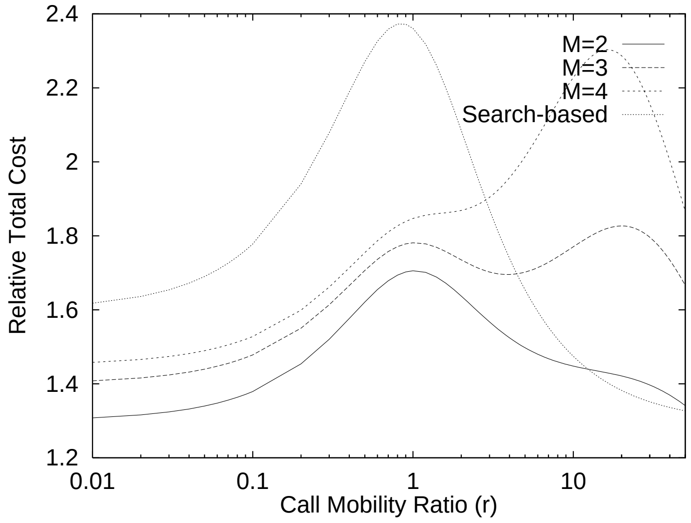


Fig. 27. Relative Total Costs (with Search Updates): $\alpha = 0.8$

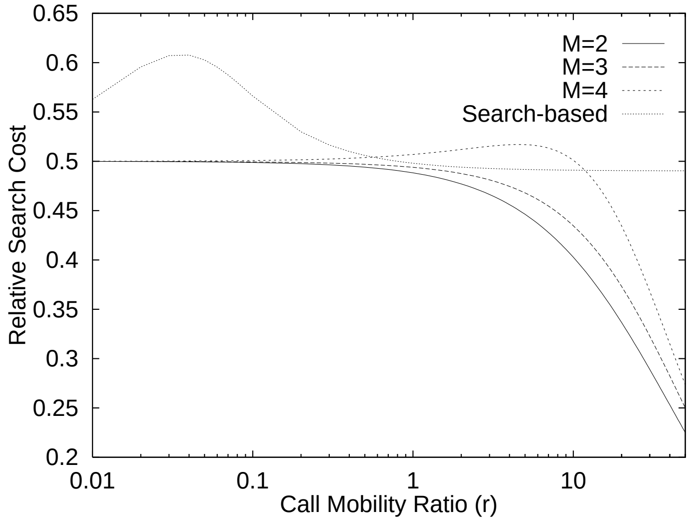


Fig. 28. Relative Search Costs (with Search Updates): $\alpha = 0.2$

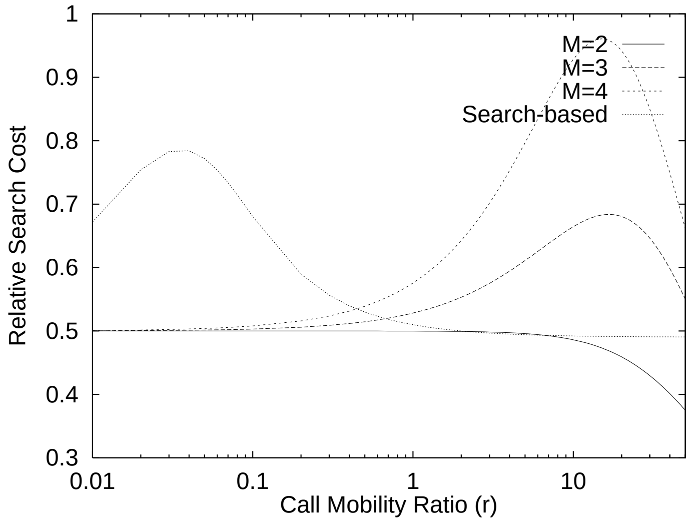


Fig. 29. Relative Search Costs (with Search Updates): $\alpha = 0.5$

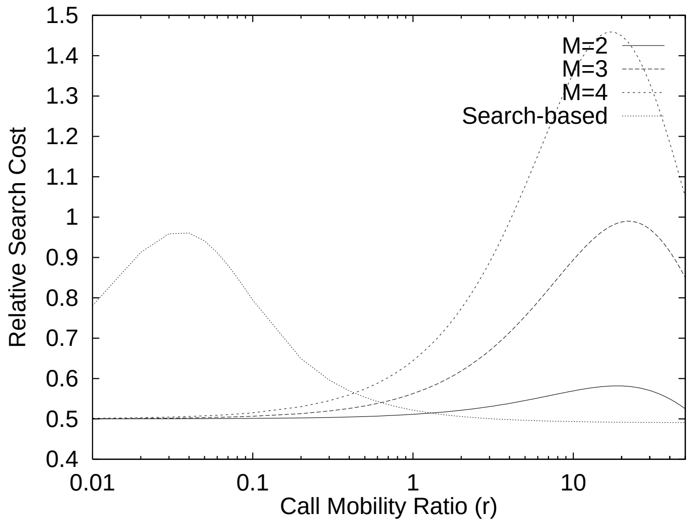


Fig. 30. Relative Search Costs (with Search Updates): $\alpha = 0.8$

b. Search-based Heuristic and Search-Updates

As stated earlier in Section D.2.d, we fix $\mathcal{S} = 1$. Similar to the movement-based heuristic, the update cost will also include the purge costs incurred due to forwarding pointer maintenance. As determined earlier in Section E.3, the upper bound on average purge cost for the search-based heuristic when search-updates are used is $(1+\frac{1}{r})F$. Therefore,

$$C''_u = C'_u + r\left(1 + \frac{1}{r}\right)F \text{ where,}$$

C'_u is the update cost for search-based heuristic scheme without search-updates. Similar to the movement-based heuristic scheme with search-updates, the search cost for the search-based heuristic with search-updates is,

$$C''_s = C'_s - \left(s\left(\frac{C_s}{2} - A\right) + 2F\left(\frac{1}{r} - k'\right)\right) \text{ where,}$$

C'_s is the search cost for search-based heuristic scheme without search-updates. The total cost C''_t is determined as,

$$\begin{aligned} C''_t &= C''_u + r(C''_s + C''_{su}) \\ &= C'_u + r(1+\frac{1}{r})F + r\left(C'_s - s(\frac{C_s}{2} - A) + 2F(\frac{1}{r} - k') + F\right) \end{aligned}$$

After substitutions, we determine $\frac{C''_t}{C_t}$ and $\frac{C''_s}{C_s}$ for the search-based heuristic scheme with search-updates as,

$$\frac{C''_t}{C_t} = \frac{C'_t}{C_t} + \frac{r}{1+r^2}\left(\frac{sr}{2} - \frac{\alpha(r+1)}{r} + 2r\alpha(\frac{1}{r} - k') - r\alpha\right) \qquad (3.11)$$

$$\frac{C''_s}{C_s} = \frac{C'_s}{C_s} - \left(\frac{s}{2} + 2\alpha(\frac{1}{r} - k')\right) \qquad (3.12)$$

For search-based heuristic scheme with search-updates to be beneficial, we require $\frac{C''_t}{C_t} \le \frac{C'_t}{C_t}$ in (3.11). The value of s for search-based heuristic scheme with search-updates to be beneficial is given as:

$$s \geq \frac{2\alpha(r+1)}{r^2} + 2\alpha\left(1 - \frac{2}{r} - 2k'\right)$$

We determine the value of s and k' using the markov state analysis presented in Appendix C. Using the s and k' value obtained from Appendix, we determine the relative total cost (in (3.11)) and the relative search cost (in (3.12)). As seen in Figs. 28-30, the relative search cost of the search-based heuristic scheme has considerably reduced upon using search-updates. This has happened at the expense of a small increase in the relative total cost (Figs. 25-27). It should however be noted that at high α values, search-updates are not suitable for search-based heuristic scheme. This is because of the high purging cost involved.

c. Observations for Search-Updates

In this section we have presented a scheme for search-update. We augment the forwarding strategy with this search-update scheme and analyze its performance. One of our concerns with forwarding was the increase in search cost. It is somewhat alleviated using search-updates; the search-based heuristic using search-updates has very low search costs for all values of r, and the movement-based heuristic using search-updates have low search costs at high α. However, this saving in search cost comes with an increase in the total cost. This increase in total cost becomes very significant at high values of α (relative forwarding cost). It is observed that both the heuristics using search-updates performs poorly compared to IS-41 at high values of α. Therefore, search-updates are not suitable at high values of α.

Fig. 31 presents a summary of the results. This figure has been repeated from Section A for convenience. As stated earlier, we divide the r space into three regions: *low* ($r < 0.1$), *moderate* ($0.1 \leq r < 5$) and *high* ($r \geq 5$). We divide the α space into two regions: *low* ($\alpha < 0.5$), and *high* ($alpha \geq 0.5$).

This figure gives us a fair idea about when forwarding pointers and search-updates are beneficial. If they are beneficial, the table shows which combination performs the best.

α r	Low	High
Low	Search-based with Search-Updates	Movement-based (M<5)
Moderate	Movement-based with Search-Updates (M<5)	Movement-based (M<3)
High		Search-based

Fig. 31. Performance Chart

F. Fault Tolerance

The use of forwarding pointers introduces the issue of corruption/omission of a forwarding pointer due to a transient failure at a location server. It is assumed that the protocol software is stored in some stable storage and hence does not get corrupted due to these failures. Thus, if a location server crashes and recovers, the location information (includes forwarding pointers) stored in the volatile memory may be initialized randomly.

Fault tolerance requires that regardless of the initial state, the system will eventually converge to the correct state [18, 19]. The correct state in the scope of location management is that the location servers have the correct location information of the hosts. To recover from forwarding pointer corruption due to a transient failure, the location servers have to periodically send the pointer information to other location servers reachable using the bi-directional pointers. An inconsistency can occur if a location server ls has a pointer for a host h to another location server ls_1, and the location information of h at ls_1 is corrupted by a transient failure. Thus, periodic exchange of information along the bi-directional pointers will ensure consistent forwarding pointer information to be stored in the location servers. Apart from the forwarding pointers, the host database (which stores the cell location of the host in its registration area) at the location server should also be fault tolerant. This will require periodic exchange of information between the MSSs in the registration area and the location server. An inconsistency can occur if according to the location server database, a host is located in more than one cell, or not located in any cell although the host is present in some MSS's cell in the registration area.

If the location server crashes and recovers at a later time, its database will be initialized to an inconsistent state with respect to the rest of the system. Thus, the system will have to eventually converge to the correct state. This can be ensured in the following manner. The failed location server, upon recovery, informs the MSSs in its registration area and the other neighboring location servers about its recovery. Upon receiving the recovery message from the location server, the other location servers check their forwarding pointer database for any forwarding pointers to/from the recovered location server. This is possible because we assume bi-directional pointers. The location servers will send the forwarding pointer information to the recovered location server, which in turn will update its forwarding pointer database. The MSSs upon receiving the recovery message from the location server send the list of hosts in their cell to the location server so that the location server can initialize the host database (which stores the cell location of the host in its registration area).

The table which represents the mapping of each mobile host to its home location server is assumed to be stored in the stable storage (e.g., disk) and thus is unaffected by any failures. It is a safe assumption because this table needs to be always error-free and consistent with rest of the system (which includes other location servers), and is also not updated as often as the forwarding pointer database or the host database.

G. Memory Overhead

A limitation of the forwarding pointer approach could be the memory overhead at the location servers. In this section we try to analyze the memory overhead of the forwarding pointers. To assist in our analysis, we classify the pointers into two types:

- Forwarding Pointers: A forwarding pointer for a host is created at a location server when the host goes out of the location server's registration area.
- Bi-directional Pointers: This is required if the network needs to be fault-tolerant, or if search-updates are going to be used. In this case, a pointer for a host is created at a location server when, (i) the host enters the location server's registration area, (ii) the host goes out of the location server's registration area, or (iii) there is a call originating from the location server for the host.

We use the following notations and data for our analysis. The data is obtained from [58]. Let:

- U = Total number of users in the network = 2.87 million.
- N = number of registration areas in the network = 128.
- ρ = mean density of the users = 390/sq.km.
- L = Registration area boundary length = 30.3 km.
- v = average speed of the user = 5.6 km/hr.

- λ = Rate of call origination and delivery = 1.4/hr/user.
- B_1 = Size of a forwarding pointer in bytes.
- B_2 = Size of a bi-directional pointer in bytes.
- t_p = purge time interval for the pointers.

A forwarding pointer has the following fields: (i) identity of the mobile user, (ii) identity of the next registration area (destination of the forwarding pointer), (iii) timestamp indicating the time of creation of the pointer, and (iv) purge time interval, t_p. We assume that we require (8+4) bytes to store the timestamp and t_p respectively. In our analysis, we assume t_p is same for all the pointers and is equal to 1 hour. The size of a forwarding pointer B_1 is given by $\lceil\frac{log_2(U)+log_2(N)}{8}\rceil + 12$. Replacing the values we get, $B_1 = 16$ bytes.

The rate of registration area crossing R is given as $\frac{\rho vL}{\pi}$ [75]. Replacing the values, we obtain $R = 5.85/s$. Let the maximum number of pointers that exist at a location server at any time be P. Let us consider forwarding pointers first. Since pointers are created only due to moves, $P = R * t_p$. The memory required for forwarding pointers is $M_1 = P * B_1 = 5.85 * 3600 * 16 \approx 337 Kbytes$.

Let us consider bi-directional pointers now. The bi-directional pointer will have the following fields: (i) identity of the mobile user, (ii) identity of the next registration area (destination of the forwarding pointer), (iii) identity of the previous registration area (source of the forwarding pointer), (iv) timestamp indicating the time of creation of the pointer, and (v) purge time interval, t_p. Thus, the size of a bi-directional pointer B_2 is given by $\lceil\frac{log_2(U)+2*log_2(N)}{8}\rceil + 12$. Replacing the values we get, $B_2 = 17$ bytes.

Bi-directional pointers are created due to moves in and out of the registration area, and also due to calls. Thus, $P = 2 * R * t_p + \frac{\lambda U}{N} * t_p$. The memory required for bi-directional pointers is $M_2 = P * B_2 = (2 * 5.85 * 3600 + 1.4 * 2.87 * 10^6/128) * 17 \approx 1250 Kbytes$.

The figures above indicate the memory overhead required to implement the forwarding pointer scheme is insignificant when compared to the savings in terms of network bandwidth.

H. Estimation of Call-Mobility Ratio r

As stated earlier, the performance of the proposed scheme depends on the call-mobility ratio (r) and the relative cost of forwarding pointers (α). High network load is encountered if forwarding strategies are applied in networks where α is high and users have low r. The value of α is network dependent and should be known to the system designer. However, user's call-mobility ratio is not known a priori to the system designer. Thus, it is important to estimate the call-mobility ratio of the users so that the proper forwarding strategy could be applied. Various techniques have been proposed in literature to estimate r. A detailed investigation of the algorithms for estimating r, namely, *running average*, and *reset-K* were presented in [36]. In Chapter IV [46], we propose algorithms to estimate r based on the history of call and mobility patterns. The basic assumption behind these algorithms are that past history of the user's call mobility patterns will reflect the behavior in future.

I. Comparison with Other Centralized Schemes

Our work presented in this chapter [49, 50] differs from other centralized schemes [36, 37] in the following respect:

- A new heuristic (search-based) is proposed in this work. In [37] only movement-based heuristic is discussed.

- This chapter studies the impact of combining forwarding and caching on the perfomance. We show that caching in addition to forwarding does improve performance over schemes that employ only forwarding.
- However, combining caching and forwarding brings about unique problems with forwarding pointer maintenance. In this chapter we propose schemes for forwarding pointer maintenance. We also account for the overhead incurred due to maintenance in our analysis. Forwarding pointer maintenance and its associated cost was not taken into account in [37].
- We propose schemes to make location management strategy robust in the presence of failures.
- To assist in maintenance and fault-tolerance, we employ bi-directional pointers instead of uni-directional pointers [37].

J. Summary

This chapter presents location management strategies using forwarding pointers and *search-updates* for network architectures with HLSs. Also presented are two heuristics to limit the length of the chain of forwarding pointers namely, movement-based heuristic and search-based heuristic. We analyze the performance of these heuristics, and compare them with the scheme used in the IS-41 standard. It is observed that using forwarding pointers is beneficial for most of the call-mobility ratios. However, the search cost of some of the schemes became very high and unacceptable for some parameter values. A scheme for search-update is presented to reduce the search costs. The performance of location strategies using forwarding pointers and search-updates are analyzed.

The main assumptions made in this chapter are: (1) HLS interaction cost is the dominant cost, and (2) the length of the chain is a non-decreasing function of the moves. Assumption (1) may not be true for all network architectures. This assumption is mainly motivated by the studies which indicate HLS to be the bottleneck in PCNs [36, 37, 58]. On the other hand, assumption (2) helps in determining the lower bounds for the performance of our schemes.

Using the results from our performance analysis, an appropriate forwarding and search-update strategy can be selected for the user based on the user's call-mobility ratio (r) and the network design parameter (α). This will result in lower network load enabling the network to support more mobile hosts than it would using the location management strategy proposed in IS-41 standard.

CHAPTER IV

DISTRIBUTED LOCATION MANAGEMENT

A. Introduction

As the cell size decreases, the number of calls transferred from one cell to another increases. This causes the signalling traffic to increase and spurs requirements for more efficient location management schemes. We believe that the signalling traffic generated by moving hosts in a distributed location management scheme will be lower than the current existing centralized algorithms (Chapter III). This improvement can be accomplished by storing the location information of a mobile host in various location servers. The network architecture consists of a hierarchy of location servers which are connected to themselves and to the base stations by a static network. Another advantage of hierarchical schemes is that the location address of a host can be permanent. This is a desirable feature for a user who want the same address (e.g., telephone number) associated with them for their entire life, irrespective of whether they change service providers or residence. However, in centralized schemes using *home location servers* (HLS), the user's address determines the HLS of the user. If a user changes service providers, there is a strong possibility that the HLS of the user will also change. Same will be true if the user shifts to a new residence. For example, if a user is a resident of Texas, the user's HLS will typically be somewhere in Texas. Now, if the user wishes to shift residence to New York, the user's HLS will no longer be in Texas for it will be cost prohibitive. Instead, the user will have a new HLS somewhere in New York. This will change the user's address. Thus, life-long addressing will not be possible in HLS based schemes without decoupling the HLS from service providers and geographical locations.

In this chapter we present several location management strategies based on a hierarchical tree structure database. These strategies try to satisfy the goal of providing efficient searches and updates. A location management strategy is a combination of a *search* strategy, an *update* strategy, and a *search-update* strategy. *Static* location management uses one fixed combination of *search*, *update* and *search-update* strategies. This chapter presents the results of simulations carried out to evaluate the performance of various static location management strategies for various call-mobility patterns.

If the system designer has a priori knowledge of the call-mobility pattern of the user, the strategy which performs best for the given values of call and mobility can be selected. However, this information is not always available. Thus, there is a need for *adaptive* location management. The basic philosophy behind adaptive management is that the past history of the system will reflect the behavior in the future, and hence by keeping track of the past history and modifying the management strategy accordingly, one expects to perform well for any call-mobility pattern. In this chapter we present preliminary ideas and results for adaptive location management.

Unlike in previous chapter, we resort to simulations to analyze the schemes presented in this chapter so that we can accomodate non-uniform call-mobility patterns. Real-life call-mobility traces are expected to be non-uniform and we show in this chapter that *adaptive* location management performs well in such conditions.

The review of related work in location management can be found in Chapter III. This chapter is organized as follows. Section B presents the static location management strategies, and Section C presents the simulation results for the various static location management strategies. Section D presents the adaptive location management scheme and summary is presented in Section E.

B. Static Location Management

A location management strategy will be a combination of a *search* strategy, an *update* strategy, and a *search-update* strategy. Fig. 32 illustrates the space of location management strategies discussed in this chapter. In this chapter, we are going to discuss location management strategies in the absence of a *home location server* (HLS).

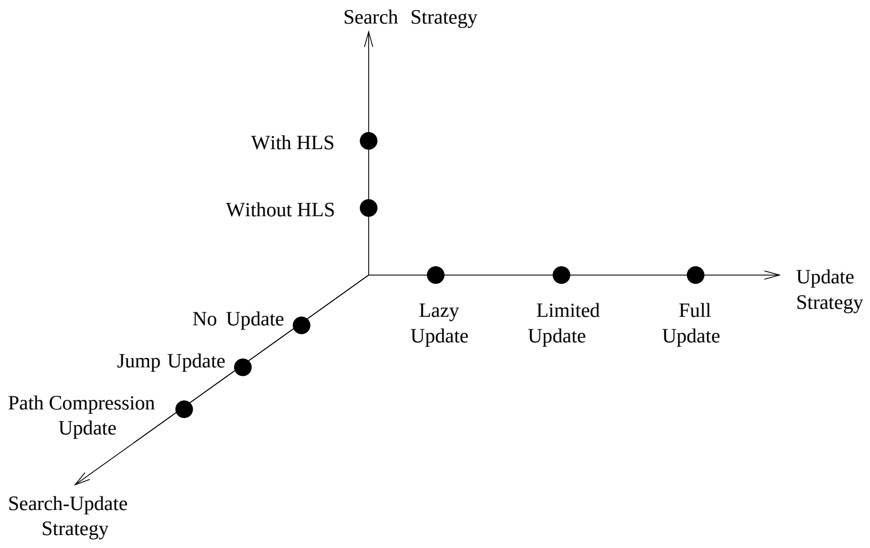


Fig. 32. Space of Location Management Strategies

1. Logical Network Architecture (LNA)

Mobile systems consist of mobile hosts, mobile support stations (base stations), and location servers. The logical network architecture (LNA) is a hierarchical structure (tree) consisting of mobile support stations and *location servers*[1]. As shown in Fig. 33, the mobile support stations (MSS) are located at the leaf level of the tree. Each MSS maintains information of the hosts residing in its cell. The other nodes in the tree structure are called location servers (LS). Each location server maintains

[1]Typically location servers correspond to the mobile switching centers.

information regarding mobile hosts residing in the subtree beneath it.

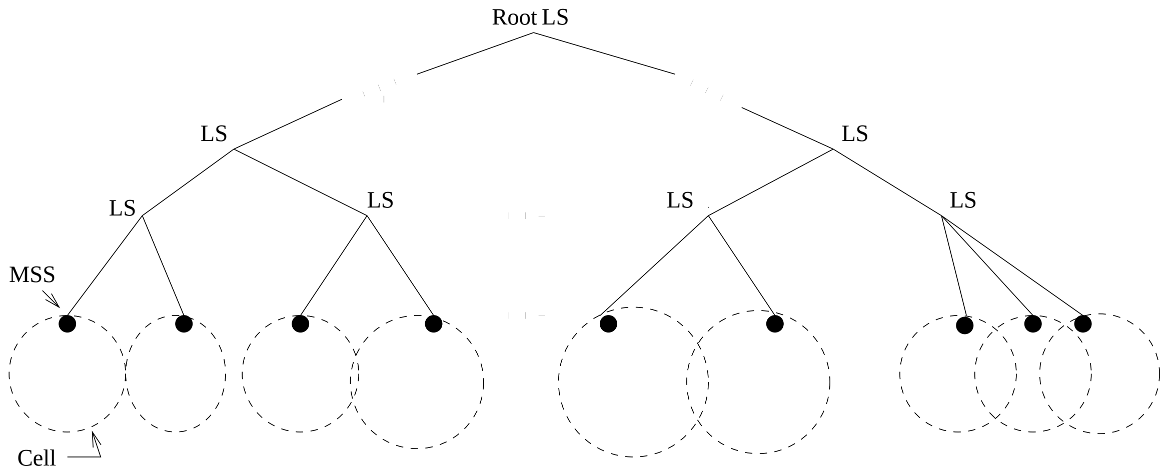


Fig. 33. Logical Network Architecture

2. Data Structures

There is an unique "home" address for every mobile host. The home address is the identifier or name of the mobile host. The "physical" addresses of a mobile host might change, but its home address remains the same, irrespective of the host's location [74, 78]. Each LS maintains an *address mapping* table that maps the home address to the physical address of the mobile hosts residing in the subtree beneath it. Thus, the problem of location management basically focuses on the management of the address mapping table.

There is a location entry in LS corresponding to a host h, if the host h is in one of the cells in the subtree of a location server LS. If the host h moves to a cell which is not in the subtree of LS, then the entry corresponding to h is updated (as explained later) at LS. All the nodes maintain location information using 3-tuples which have the following elements : (i) Mobile host identifier (id), (ii) Forwarding pointer destination (fp_dest), and, (iii) Time at which last forwarding pointer update took place (fp_time). Each location server maintains a 3-tuple for each mobile host residing in the subtree beneath it, and each mobile support station maintains a 3-tuple for each mobile host residing in its cell.

At the location servers and the MSS, forwarding pointer destination (fp_dest) is the location of the mobile host. At the MSS, the fp_time value for a host residing in its cell is $NULL$. Let us illustrate the use of forwarding pointers with an example. Let us suppose that we are using a strategy which uses forwarding pointers for location updates. Let a host h reside initially in cell c. The MSS of the cell c will have an entry (h,c, $NULL$). Let there be a location server L which maintains information of the hosts residing in cell c. There will be an entry (h, c, t_l) corresponding to host h at L, where t_l is the local time at L at which the entry was recorded at L. Let host h move to a new cell c'. Let t be the local time at the MSS of cell c at which the change of location of h is recorded at the MSS. Let t' be the local time at L at which the change of location of h is recorded at L. Thus, the location information of h will be (h, c', t') at L, and, (h, c', t) at MSS of cell c.

A Note: The above data structures contain fp_time field to store time. The fp_time entry for a data structure on a node, say v, contains the local time at node v when the data structure was last modified. It should be noted that the correctness of the algorithms does not require the clocks at various nodes to be tightly synchronized.

3. Initial Conditions

It is assumed that, initially, the location information of the mobile hosts is stored in the corresponding location servers, i.e., each location server (LS) should have the correct location information for all the hosts residing in the cells in its subtree. Thus, the root location server should have the correct location information of all the hosts in the system. Let us illustrate this with an example. In Fig. 34, nodes 1-7 are location servers, and 8-15 are mobile support stations. There are two mobile hosts $h1$ and $h2$. In the initial state, host $h1$ is in cell 8, and $h2$ is in cell 12. Initially, the correct location information of host $h1$ will be available at the location servers {4,2,1}. Likewise, the location information of $h2$ will be available at the location servers {6,3,1}. Thus, the location information of a host is available at all the location servers located on the path from its current MSS to the root.

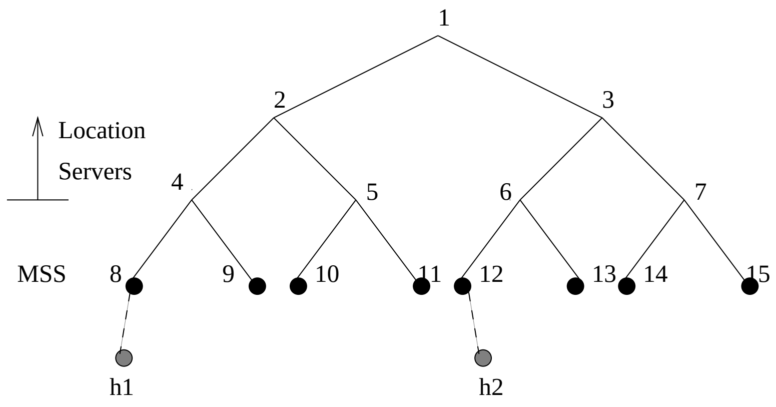


Fig. 34. An Example of Location Information Maintenance

4. Update Protocols

The strategies for updating the location information at the location servers and the mobile support stations, when the host moves[2], are as follows.

Let src and $dest$ be the identifiers of the source and destination cells, respectively. Let h be the identifier of the mobile host. Let t be the local time at the node at which the change of location of h is recorded at the node. i.e., for example, t is the local time at the MSS of $dest$ at which a location entry is recorded at the MSS.

[2]A *move* occurs when the host crosses cell boundary.

a. Lazy Updates (LU)

This is the simplest update scheme. Updates take place only at the MSS of the source and destination cells. A forwarding pointer is kept at the source MSS. The updated entry at the source MSS becomes $(h, dest, t)$. An entry for host h, $(h, dest, NULL)$ is added at the destination MSS. The location information at the location servers are not updated. The cost of update is zero, because there are no update messages being sent.

b. Full Updates (FU)

Upon a move, apart from the MSSs involved (i.e., the MSS of the source and destination cells), location updates take place in all the LSs located on the path from the MSS of the source and destination cells to the root. The scheme and an example follows.

Source cell:

1. At the MSS : For host h, set $fp\_dest = dest$, and $fp\_time = t$. The updated entry for host h at the MSS becomes $(h, dest, t)$.

2. All location servers on the path from src to the root : The MSS of src sends update message to these location servers. Upon receipt of the update message, the location servers update the entry for h to $(h, dest, t_l)$, where t_l is the local time at the location server.

Destination cell:

1. At the MSS : An entry $(h, dest, NULL)$ is added for host h. If there was an old entry for h, it is overwritten by this new entry. There can be only one entry

per host in the MSS and the LS.

2. All location servers on the path from $dest$ to the root : The MSS of $dest$ sends update message to these location servers. Upon receipt of the update message, the location servers create an entry $(h, dest, t_l)$, where t_l is the local time at the location server. If there was an old entry, it is overwritten by this new entry.

Therefore, in an H-level tree, the update cost per move is $2(H-1)$, where, the cost metric is the number of messages. Let us illustrate this scheme with an example. Suppose in Fig. 34, host $h1$ moves from 8 to 14. Forwarding pointer to 14 will be kept at MSS 8. MSS 8 sends update message to {4,2,1}, and these location servers also maintain forwarding pointer to 14. An entry for $h1$ will be made at MSS 14. MSS 14 sends update message to the location servers {7,3,1}, and these location servers also make an entry for host $h1$.

c. Limited Updates (LMU)

Update in the location information takes place at a limited number of levels of location servers in the tree. Here updates occur at $m(< H)$ lower levels of location servers on the path to the root. Updates at these location servers are similar to the FU scheme. The location servers at levels higher than m are not updated. Thus, the update cost per move is $2m$. Let us illustrate this scheme with an example. Let the value of m be chosen to be 1. Suppose in Fig. 34, host $h1$ moves from cell 8 to cell 14. Forwarding pointer to 14 will be kept at MSS 8. MSS 8 sends an update message to {4}, and 4 maintains forwarding pointer to 14. An entry for $h1$ will be made at MSS 14. MSS 14 sends an update message to {7}, and 7 makes an entry for host $h1$.

5. Search Protocol

If a host h in cell C wants to communicate with another host h', h has to know the location of h'. This requires that host h search for host h'. As stated earlier, we do not make explicit use of *home location server* (HLS) for searches. The search process in the absence of a HLS is as follows. If the mobile support station of C has no location information for h', it forwards the location query to the next higher level location server on the path to the root. If that location server does not have any location information for h', it again forwards the location query to the next higher level location server on the path to the root. This process repeats until a location server which has location information for h' is reached. In this process if the root location server is reached, and the root also does not have the location information for h', then the root broadcasts to find out the location of the host h'. In our schemes we make sure that the root has location information (as explained later), so the broadcast will not be necessary. Once the location information (cell identifier) for h' is obtained, the location query is forwarded to the MSS of the cell. Host h' is either in the cell of MSS, or, MSS has a forwarding pointer corresponding to h'. If host h' is in the cell of MSS, the search is complete. Else, a chain of forwarding pointers is traversed till the MSS containing the host h' is reached. The search protocol is as shown in Fig. 35.

6. Search-Update Protocols

The idea of manipulating forwarding pointers upon a successful search was earlier suggested in [21]. It was used to track objects in a decentralized object oriented computer system. Location management becomes more efficient if the location updates also take place after a successful search. For example, suppose there is a host h that

```
Initially, the $search\_cell$ is $C$.
Step 1 : If the $MSS$ of the $search\_cell$ has an entry for $h'$,
        If $fp\_dest = search\_cell$,
              host $h'$ is in the $search\_cell$. Search for $h'$ is complete.
        Else $search\_cell = fp\_dest$. Repeat step 1.
      Else forward the query to the next higher level location server
      on the path to the root.
Step 2 : If the location server has an entry $(h', fp\_dest, fp\_time)$ for $h'$
        $search\_cell = fp\_dest$. Go to step 1.
      Else If the location server is the root
        Root broadcasts to find out location of $h'$. The location server that has
        an entry $(h', fp\_dest, fp\_time)$ forwards the location query to $fp\_dest$.
        Set $search\_cell = fp\_dest$. Go to step 1.
      Else Forward the query to the next higher level location server
      on the path to the root.
      Go to Step 2.
```

Fig. 35. Search Protocol

frequently calls h'. It makes sense to update the location information of h' after a successful search, so that in the future if h calls again, the search cost is likely to reduce. The location information update takes place at the MSS of the caller. Let host h be the caller, and host h' be the destination host. Let the location of h and h' be C and C' respectively. Following are the strategies to update location information upon a search.

a. No Update (NU)

In this strategy, there are no location updates. But, the *fp_time* field of the entry corresponding to h' at the MSSs on the search path are updated to the current time at the MSS. The cost is zero. This is because the update of the time field could be done during the search process itself, and no additional messages need to be sent for this purpose. The update in *fp_time* is done to avoid *purging* of the forwarding pointer data at the MSSs. The *purge* protocol is explained in the next section.

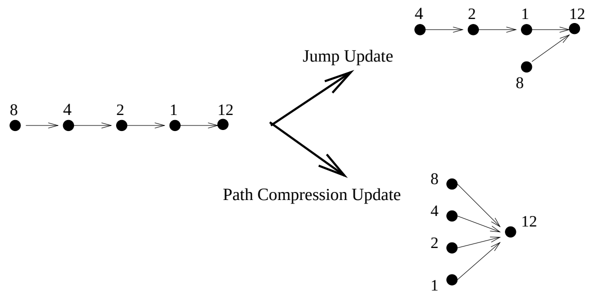


Fig. 36. Search Updates

b. Jump Update (JU)

In this strategy, a location update takes place only at the caller's MSS, i.e., MSS of the cell C. The entry for h' at the MSS of cell C is set to (h', C', t), where t is the local time at the MSS when the location information is updated. Let us illustrate with an example. In Fig. 34, let host $h1$ call host $h2$. Suppose the location information of $h2$ is available only at the location servers {6,3,1}. Using the search protocol described previously, the search path will be $8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 12$ (as shown in Fig. 36). During a jump update [21] following the search, the location information at 8 is updated. Thus, 8 jumps from wherever it is in the forwarding path to having the current location information (as shown in Fig. 36). The update cost is 1. This is because only one message needs to be sent from the MSS of C' notifying the location information of host h'.

c. Path Compression Update (PCU)

In this strategy, upon a successful search, a location update takes place at all the nodes in the search path. All the location servers on the search path have the entry of h' updated to (h', C', t), where t is the local time at the location server when the location information is updated. All the MSSs on the search path including the caller's MSS have an entry of h' updated to (h', C', t), where t is the local time at the MSS when the location information is updated. Let us illustrate with an example. In Fig. 34, let host $h1$ call host $h2$. Suppose the location information of $h2$ is available only at the location servers {6,3,1}. As shown in Fig. 36, the search path will be $8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 12$. During a path compression update [21] following the search, location updates take place at location servers {4,2,1}, and MSS 8. Thus, all the nodes in the search path have the current location information (as shown in Fig. 36). The update cost is the length of the search path, which in this example is 4.

7. Purging of Forwarding Pointers

We need to periodically purge the stale forwarding pointers at the location servers and the mobile support stations. This should be done in order to (i) save storage space at the nodes, and (ii) avoid storing stale location information. It does not make any sense to keep the forwarding pointer information for a host h, if no other host is going to query this location server for the location information of h. We use a design parameter *purge interval* (PI) to decide whether to purge a forwarding pointer information or not.

The storage requirement at the location servers for forwarding pointers is a linear function of the purge time interval. For a typical personal communications network, it was determined that the memory required at the first level (the parents of the leaf level nodes) location servers for PI equal to 1 hour, was about 340 Kbytes when no search updates were used, and, about 840 Kbytes when jump updates were used [46].

Let the current time be $curr\_time$. If $fp\_time \neq NULL$, and $curr\_time - fp\_time \geq PI$, then the entry for the host is purged from the MSS [3]. If $curr\_time - fp\_time < PI$, it means that there is some other host in the system which has recently used the forwarding pointer information of i.

In the location servers, if $curr\_time - fp\_time \geq PI$ for a host, the location entry for the host is purged.

a. Updating of Forwarding Pointers with a Purge

When LU and LMU strategies are used, the forwarding pointers at higher level location servers do not get updated, and become stale. Thus, these forwarding pointers get purged periodically. However, some of the searches for the host might reach the higher levels. If the location servers at the higher levels do not have the information of the host, the root has to broadcast to find out the location. To avoid this, the

[3]Note that the fp_time value for a host residing in the cell will be $NULL$. So we are considering hosts which are currently not residing in the MSS's cell and whose forwarding pointer information is stored at the MSS.

forwarding pointers at the location servers on the path to the root from the current MSS must be updated periodically along with purging. This is achieved by the current MSS of each mobile host by sending a location update message to the location servers on the path to the root.

C. Simulations

A trade-off exists between the cost of updates (upon moves and searches) and cost of searches. The parameters that affect this trade-off are (i) call frequency, and (ii) mobility. In this chapter we will evaluate the effects of mobility and call frequency on the cost of updates, search-updates and searches. As stated earlier, the location management strategy is a combination of a *search strategy*, an *update strategy* and a *search-update strategy*. The search protocol is the same for all location management strategies. A total of 9 *static* location strategies are obtained using above strategies for *updates* and *search-updates*. We performed simulations to analyze the performance of the proposed location management strategies for various call frequency and mobility values. The location management strategies simulated were obtained by choosing one update strategy (say XX, where XX = LU, FU or LMU) and one search-update strategy (say YY, where YY = NU, JU or PCU). The location management strategy thus obtained is denoted as XX-YY.

1. Model

We assume a binary tree as the logical network architecture for the simulations. The height of the tree is H. The number of location servers in the network is $2^{(H-1)} - 1$, and the number of mobile support stations (or the number of cells) is $2^{(H-1)}$. Physical proximity of the cells under the same location server is assumed. This will help in determining *short* and *long* moves. The height of the tree H was chosen to be 10 for the simulations[4]. Thus, there were 512 cells in the network.

The main aim of the chapter is to develop protocols for efficient searches and updates, i.e., reduce the number of messages due to location updates, without increasing the number of messages required for searches. We assume that the message delays are negligible. Since, message delays are small compared to the time between calls or moves, performance of the schemes will not be significantly affected by this assumption.

Simulations were performed for two types of environments : (i) *arbitrary* moves and *arbitrary* callers, (ii) *short* moves and a *set of callers*. In type (i), the user can move to any location (cell), and, get calls from any other host in the network. This is not necessarily true in real life, but it gives a fair idea of the performance of the location management schemes in such extreme conditions. Type (ii) is closer to real life mobile environments. Users are expected to make a lot of *short* moves to nearby destinations, and are expected to receive calls from a specific set of callers (e.g. family, business colleagues)[5].

a. Call and Mobility Distribution for Type (i)

The time between moves of a host is assumed to follow an exponential distribution with a mean M. The destination cell is chosen randomly among the 512 cells. The time between calls for a host is assumed to follow an exponential distribution with a

[4]In existing networks like GSM or Internet, the height will be 3 to 4. Since a binary tree was assumed for the simulations, we needed to have higher number of levels to have a sizeable number of cells in the network. However, similar performance trends are expected for other networks.

[5]The callers are assumed to be immobile. They are either part of the static network, or, do not leave their cell.

mean C. The caller's cell is chosen randomly from among the 512 cells.

b. Call and Mobility Distribution for Type (ii)

Type (ii) consists of generating calls from a specific set of callers and short moves. One option to generate short moves is to put an upper limit on the length of the move, in terms of the *difference* between the source and destination cell identifiers, and randomly vary the length of the move within the upper limit. For example, in Fig. 34, if we keep an upper limit of 1, the host $h2$ will be able to move to any cell in the set $\{11,12,13\}$. But, our logical network architecture just assumes proximity of cells which are under the same location server. Thus, a move from $12 \rightarrow 11$ is not equivalent to a move from $12 \rightarrow 13$.

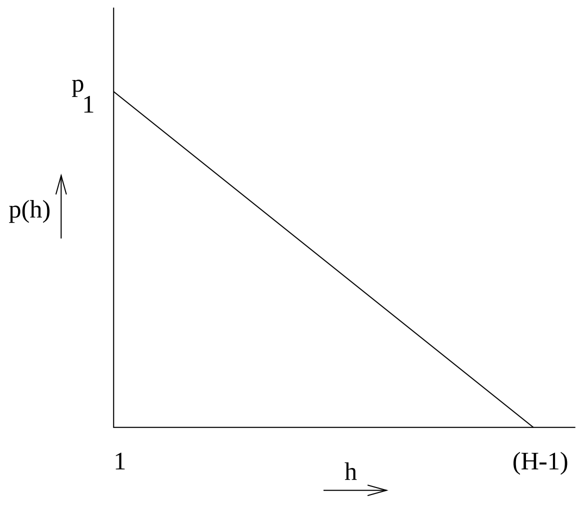


Fig. 37. Probability Distribution Function in Terms of Height

We use a different approach to characterize *short* moves. We randomly choose the number of levels of location servers where an update would have occurred due to the move, if FU update strategy were to be used. The number of levels can be between from 1 to $(H-1)$. Level 0 is the MSS level. Smaller the number of levels chosen, shorter is the length of the move. The probability distribution function of the length of the move in terms of height (number of levels) is shown in Fig. 37.

$$p(h) = \frac{2}{(H-1)(H-2)} * (H - 1 - h).$$

The cumulative distribution function (*cdf*) is as follows: $cdf(h) = \sum_{x=1}^{h} p(x)$. We randomly chose a height h based on the given probability distribution function. The number of choices for the destination cell is 2^h. Let the identifier of the current cell (i.e., the source cell) be *curr*. Knowing the height h and *curr*, one can easily determine the ancestor of *curr* at level h in the binary tree. Let it be ls. Knowing ls, the set of destination cells possible is $\{ls * 2^h, ls * 2^h + 1, ..., ls * 2^h + 2^h - 1\}$. A destination cell is chosen randomly from this set. Let us illustrate with an example. In Fig. 34, for host $h2$, let the h obtained randomly be 2. Thus the number of choices is 4. The location server at level 2 is 3. Thus, the destination cell is randomly chosen from {12,13,14,15}. This is in coherence with the assumption of proximity of cells under the same location server. It should be noted that when the destination cell is same as the current cell, there are no location updates. The time between moves of a host is assumed to follow an exponential distribution with a mean M.

In type (ii), for each mobile host, callers were chosen from a specific set of cells. The size of the set was chosen to be 20. The set was chosen arbitrarily, and were not necessarily neighboring cells. The calls always originate from those cells. The time between calls for a host is assumed to follow an exponential distribution with a mean C.

c. Purge

Purge is performed periodically every PI units of time. The value of PI (*purge interval*) was chosen to be 10 units of time. We simulate the purge operation (described in Section B.7), however, we do not consider the cost due to purging in our analysis.

2. Cost Model

The cost metric is the number of messages required for each operation (search, update, and search-update). Thus, the cost of an update is the number of location servers which update the location information of the host. The cost of a search is the number of location servers and mobile support stations visited before locating the host. Cost of a search-update is the number of location servers which update the location information of the host.

The performance parameter of interest is the total cost, defined as the sum of average update cost, average search cost, and the average search-update cost. Other cost metrics are also possible (For example, see Chapter III.D.3.a).

3. Results

Simulations were performed to analyze the performance of the various location management strategies. Results were obtained for the two type of environments, Type (i) and (ii). The values of C and M were both varied from 1 to 15 units of time. Value of C was changed to vary the time interval between two successive calls. Value of M was changed to vary the mobility of the host. For example, $C = 1$ and $M = 1$ characterizes a communication intensive and ultra-mobile environment.

Type(i) : It was observed that the *LU-PC* strategy outperforms all the other strategies for all values of M and C. Therefore, we have only plotted the curves for *LU-PC*. The strategies using *FU* and *LMU* suffered due to the high cost of updates upon each move. *LU-NU* strategy suffered due to very high search costs. Because the callers were arbitrary, *LU-JU* strategy did not perform well as the update upon a successful search was not helping in reducing the search cost. Fig. 38 plots the total cost for the *LU-PC* strategy as a function of C for different values of M. As seen in the figure,

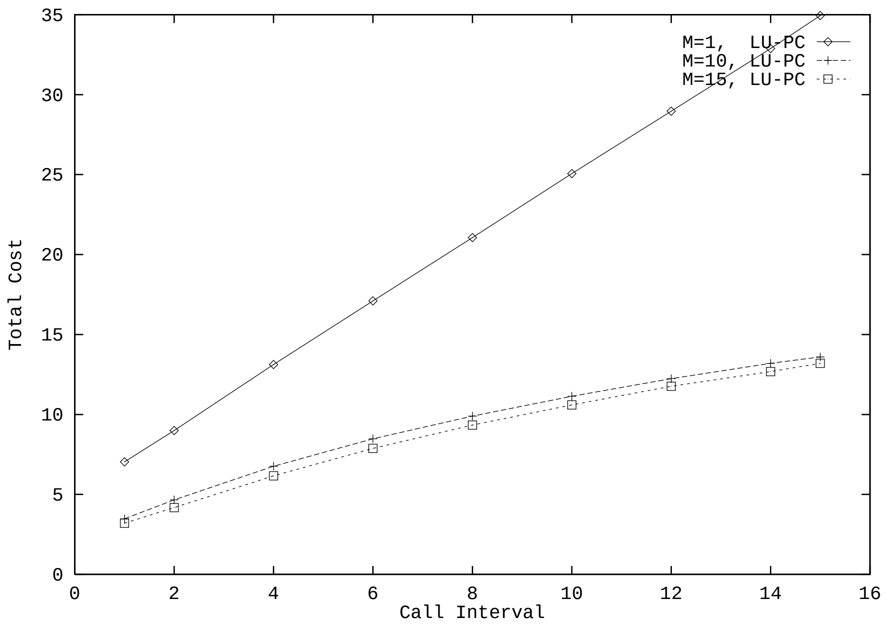


Fig. 38. Performance of *LU-PC* for Type(i)

the total cost increases with increasing C, and decreases with increasing M. This is because as C increases, the calls become infrequent, and the hosts might have moved to new locations, requiring new searches. Thus the reduction in search cost by path compression is not much effective. We also observe that the rise in total cost with C is higher for lower values of M. Lower the value of M, higher is the mobility, and thus the search cost will be higher. At high values of M, the difference in the total costs due to different values of M is low. This is because as M increases, the host movement reduces. Beyond a point, increasing M does not affect the total costs, and the curves converge to a single curve.

Type(ii) : It was observed that the *LU-PC* and the *LU-JU* strategies outperformed all the other strategies for all values of M and C. In contrast to Type (i) scenario, *LU-JU* performed well, because, there is a specific set of callers. Thus, the jump update

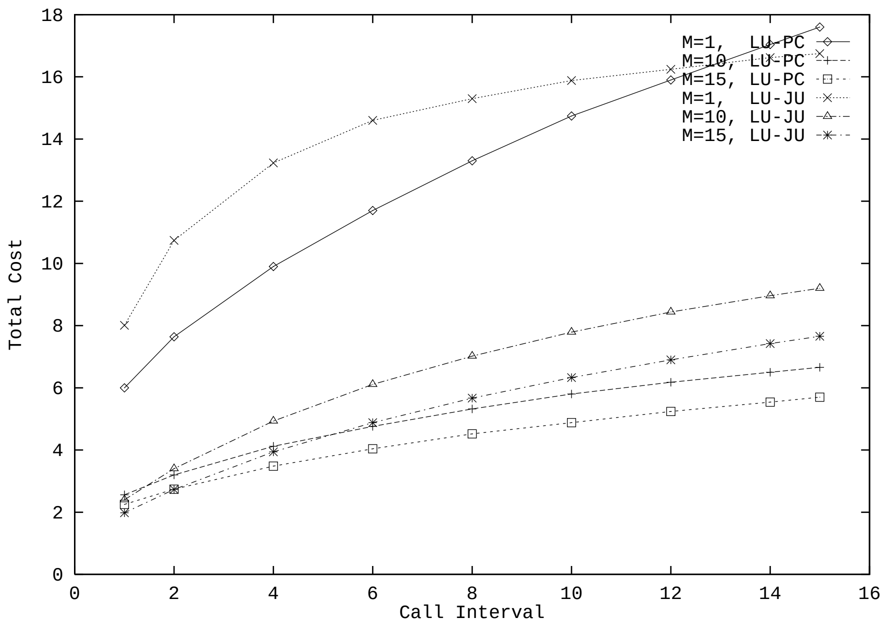


Fig. 39. Comparison of *LU-PC* and *LU-JU* for Type(ii)

at the caller is much more effective in reducing the search cost, because the caller is going to call the host again with a higher probability than in Type (i) environment. Fig. 39 plots the total cost for the *LU-JU* strategy and the *LU-PC* strategy as a function of C for different values of M. As seen, *LU-JU* performs better than *LU-PC* in high-communication and low-mobility, and, low-communication and high-mobility environments. In these environments, the search cost for *LU-PC* and *LU-JU* are comparable. Since the search-update cost is same as the search cost for *LU-PC*, the total cost for *LU-PC* is simply twice the search cost. Whereas, the average search-update cost for *LU-JU* is equal to 1. Thus, the total cost of *LU-JU* is lower than *LU-PC*. *LU-PC* performs better for other values of M and C because the search cost for *LU-JU* becomes large compared to *LU-PC*. Fig. 40 demonstrates the average search cost for the *LU-JU* strategy and the *LU-PC* strategy as a function of C for different values of M. As seen, *LU-PC* has a much lower search cost than *LU-JU*.

The search cost of *LU-JU* is slightly lower than *LU-PC* for high-communication and low-mobility environment.

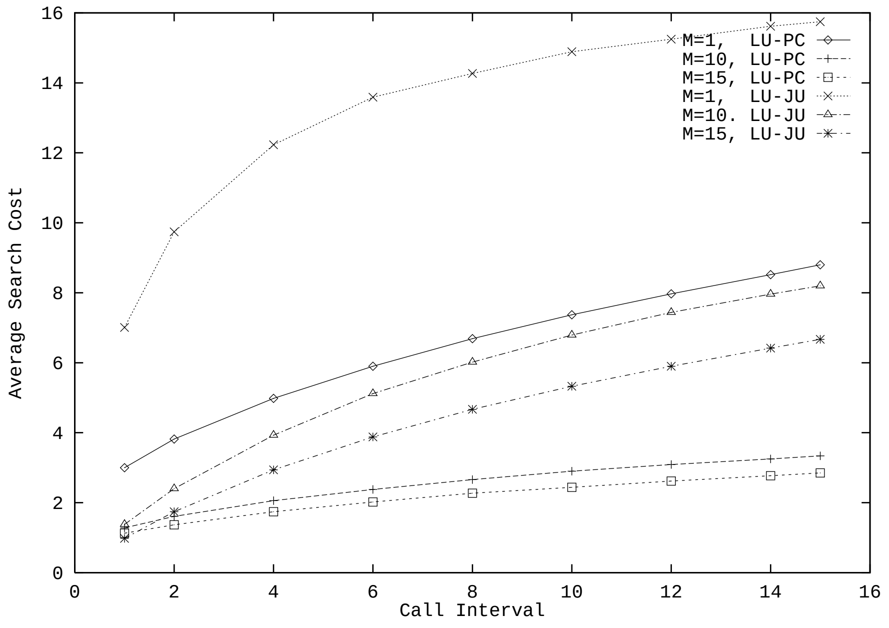
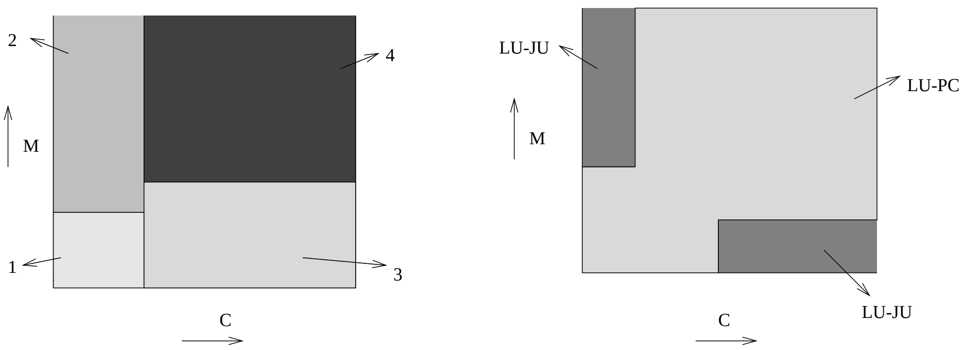


Fig. 40. Comparison of Search Costs of *LU-PC* and *LU-JU* for Type(ii)

4. Discussion

It was noticed that performing search-updates significantly reduced the search and total costs. For the logical network architecture assumed, it is seen that the *LU-PC* strategy performs better than the other strategies for most of the values of C and M. It is expected that *LU-PC* will perform well in other network models too. For other cost models, we expect the other proposed strategies to perform well, and sometimes better than the *LU-PC* strategy for some values of M and C. As shown in Fig. 41a, we expect zones in the M-C plane, where one scheme will outperform others for the call frequency and mobility values in the zone. This was evident in the Type (ii) environment. As shown in Fig. 41b, the M-C plane is divided in two zones, *LU-JU* and *LU-PC*. Thus, if the behavior of the mobile hosts (call frequency, mobility) is known a priori, the designer can obtain such an *M-C* chart and decide which location strategy will best suit the system.



(a) Generic Scenario

(b) Type (ii) Environment

Fig. 41. Partitioning of the *M-C* Plane

In the next section we will present some preliminary ideas and results for adaptive location management.

D. Adaptive Location Management

The system designer does not always have prior knowledge of the call-mobility pattern of the hosts. In these cases, one would require a location management scheme that can dynamically change the update and search-update strategy, such that the overall overhead incurred due to updates and searches is minimized. At the same time, we would not want to use up the battery power of the mobile hosts to determine the appropriate strategy dynamically. We require the MSS to take up the responsibility.

1. Data Structures

Let τ be the current time at the mobile host h. $M(h)$ is the sequence of moves of the host h. $M(h) = \{m_1, m_2, ..., m_n\}$, where, $m_1 = (t_1, src, dest)$, i.e., element m_1 is a move by the host h at time t_1 from src to $dest$, and $t_1 < t_2 ... < t_n$. (The time of move is observed at the mobile host h.) Each element of the set $M(h)$, m_i, contains two identifiers – the source cell identifier, and the destination cell identifier. If both identifiers are the same, then the host has not left the cell. This kind of entry is not necessary (hence will not be present), because it does not affect the location database. But if the identifiers are different, the source cell should determine whether the move is *long* or *short*.

$C_u(h)$ is the sequence of costs incurred due to updates upon the moves $M(h)$. $C_u(h) = \{c_{u1}, c_{u2}, ..., c_{un}\}$, where c_{uj} = cost of update upon a move m_j.

If another host h' wants to communicate with h, and if h is not in the same cell or if the MSS of h' does not know the cell identifier of h, h' has to search for h. A set $S(h)$ is maintained at the current MSS of h. $S(h) = \{s_1, s_2, ..., s_n\}$, where $s_i = (t_{si}, h')$; i.e., there was a call from h' for h at time t_{si}, and $t_{s1} < t_{s2} ... < t_{sn}$. Again, the time of call is observed at the mobile host h.

$C_s(h)$ is the sequence of costs incurred due to the searches $S(h)$. $C_s(h) = \{c_{s1}, c_{s2}, ..., c_{sn}\}$, where c_{sj} = cost of search s_j. $C_{su}(h)$ is the sequence of costs incurred due to search-updates upon searches $S(h)$. $C_{su}(h) = \{c_{su_1}, c_{su_2}, ..., c_{su_n}\}$, where c_{su_j} = cost of search-update upon the search s_j.

The data structures are obtained as explained in the next section.

2. Basic Idea

The above data structures are stored at the current MSS of the host. They get transferred to the new MSS during handoff. The decision of the type of updates and search-updates are done by the current MSS. The current MSS uses the data structures to determine the best suited strategy. The appropriate update and search-update strategy will be one of the proposed static location management update and search-update strategies.

It is assumed that the mobile host h knows the identifier of the cell it is currently residing in. When a host h moves, h sends a message (containing the identifier of its old cell, and the time of move) to the new MSS. The new MSS forwards a copy of this message to the old MSS. The move is recorded as a new element m_j in the sequence $M(h)$. The old MSS takes a local decision (explained later) regarding the updates. The cost of the update is recorded as a new element c_{uj} in the sequence C_u. The new MSS requests the old MSS for the data structures corresponding to h. If the new MSS makes any updates, the cost of the update is added to c_{uj} in C_u.

When a host h' wants to communicate with h, and if h is not in the same cell or if the MSS of h' does not know the identifier of the cell of h, h' has to search for h. A location query message is sent during the search. This message has a field to store the search cost. At any time, the search cost field indicates the cost incurred due to the search till now. The search cost gets incremented as the location query message is forwarded to a new location server or a mobile support station. Once h is located, a new element s_j is added to the sequence $S(h)$ at the MSS of h. The time of the call is the time observed at the mobile host h. The search cost is recorded as a new element c_{sj} to $C_s(h)$. The MSS decides upon the appropriate search-update strategy. It is determined based on the call history (explained later). For example, if a host h' frequently calls host h, it makes sense to use JU to reduce the subsequent search cost for h'. The cost of the search-update is recorded as a new element c_{su_j} to $C_{su}(h)$ at the MSS.

3. Mobility and Call Frequency

a. Determining Mobility

Let at time $t = \tau$, $M(h) = \{m_1, m_2, ..., m_n\}$, where $m_n = (t_n, src, dest)$, and $t_n \leq \tau$. Thus, m_n describes the move of host h that took place at time t_n from a cell src to a cell $dest$. Thus, the average time interval between successive moves $\triangle t_{avg} = \frac{t_n - t_1}{n}$.

We assume a system parameter *maximum threshold move interval* ($MTMI$). If there are no moves by the host for $MTMI$ amount of time, the host can be declared to be immobile or stationary. The sets $M(h)$ and $C_u(h)$ maintained at the current MSS are stale because the history does not reflect the behavior in future anymore. Therefore, they are deleted. In the absence of $M(h)$ set, the host is assumed to have a high mobility upon the first move.

We have defined two degrees of mobility – (i) low mobility, and (ii) high mobility. At any time τ, let t_n be the time of the last move by the host. If $\triangle t_{avg} < MTMI$, the host has a high mobility, else if $\triangle t_{avg} \geq MTMI$, the host has a low mobility.

b. Determining Call Frequency

Let at time $t = \tau$, $S(h) = \{s_1, s_2, ..., s_n\}$, where $s_n = (t_{sn}, h')$, and $t_{sn} \leq \tau$. s_n describes the call for host h from h' that took place at time t_{sn}. We define an average time interval between calls for **each** caller to host h. The average time interval between successive calls of caller h', $\triangle t_{savg}[h'] = \frac{\sum_{i=1}^{n'} \triangle t_{si}}{n'}$, where, n' is the number of calls made by h', and the $\triangle t_{si}$'s are the time intervals between two consecutive calls made by host h'.

We assume a system parameter *maximum threshold call interval* ($MTCI$). If there are no calls by host h' for $MTCI$ amount of time, the host h' can be declared to have no communication with h. The elements corresponding to host h' in the set

$S(h)$ are stale because the history does not reflect the behavior in future anymore. Therefore, they are deleted. In the absence of $S(h)$ set, the caller h' is assumed to be a frequent caller upon the first call of h' to host h, i.e., $\triangle t_{s1} = 0$.

Similar to mobility, based on the degree of call frequency, we have two types of caller – (i) non-frequent caller, and (ii) frequent caller. Then, if $\triangle t_{savg}[h'] < MTCI$, the caller is a frequent caller, else if $\triangle t_{avg}[h'] \geq MTCI$, the caller is a non-frequent caller.

c. Size of Data Structures

The maximum size n of the move set $M(h)$ and search set $S(h)$ can be chosen as a design parameter. Larger the value of n, better will be the learning of the host behavior, and thus a better predictability will be attained. However, the storage capacity available at the MSS restricts the value of n. The MSS has to maintain these sets for each mobile host in its cell. Thus, larger the value of n, larger is the storage cost.

4. An Example

In this section we will present an example algorithm for adaptive location management. It is for the network model assumed for static location management strategies. The knowledge of Fig. 41b, and the fact that *LU-PC* is the best scheme for long moves, will prove to be useful in dynamically determining the best strategy. From the previous section, we have the techniques to classify the moves, calls and the mobility of the host. If a host has a lot of frequent callers, the host is being frequently searched, else, if a host has a lot of non-frequent callers, the host is not frequently searched. The algorithm is as shown in Fig. 42.

We present an example where a simple algorithm *adaptive* as shown in Fig. 42

```
Algorithm adaptive
if (host makes a lot of long moves)
    Employ LU-PC.
else if ((frequently searched) and (low mobility))
    Employ LU-JU.
else if ((frequently searched) and (high mobility))
    Employ LU-PC.
else if ((Not frequently searched) and (high mobility))
    Employ LU-JU.
else Employ LU-PC.
```

Fig. 42. *adaptive* - An Adaptive Location Management Algorithm

performs better than the *static* location management strategies. Simulations were performed for type (ii) environment. As stated earlier, a mobile host makes a lot of short moves in type (ii) environment. Thus, the adaptive location management algorithm *adaptive* makes a choice between *LU-JU* and *LU-PC* based on call frequency and mobility of the host. Fig. 43 illustrates the mobility distribution of an user. The x-axis represents the time at which the user moves, and the y-axis represents the length of the move. Fig. 44 illustrates the incoming call distribution for the user. The x-axis represents the time at which the call is made for the user, and y-axis represents the distance of the caller from the user. The value of $MTCI$ and $MTMI$ was chosen to be 10 units of time. For this non-uniform call and mobility distribution, we evaluated the *LU-PC*, *LU-JU* and *adaptive* strategies. We define the *aggregate cost* at time t as the sum of update cost, search cost, and the search-update cost for the event that occurs at time t. Figs. 45-47 illustrate the aggregate

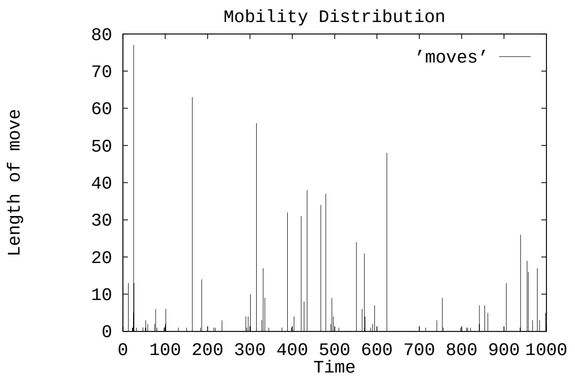


Fig. 43. Mobility Distribution

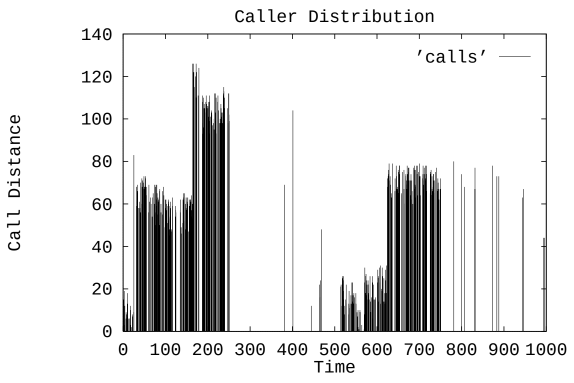


Fig. 44. Call Distribution

cost for *LU-JU*, *LU-PC* and *adaptive* strategies. For the given call and mobility distribution, results were obtained for different sizes of the move and call sets. It was observed that the minimum size of the move and call sets that was required for good performance of *adaptive* strategy was 7. Fig. 48 illustrates the difference of aggregate cost between *adaptive* and *LU-JU* schemes. Fig. 49 illustrates the difference of aggregate cost between *adaptive* and *LU-PC*. In Figs. 48-49, negative difference implies that *adaptive* is better. As seen in Fig. 48 and Fig. 49, *LU-JU* performs poorly during periods of high-communication, and *LU-PC* performs poorly during periods of low-communication. However, on the average, *adaptive* performs better than both the schemes during periods of low and high communication, as illustrated in Table 1. Time interval 100.0-200.0 is the high communication period (107 calls or 1.07 calls per unit time). During this period, if the system designer uses *LU-JU* instead of *adaptive*, the network load (in terms of number of messages) will increase by 33%. Time interval 400.0-600.0 is the low communication period (91 calls or 0.45 calls per unit time). During this period, if the system designer uses *LU-PC* instead of *adaptive*, the network load (in terms of number of messages) will increase by 12%. For the given call and mobility distribution (shown in Fig. 44 and Fig. 43), the total savings of *adaptive* over *LU-PC* is 4% and over *LU-JU* is 17% (as shown in Table III). Thus, the results show that a simple adaptive location management algorithm as shown in Fig. 42 performs better than the *static* location management strategies for a wide range of call-mobility patterns.

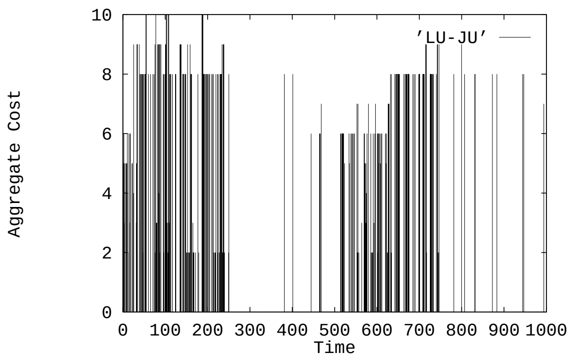


Fig. 45. Aggregate Cost for *LU-JU*

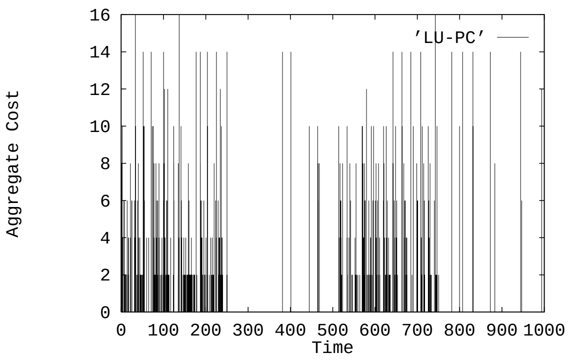


Fig. 46. Aggregate Cost for *LU-PC*

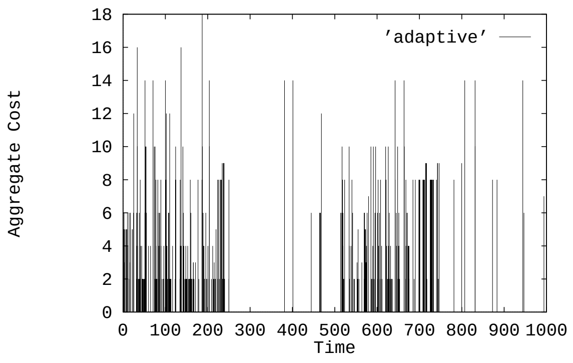


Fig. 47. Aggregate Cost for *adaptive* - $n = 7$

E. Summary

Presented in this chapter are strategies for updates, search-updates, and a search protocol for a hierarchical network architecture. A location management strategy is

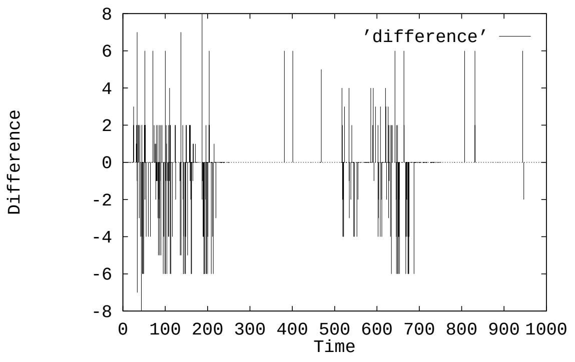


Fig. 48. ($adaptive - LU\text{-}JU$)

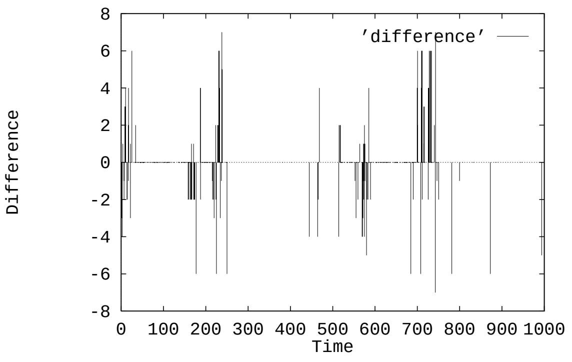


Fig. 49. ($adaptive - LU\text{-}PC$)

Table III. Comparison of Average Costs for Non-Uniform Distribution

Interval	# Calls	*LU-PC*	*LU-JU*	*adaptive*	Savings over *LU-PC*	Savings over *LU-JU*
100-200	107	3.32	4.08	3.1	6%	33%
400-600	91	3.36	3.02	3.0	12%	1%
0-1000	562	3.35	3.73	3.2	4%	17%

a combination of the search strategy, a update strategy, and a search-update strategy. Simulations were carried out to evaluate the performance of the various location management strategies. It was noticed that performing search-updates significantly reduced total costs. For the logical network architecture assumed, it is seen that the *LU-PC* (combination lazy updates and path compression search-update) strategy performs better than the other strategies for most call-mobility patterns. It is expected that *LU-PC* will perform well in other network models too.

Static location management uses one combination of search, update and search-update strategies throughout the execution. In order to obtain good performance using static location management, the system designer should a priori have a fair idea of the call-mobility pattern of the users. The host behavior (call frequency, mobility) is not always available to the system designer. Thus, there is a need for an adaptive location management. In this chapter we present preliminary ideas for adaptive location management. The basic philosophy behind adaptive management is that the past history of the system will reflect the behavior in the future and hence by keeping track of the past history and modifying the management strategy accordingly, one expects to perform well for any call-mobility pattern. Simulation results show that the performance of adaptive location management can be better than static location management. Adaptive location management results in lower network load, enabling the network to support more mobile hosts than it would using static location management.

CHAPTER V

ROUTING IN DYNAMIC NETWORKS

A. Introduction

A *dynamic* network composes of a set of mobile hosts that can communicate with each other over the wireless links (direct or indirect) without any static network interaction. Example of such networks are *ad-hoc* networks [17, 39, 61], and packet radio networks [16, 40, 42].

An important issue in dynamic networks is the design and analysis of routing schemes. This chapter investigates the consequence of mobility and disconnections of mobile hosts on the design and performance of routing protocol in a dynamic network.

In the existing proposals for *infrastructure* wireless networks, routing information of each mobile host is maintained in some database (*HLR* and *VLR* in IS-41 [46, 58], *home agent* and *foreign agent* in mobile IP [32, 62]) which is located in the static network. However, there is no such database available for *dynamic* networks. Due to limited range of the wireless transreceivers, a mobile host can communicate with another host only within a limited geographical region around it. Thus, it may be necessary for a mobile host to require the aid of other mobile hosts in forwarding data packets to its destination. The routing information will thus be maintained at the mobile hosts to assist in forwarding packets to other hosts. The problem here is the complexity of updating the routing information in such a dynamic network. Let us illustrate it with an example.

Example A.1: Fig. 50(a) is an example of a dynamic network. Routing information is maintained at each host. For example, routing table at $MH4$ requires packets destined for $MH1$ to be forwarded to $MH2$, which in turn will forward the packets to

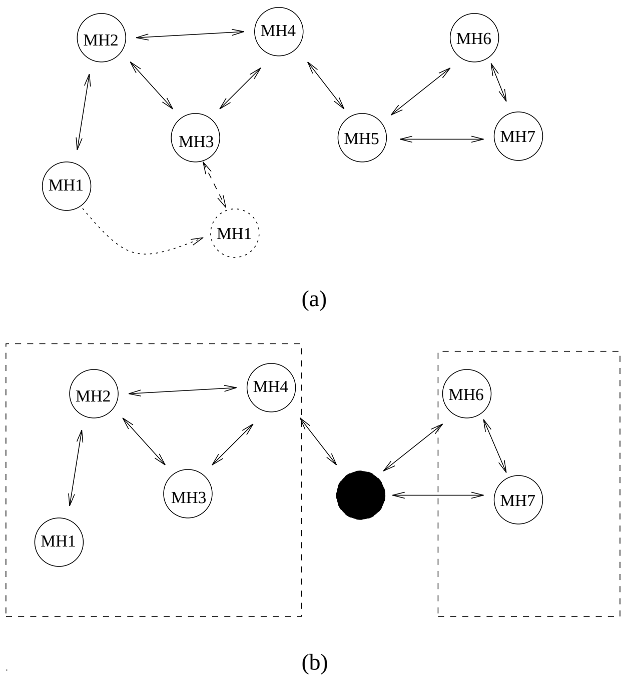


Fig. 50. A Dynamic Network

$MH1$. However, due to the movement of host $MH1$, the network topology changes. The communication link between $MH2$ and $MH1$ breaks, and, there is a new link between $MH3$ and $MH1$. Thus, the routing tables at the hosts have to be updated to indicate this change in topology. For example, the routing table at $MH4$ has to be updated to indicate that the packets destined to $MH1$ have to be forwarded to $MH3$ and not $MH2$. The network topology also changes due to host disconnections. As illustrated in Fig. 50(b), the network gets partitioned due to the disconnection of host $MH5$. Thus, the routing information at the hosts have to updated to indicate the change in topology during disconnections too. □

1. Previous Work

Numerous routing protocols have been proposed in the recent years. One of the most popular techniques for routing in communication networks is via distributed algorithms for finding shortest paths in weighted graphs [24, 35, 69, 55]. These distributed algorithms differ in the way the routing tables at each host are constructed, maintained and updated. The primary attributes for any routing protocol are :

- Simplicity : Simple protocols are preferred for implementation in operational networks [61].
- Loop-free : At any moment, the paths implied from the routing tables of all hosts taken together should not have loops. Looping of data packets results in considerable overhead.
- Convergence characteristics : Time required to converge to new routes after a topology change should not be high. Quick convergence is possible by requiring the nodes to frequently broadcast the updates in the routing tables.
- Storage overhead : Memory overhead incurred due to the storage of the routing information should be low.

Conventional routing protocols can be broadly classified as *distance vector* and *link state* protocols. *Distance vector* routing uses the classical distributed Bellman-Ford algorithm [11, 30, 40, 55]. Each host maintains for each destination a set of distances through each of its neighbors. In order to maintain up-to-date information, each host periodically broadcasts to each of its neighbors, its current estimate of the shortest path to every other host in the network. For each destination, the host determines a neighbor to be the next hop for that destination if the neighbor has the shortest path to the destination.

Link state routing requires each host to have knowledge of the entire network topology [56]. To maintain consistent information, each host monitors the cost of each communication link to each of its neighbors, and periodically broadcasts an update in this information to all other hosts in the network. Based on this information of the cost of each link in the network, each host computes the shortest path to each possible destination host. The processing overhead and the network bandwidth overhead of *link state* protocols are generally more than *distance vector* protocols.

The problems in using conventional routing protocols in a dynamic network have been discussed in great detail in [39, 61]. For completeness sake, we briefly list the problems in the following.

- The conventional routing protocols were not designed for networks where the topological connectivity is subject to frequent, unpredictable change as evident in dynamic networks. Most of them exhibit their least desirable behavior for highly dynamic networks.
- Existing protocols could place heavy computational burden on mobile computers in terms of battery power, and the wireless networks in terms of network bandwidth.
- Convergence characteristics of these protocols are not good enough to suit the needs of dynamic networks.

The protocol described in [61] addresses some of the above stated problems by modifying the Bellman-Ford routing algorithm. They use sequence numbers to prevent routing table loops, and, settling-time data for damping out fluctuations in route table updates. The convergence on the average is rapid, however, the worst case convergence is large. Moreover, their protocol required frequent broadcasts of the routing table by the mobile hosts. The overhead of the frequent broadcasts goes up as the population of mobile hosts increases. Another scheme based on distance vector path-finding algorithm was proposed by [59]. Although loops are avoided completely, all the nodes end up sending an update message to their neighbors during a topology update operation. In dynamic networks, where topology updates are frequent, the update overhead may be very high.

Johnson propose a new routing method for ad-hoc networks based on separate route discovery and route maintenance protocols [39]. The concept of Address Resolution Protocol (ARP) is extended to discover routes. However, if proper measures are not taken, the network performance can degrade due to the propagation of redundant route discovery requests. Route maintenance is achieved by using hop-by-hop acknowledgement. However, due to such relaxed maintenance measures, the hosts can be using poor (long) routes when better (shorter) routes are available. This will degrade the network performance.

A loop-free routing protocol for dynamic networks is proposed in [22]. Routing optimality is of secondary importance. Rather, their goal is to maintain connectivity between the hosts in a fast changing topology. A distributed routing protocol for mobile packet radio networks is proposed by Corson et al. [16]. Similar to [22], routing optimality is of secondary importance. Instead of maintaining distances from all sources to a destination, the protocol guarantees route maintenance only for those sources that actually desire routes. This property helps in reducing the topology update overhead. However, because of the query-based synchronization approach to achieve loop-free paths, the communication complexity could be high.

2. Proposed Approach

This chapter presents a new methodology for routing and topology information maintenance in dynamic networks [47]. Our approach is motivated by our study of existence of clusters (size greater than 2) in random graphs. The basic idea behind the protocol is to divide the graph into number of overlapping clusters. A change in the network topology corresponds to a change in the cluster membership. The performance of the proposed routing protocol (reconvergence time, and topology update overhead) will then be determined by the average cluster size in the network.

For future reference, let us formally define *clusters*.

Definition 1: A k-cluster *is defined by a subset of nodes which are 'reachable' to each other by a path of length at most* k *for some fixed* k*. A* k-cluster *with* k = 1 *is a clique.* □

This work deals with clusters of $k = 1$, i.e., *1-clusters*. (Hereafter, we refer *1-cluster* simply as cluster.) However, we can also generalize our protocols with values of k greater than one. Each cluster is identified by its members.

Definition 2: The size, $S(C)$ *of a cluster* C *is the number of nodes in* C.□

Definition 3: Edges of a cluster comprise of edges between nodes that are members of the cluster.

Definition 4: A graph is cluster-connected *if it satisfies the following two conditions :*
1) The union of the clusters cover the whole graph.
2) For a connected graph, there is a path from each node to every other node through the edges of the clusters in the graph. □

The main problem here is to develop protocols for cluster maintenance. The protocols should be simple, and should incur low overhead. To this effect, we develop simple protocols to detect, and, build clusters in a graph. We maintain a small number of clusters based on the connectivity criteria (Definition 4). Section B presents the problem of routing in dynamic networks. Protocols to create and maintain the clusters are presented in Section C. Section D presents the proposed routing protocol based on clusters. Section E presents the performance evaluation of the cluster-based approach. Section F presents an overview of the other clustering approaches in literature. Summary is presented in Section G.

B. Preliminaries

The problem addressed in this chapter can be defined as follows:
Given: A dynamic network configuration.
Problem: Find a 'good' loop-free routing between each pair of mobile hosts in the network, where the topological connectivity is subject to frequent unpredictable changes.

The problem requires a *loop-free distributed routing* protocol which determines an acyclic route between each pair of hosts whenever a change in the topology is detected. The protocol is intended for use in networks where the rate of topological change is not so fast as to make "flooding"[1] the only viable routing method, but not so slow as to make any static topology routing applicable. In a loop-free[2] route, the path from one host to another does not traverse through the same node twice. Loop-free routing is desirable to minimize the consumption of resources during routing.

Our algorithm determines 'good' routes from one host to another which are not necessarily the shortest paths. In an environment of frequent topological change, a 'good' route's length is comparable to the shortest route. Each host maintains a data-structure describing the network topology and some routing information. The

[1]Flooding is an algorithm whereby a node broadcasts a message packet to its neighbors, who in turn broadcast the packet to all their neighbors, except the neighbor from which it was received. This process goes on till the message packet reaches the intended destination. This happens provided the destination is connected to the node which originated the flood [16].

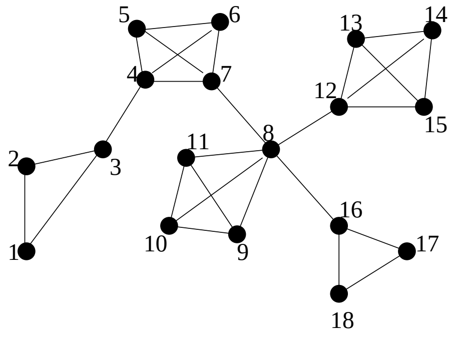
[2]Loop-free routing requires prevention of loops in the routing tables. Here, existence of temporary loops are not of concern.

routing protocol adapts in a distributed fashion to arbitrary changes in topology in the absence of global topological knowledge. Let an undirected graph, $G = (V,E)$, represent a network of mobile hosts. Each node u, in the graph denotes a mobile host H_u. Due to the limited range of wireless transreceivers, a mobile host can communicate with another host only within a limited geographical region around it. This region is called the host coverage area – d being the radius. The geographical area covered by a host coverage area is a function of the medium used for wireless communication. A host H_u is in the vicinity of H_v if the distance between nodes u and v is less than or equal to d. An edge (u,v) connects node u and node v if the corresponding hosts are in the vicinity and have a communication link established between each other. A host may sometime be isolated where there is no other mobile hosts in its vicinity. Such a host will be represented in the graph by a disconnected node. A host H_{v1} is connected to another host H_{v2} if there exists at least one path from node $v1$ to $v2$.

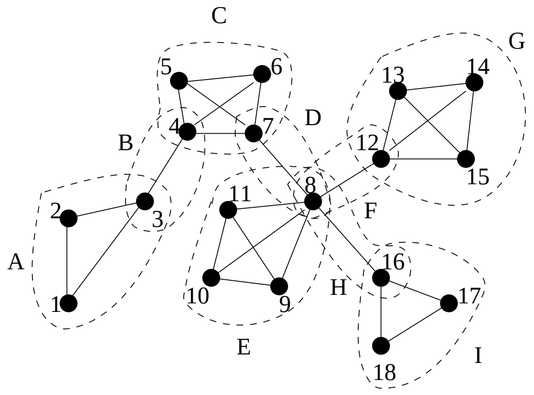
Similar to [16, 59], an underlying link-level protocol is assumed which assures the following:

- A node is aware of all its neighbors at all times.
- All packets transmitted over a link are received correctly and in proper sequence within a finite time.
- All control messages are processed one at a time at the nodes in the order in which they occur.

Example B.1: The graph in Fig. 51(a) is formed based on geographical locations of 18 mobile hosts. In this example, the graph is connected as each node is *reachable* from every other node. It can be observed that based on the positions, some nodes form clusters. The graph can be divided into nine clusters as shown in Fig. 51(b). The clusters and their respective members are as follows : A (1,2,3), B (3,4), C (4,5,6,7), D (7,8), E (8,9,10,11), F (8,12), G (12,13,14,15), H (8,16) and I (16,17,18). If the routing information is based on clusters, routing from node 1 to node 16 will be done through the edges of the clusters A, B, C, D and F. The graph in Fig. 51(b) is *cluster-connected* because, (i) the union of the clusters covers the whole graph, and (ii) there is a path from each node to every other node using the cluster edges. □



(a)



(b)

Fig. 51. An Example of Clusters

A topological change in the mobile host network corresponds to a change in the graph structure $G(V,E)$ to $G'(V',E')$. We outline four events that can cause changes in the graph (in the following H_A and H_B are mobile hosts) :

A) $\underline{H_A \text{ switching ON:}}$ A host H_A switching ON will include itself in the graph and make connection with all the hosts in its 'vicinity'. Hence, $V' = V \cup \{A\}$ and $E' = E \cup \{(u, A)$, s.t. H_u is connected to $H_A\}$.

B) $\underline{H_A \text{ switching OFF:}}$ A host H_A switching OFF will exclude itself from the graph and delete all its edges. Hence, $V' = V - \{A\}$ and $E' = E - \{(u, A)$, s.t. $(u, A) \in E\}$.

C) $\underline{H_A \text{ gets connected to } H_B\text{:}}$ Here, an edge between A and B will be added to the graph. Hence, $V' = V$ and $E' = E \cup \{(A, B)\}$.

D) $\underline{H_A \text{ gets disconnected from } H_B\text{:}}$ Here, the edge between A and B will be removed from the graph. Hence, $V' = V$ and $E' = E - \{(A, B)\}$.

A routing protocol will change its routing information based on the above four types of changes in the graph. We now present some definitions and properties which will assist in describing the proposed routing protocol.

Definition 5: The cluster set S_n of a node n is defined as the set of all clusters in which n is a member. $\square$

Definition 6: If cluster-connectivity between any pair of nodes (n, n') is not affected due to removal of a cluster C, then cluster C is redundant. $\square$

In other words, if two nodes initially cluster-connected, are no longer cluster-connected after the removal of a cluster C, then cluster C is not redundant (i.e., irredundant). For example, in Fig. 51(b), there are no redundant clusters.

Definition 7: A node is a boundary *node if it is a member of more than one cluster.* $\square$

In Fig. 51(b), node 3 is a boundary node as it belongs to two clusters, (1,2,3) and (3,4). However, node 1 is not a boundary node as it only belongs to (1,2,3).

Property 1: Addition of each new node to the graph adds at least one new irredundant cluster. However, when the new cluster is added to the graph, the new cluster may cause one or more clusters to be redundant. □

At least one new cluster should be added to include the new node. Otherwise, the graph will not remain cluster-connected after addition of the new node.

C. Cluster Formation

Our proposed routing protocol is based on the formation of clusters. Hence, efficient cluster formation will be the crux of a routing protocol of this nature. Clusters should be formed in such a way that the resulting graph is *cluster-connected* (See Definition 4). Routing from one node to another will consist of routing inside a cluster and routing from cluster to cluster. A change in the dynamic network may or may not result in a change in the cluster compositions. Here, we have assumed clusters with $k = 1$ (See Definition 1). As mentioned in Section B, we have identified four different possible types of changes in the dynamic network graph in the occurance of a single event. We assume that each cluster has an unique identifier, *id*. Each node maintains a list of its neighbors, a list of clusters (*Clus_List*) in the network, and a list of boundary nodes (*Bound_List*) in the network. There can be multiple boundary nodes between overlapping clusters. If there are multiple boundary nodes between clusters, one with the biggest cluster set is chosen to be the boundary node and is maintained in the *Bound_List*. Note that a node can be a boundary node for more than two overlapping clusters.

In a connected network, *Clus_List* is the same in all the nodes. It is not true in a partitioned network. This is because nodes in a partitioned network may not be aware of all the clusters in the network. Unless otherwise mentioned, the following discussions of the protocols consider a connected graph. Thus, unless otherwise mentioned, all the nodes in the network have the same *Clus_List*. We now present the protocols for cluster updates with each type of topological change.

1. Host H_A Switches ON

The new graph structure $G'(V',E')$ is formed with the added node. The new node A will result in at least one new cluster so that with the cluster, node A can route to the rest of the graph. However, if A connects two disjoint subgraphs, it may result in more than one added cluster. These new clusters are denoted by *essential* clusters and are determined by A itself. The addition of new clusters may result in zero or one or more clusters being redundant. The two tasks performed during the topological change are (i) addition of new clusters, and (ii) removal of redundant clusters. The goal is to have small number of clusters such that the network remains *cluster-connected*. The protocol initiated by new node A is shown in Table IV.

The new node A broadcasts a message to its neighbors indicating its arrival. Upon receipt of the arrival message, the neighbors send a list of their neighbors, and *Clus_List* to A. Based on the neighbor information received from its neighbors, A determines the possible clusters using **Create Clusters** function shown in Table V, and stores them in *All_List*. The clusters that **Create Clusters** function determines depends heavily on the order in which each node is added in the network. This function will not return the maximum clique for all the orders. This function uses a 'first-fit' strategy to generate clusters, which does not necessarily produce maximum sized clusters. The time complexity of the **Create Clusters** function is O(D^3).

Property 2: The clusters returned by the **Cluster Create** *function are characteristic of the order in which each node is added to the network.* □

Table IV. Switch ON Procedure

```
    **Procedure Switch ON**($A$);
    **Begin**;
1.  $A$ sends messages to its neighbors about its arrival;
2.  Each neighbor sends list of its neighbors and *Clus_List* to $A$;
3.  $A$ determines those clusters that are included in the cluster set of
    its neighbors and stores them in *Local_List*;
4.  $A$ uses the neighbor information and creates new clusters using
    **Create Clusters** ($A$) and stores them in *All_List* ;
5.  $A$ executes **Find Essential**($A$,*All_List*);
6.  $A$ assigns new *id*s to the *Essential Clusters*;
7.  $A$ appends the *Essential Clusters* to *Local_List*;
8.  $A$ executes **Find Redundant** (*Local_List*) ;
9.  $A$ appends *Local_List* returned by **Find Redundant** to *Clus_List*;
10. $A$ determines new boundary nodes from the updated *Clus_List* ;
11. $A$ broadcasts the updated boundary node list (*Bound_List*) and
    cluster list (*Clus_List*) to its neighbors;
12. Updated boundary node list and cluster list is then propagated to rest
    of the network by only the boundary nodes;
    **End**;
```

Table V. Create Clusters Function

```
     **Function Create Clusters($A$);**

     **Data Structures**:
     $C_i$ = $i$-th Cluster;
     $Neighbor(n)$ = List of neighbors of node $n$ that are **also** neighbors of $A$;
     DONE($n$) = Indicator of whether clusters including node $n$ have been
     already created or not.
     **Initialization**:
     $1 \leq n \leq |V|$, DONE($n$) = FALSE;
     $All\_List = \{\emptyset\}$
     i = 1;
     **Begin**;
1.   For each node $x$ in $Neighbor(A)$ do
2.       $C_i = \{x, A\}$ ;
3.       For each node $y$ in $Neighbor(x)$ do
4.           if DONE($y$) = FALSE
5.               if $C_i \subseteq Neighbor(y)$
6.                   $C_i = C_i \cup \{y\}$ ;
7.               else
8.                   $All\_List = All\_List \cup C_i$ ; i = i + 1 ;
9.                   $C_i = \{x, y, A\}$;
10.                  $All\_List = All\_List \cup C_i$ ; i = i + 1 ;
11.      DONE($x$) = TRUE ;
     **End;**
```

Fig. 52 illustrates an example where different node numbering (i.e., order in which a node is added to the network) leads to two different set of clusters being created at the new node (Node 6) by the **Create Clusters** function. One can note that a cluster of the largest size may or may not be determined by the algorithm[3] presented in **Create Clusters** function. However, as has been shown later, the algorithm ensures the connectivity of the new node with its neighbors through the clusters.

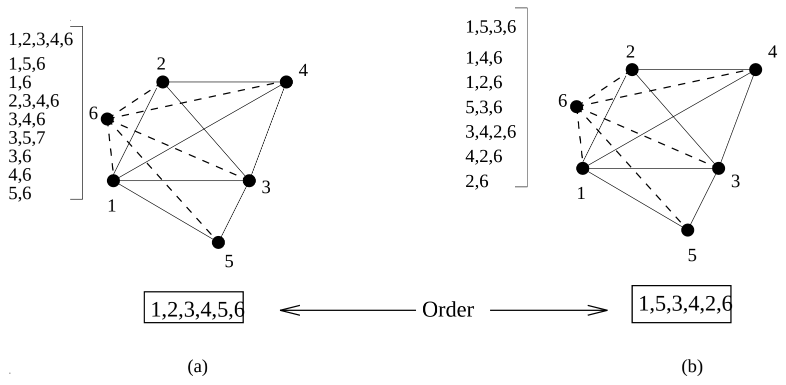


Fig. 52. Different Clusters Created at New Node for Different Orders of Node Addition

Once, the clusters are created using **Create Clusters**, the new node A then executes **Find Essential** function shown in Table VI. The **Find Essential** function sorts the clusters in *All_List* in a non-descending order of their sizes. Initially all the clusters are marked *essential*. Each *essential* cluster C is then examined to find if a node (other than the new node A) in C is a member of any other *essential* clusters. If so, it marks the cluster C as *non-essential*. This will ensure that a node (other than the new node A) is a member of no more than one *essential* cluster. Moreover, since the clusters are sorted in a non-descending order of their sizes, the **Find Essential** function returns the largest clusters possible. The *essential* clusters determined by **Find Essential** function are stored in *Essential Clusters*.

[3]Finding the largest size cluster is NP-Hard [26].

Table VI. Find Essential Function

```
      **Function Find Essential**($A$, $All\_List$);
      **Begin**;
1.    Sort the clusters in $All\_List$ in a non-descending order of their sizes;
2.    For each cluster $C \in All\_List$ do
3.      $Mark(C) := essential$ ;
4.    For each cluster $(C \in list) \wedge (Mark(C) = essential)$ do
5.      For each node $(n \in C) \wedge (n \neq A)$ do
6.        For each cluster $(C' \in All\_List) \wedge (C' \neq C) \wedge (Mark(C') = essential)$ do
7.          if $(n \in C')$
8.            $Mark(C) := non\text{-}essential$;
9.            break;
10.     if $(Mark(C) = essential)$
11.       $Essential\ Clusters := Essential\ Clusters \cup C$;
      **End;**
```

Table VII. Find Redundant Function

```
    **Function Find Redundant**($Local\_List$);

    **Initialization**; Set of nodes: $S = \{\emptyset\}$; $T = \{\emptyset\}$;
    **Begin**;
1.  Sort the clusters in $Local\_List$ in non-descending order of their size.
    Clusters of same size are sorted in non-descending order of their $id$;
2.  For each cluster $C \in Local\_List$ do
3.      $S = S \cup C$; /* Nodes in $C$ are appended to $S$ */
4.  For each cluster $C \in Local\_List$ do
5.      $T = \{\emptyset\}$;
6.      $\forall C'$ s.t., $C' \in Local\_List$, $Mark(C') = FALSE$;
7.      For each cluster $C' \in Local\_List \wedge (C' \neq C) \wedge (Mark(C') = FALSE)$
8.          if($T = \{\emptyset\}$)
9.              $T = T \cup C'$ ; /* Nodes in $C'$ get appended to $T$ */
10.             $Mark(C') = TRUE$ ;
11.         else for each node ($i \in T$)
12.             For each cluster $C'' \in Local\_List \wedge (C'' \neq C) \wedge$
                ($Mark(C'') = FALSE$)
13.                 if($i \in C''$)
14.                     $T = T \cup C''$ ; /* Nodes in $C''$ get appended to $T$ */
15.                     $Mark(C'') = TRUE$ ;
16.     if($T = S$) /* Cluster-connectivity maintained */
17.         $Local\_List := Local\_List - C$ ;
    **End;**
```

The new node A determines the new cluster *id*s of the *essential clusters* based on the information in the cluster list (*Clus_List*) obtained from its neighbors. It then appends the *essential* clusters to list of local clusters (*Local_List*). The list of local clusters (*Local_List*) is obtained from *Clus_List* (Step 3 of **Switch ON**). Local clusters are those clusters in *Clus_List* which are also included in the cluster set of A's neighbors.

Addition of the *essential* clusters may make one or more existing clusters *redundant*. The new node A then executes the **Find Redundant** function shown in Table VII. Node A first sorts the clusters in the *Local_List* in ascending order of size. Clusters of same size are sorted in the order of ascending *id*s. The **Find Redundant** function then determines *redundant* clusters based on Definition 6. The new cluster list is then obtained by appending the clusters remaining in *Local_List* after removing the *redundant* clusters, to *Clus_List*. Node A then determines the list of boundary nodes (*Bound_List*) from the updated *Clus_List*. If there are multiple boundary nodes between overlapping clusters, one with the biggest cluster set is chosen to be the boundary node. Node A then broadcasts the updated boundary node list (*Bound_List*) and cluster list *Clus_List* to its neighbors. The neighbors then replace their cluster list and boundary node list with the ones obtained from A. The updated boundary node list (*Bound_List*) and cluster list *Clus_List* is then propagated to the rest of the network only by the boundary nodes.

If $\mathcal{B}$ is the upper bound on the number of boundary nodes, and $\mathcal{D}$ the maximum nodal degree, the message complexity of **Switch ON** is $O(\mathcal{B}+\mathcal{D})$. The number of boundary nodes, $\mathcal{B}$, is upper bounded by the number of nodes in the network, N.

Example C.1: For an easier understanding, Fig. 53 gives an example involving a network with 4 nodes. Fig. 53(a) has 4 nodes and two clusters, namely, (1,2,3) and (2,3,4). When node 5 is switched ON, it sends messages to nodes 1, 3, and

4 (Fig. 53(b)). On receiving information back from the nodes 1, 3 and 4, node 5 forms clusters (1,3,5), (3,4,5) and (4,5) as seen in Fig. 53(c). It chooses (3,4,5) as the *essential cluster* and then determines redundant clusters from the cluster list of {(1,2,3), (2,3,4), (3,4,5)}. In the redundant removal phase, the new node 5 detects the cluster (2,3,4) to be redundant. The final clusters are (1,2,3) and (3,4,5) as in Fig. 53(d).□

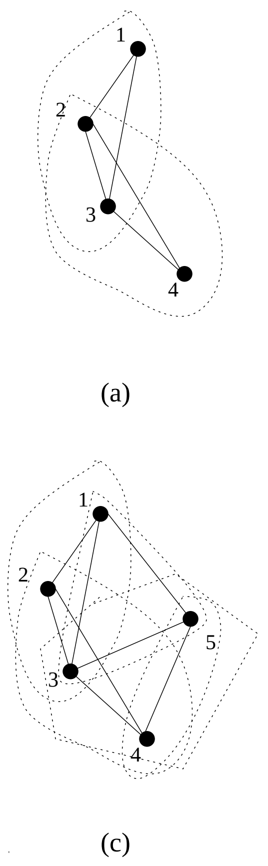


Fig. 53. An Example of a Node Addition

2. Host H_A Switches OFF

When host H_A turns OFF, its disappearance will only be detected by its neighbors. The clusters in the cluster-set of node A shrinks in size. The neighbors of node A

who are cluster-mates of the shrunk cluster will 'expand' the cluster. By expanding a cluster, we mean that the neighbor will determine new nodes to become a member of that cluster. Neighbors of node A that are not cluster-mates of A will not initiate any update procedures.

There could be more than one node detecting the removal of a node. **Switch OFF** procedure is similar to **Switch ON** procedure in the sense that, there are new clusters formed and redundant clusters removed. Concurrent independent executions of **Switch OFF** procedure could lead to violation of the *cluster-connectivity* condition. We use an arbitration procedure to avoid concurrent independent executions. We require the node (neighbor of A, say, B) that is a cluster-mate of A in most number of clusters, to initiate the **Switch OFF** procedure[4]. The execution of **Switch OFF** procedure will expand those clusters in the cluster-set of A that node B is a member of. However, there still remains clusters in the cluster-set of A which do not contain B. In those remaining clusters, we determine the node (say, C) that is a member of most number of clusters. This process continues till all the clusters in the cluster-set of A is covered. Unlike node B, node C will not execute **Switch OFF** procedure. However, node C will just try to expand the shrunk clusters that it is part of, and not remove any redundant clusters. The new boundary list (*Bound_List*) and the new cluster list (*Clus_List*) is determined by C and broadcast to its neighbors. The lists are then further propagated to the rest of the network only by the boundary nodes.

The procedure initiated by node B is shown in Table VIII. Let us illustrate it with an example.

Example C.2: Fig. 54 shows the cluster formations when a node is turned OFF in

[4]If there are multiple such nodes, we use a tie-breaking test; e.g., node with the larger $identifier$.

Table VIII. Switch OFF Procedure

```
     Procedure Switch OFF($A$,$B$);
     Begin;
1.   $B$ requests the list of neighbors and Clus_List from the cluster mates
     of the shrinked cluster(s);
2.   The cluster mates send the list of its neighbors and Clus_List to $B$;
3.   $B$ determines those clusters that are included in the cluster set of
     its cluster mates and stores them in Local_List;
4.   Using the neighbor information, $B$ expands the cluster(s). The ids of the
     cluster(s) do not change;
5.   $B$ appends the expanded cluster(s) to Local_List;
6.   $B$ executes Find Redundant (Local_List) ;
7.   $B$ appends Local_List returned by Find Redundant to Clus_List;
8.   $B$ determines new boundary nodes from the updated Clus_List ;
9.   $B$ broadcasts the updated boundary node list (Bound_List) and
     cluster list (Clus_List) to its neighbors;
10.  Updated boundary node list and cluster list is then propagated to rest
     of the network by only the boundary nodes;
     End;
```

a network. Fig. 54(a) has six nodes with three clusters, namely, (1,2,3), (2,3,4) and (4,5,6). When node 6 is turned OFF, the cluster (4,5,6) shrinks to (4,5) (Fig. 54(b)). Node 4 and 5 detect node 6 switching OFF. Since, node 5 has the higher identifier, it initiates the **Switch OFF** procedure. Node 5 gets neighbor information and the cluster list from 3 and 4. It then expands the cluster (4,5) to (3,4,5) (Fig. 54(c)). Node 5 now has {(1,2,3), (2,3,4), (3,4,5)} in the cluster list. In the redundant removal phase, node 5 detects the cluster (2,3,4) to be redundant. The final clusters are (1,2,3) and (3,4,5) as in Fig. 54(d). □

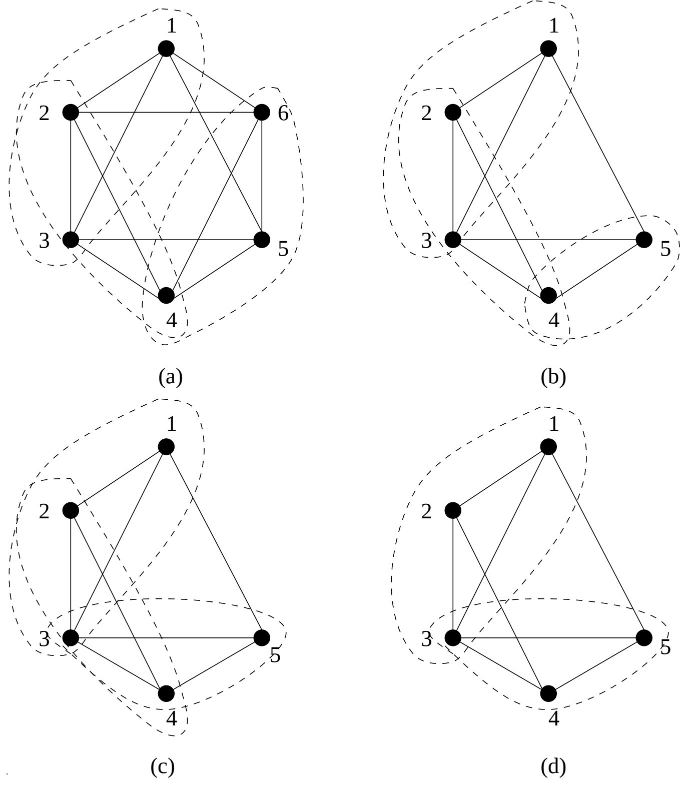


Fig. 54. An Example of a Node Removal

The message complexity of **Switch OFF** is also O($\mathcal{B}$+$\mathcal{D}$), where, $\mathcal{B}$ is the upper bound on the number of boundary nodes, and $\mathcal{D}$ the maximum nodal degree. As stated earlier, the number of boundary nodes, $\mathcal{B}$, is upper bounded by the number of nodes in the network, N.

3. Host H_A Gets Connected to Host H_B

The new connection between hosts H_A and H_B could be detected simultaneously by both the nodes. We require that only the node with the larger[5] *identifier* to execute the procedure to determine new clusters due to the new connection. This is possible because each node periodically sends a beacon which includes the node identifier (Section D.3). Let the node with the larger identifier be A, and the other node with a smaller identifier be B. Node A then initiates the **Switch ON** procedure. Node B becomes one of the neighbors taking part in the **Switch ON** procedure by sending the neighbor list and the cluster list to node A. The new cluster list and the boundary node list is determined and propagated to the rest of the network as explained earlier in Section C.1.

4. Host H_A Disconnects Host H_B

Here, we identify two cases as follows.

1. Node A was not a cluster-mate of node B: The topological change will result in no change in any clusters in the network.

2. Nodes A and B belong to same clusters: Here, the topological change will result in the shrinking of the involved clusters. Both A and B will detect that the link between them has broken. They will both initiate the **Switch OFF** protocol. **Switch OFF** protocol comprises of adding new clusters and removing redundant clusters. Concurrent independent executions of **Switch OFF** protocols at two different nodes could lead to violation of *cluster-connectivity*

[5]Any tie-breaking test will suffice.

condition. We avoid independent executions of **Switch OFF** protocols at two different nodes by requiring only the node with the larger *id* (say, A) to execute the **Switch OFF** protocol. The other node with smaller *id* (say, B) provides new *id*s to the shrunk clusters, updates *Clust_List*, determines new boundary node list (*Bound_List*) and broadcasts both these lists to its cluster mates. The lists are then further propagated to the rest of the network only by the boundary nodes. Thus, node B (i.e., the node with smaller *id*) does not remove any redundant clusters. Redundant cluster determination and removal is done only by A during its execution of **Switch OFF** protocol. The new cluster list and the boundary node list is determined and propagated to the rest of the network as explained earlier in Section C.2.

D. Routing Protocol

We first discuss the necessary data structures to be maintained at each node for the routing protocol. We will then explain the route construction and maintenance procedures in the network.

1. Data Structures

As stated earlier, the following lists are maintained at each node :

- *Clus_List*: This list provides the mapping between the clusters and its members.
- *Bound_List*: This list maintains the 'designated' boundary nodes between overlapping clusters. As stated earlier, there may be more than one boundary node between overlapping clusters. Only one among them is chosen to be the designated boundary node (Section C).

Using the information in *Clus_List* and *Bound_List*, each node then generates the routing tables used for routing packets. Each entry in the routing table contains the destination identifier, the next hop node and the number of hops it takes to reach the destination via that next hop node. This is similar to the routing tables maintained in distance-vector protocols. The routing tables are as follows:

- *AllRouteTable*: For each destination node, this table maintains route information of all possible paths via clusters from the node. This table is used to determine the shortest 'available' path to each destination node, which is maintained in *RouteTable*.
- *RouteTable*: For each destination node, the node maintains identifier of the next hop node, say n, and the number of hops it will take to reach the destination node via n. This is the table which is referred to while routing a packet.

The *Clus_List* and *Bound_List* for the network in Fig. 51 are shown in Tables IX and X. The *AllRouteTable* for Fig. 51, happens to be same as its *RouteTable* (Table XI), because, there is just one possible path via clusters between any two nodes. On the other hand, if there were multiple paths via clusters then, for each additional path, there would have been two additional columns for next hop node and number of hops in the *AllRouteTable*.

2. Protocol

A routing protocol can be divided into two phases, namely, *route construction* and *route maintenance*. During the *route construction* phase, routes are constructed between all pairs of nodes. The *route maintenance* phase takes care of maintaining loop-free routes in the face of unpredictable topological changes.

Table IX. *Clus_List* at Each Node

$ClusterId$	$Nodes$
A	1,2,3
B	3,4
C	4,5,6,7
D	7,8
E	8,9,10,11
F	8,12
G	12,13,14,15
H	8,16
I	16,17,18

Table X. *Bound_List* at Each Node

$ClusterIds$	$Node$
A,B	3
B,C	4
C,D	7
D,E,F,H	8
F,G	12
H,I	8,16

Table XI. *RouteTable* at Node 6, Cluster C

$DestNode$	$NextHop$	$Hops$
1	4	3
2	4	3
3	4	2
4	-	1
5	-	1
6	-	0
7	-	1
8	7	2
9	7	3
10	7	3
11	7	3
12	7	3
13	7	4
14	7	4
15	7	4
16	7	3
17	7	4
18	7	4

a. Route Construction Phase

The protocols to maintain clusters in the face of various network events have been explained earlier. Upon receipt of new cluster information, a *boundary* node stores the new cluster list in its *Clus_List*, the new boundary list in its *Bound_List*, and then rebroadcasts the information. A boundary node has to forward the new information only once. Nodes other than the boundary nodes listen to this information and just update their tables. In this manner, the information of each network event is distributed to all the nodes. Each node now has the topology information of the whole network. Based on the information in *Clus_List* and *Bound_List*, each node then generates the *RouteTable* and *AllRouteTable*.

Each message packet contains the identifier of the destination node in its header. When a node receives a message packet, it looks up the *RouteTable* to determine the next hop node for the packet's destination. The node then forwards the message packet to the next hop node. This process of forwarding continues till the packet reaches its destination.

b. Route Maintenance Phase

This phase begins when there is a change in the network topology (host connection/disconnection, link failure/recovery). The route maintenance in our approach basically boils down to cluster maintenance. The protocols for cluster maintenance have been explained previously. After a change in topology, all the nodes have the complete topology information in the form of cluster list (*Clus_List*) and boundary node list (*Bound_List*). If all the nodes have a consistent view of the topology, routing loops are not formed. However, due to long propagation delays, partitioned network, etc., some nodes may have inconsistent topology information. This might lead to formation of routing loops. However, these loops are short-term, because they disappear within bounded time (required to traverse the diameter of the network) [56]. Even these loops can be avoided if each route table entry is tagged with a sequence number so that nodes can quickly distinguish stale routes from the new ones and avoid formation of routing loops [61].

The new cluster information will be propagated throughout the network. It should be noted that only the boundary nodes are responsible for broadcasting and rebroadcasting any new information. This helps in quick dissemination of information across the network. Thus, the reconvergence of the cluster-based protocols is very quick. Let us illustrate it with an example. Let node 2 in Fig. 51 disconnect. This event will be detected by nodes 1 and 3. Since node 1 is not a boundary node, it will just update its tables to indicate the change. Node 3 being the boundary node broadcasts the new cluster information. Node 4, a boundary node, upon receipt of the new cluster information from node 3, re-broadcasts it. This broadcast will be received by nodes 3, 5, 6 and 7. Since node 3 has already broadcasted this cluster information, it neglects this information. Nodes 5 and 6 being non-boundary nodes just update their tables. However, node 7 being a boundary node, updates its tables and rebroadcasts the new cluster information. Similarly, other boundary nodes 8, 12 and 16 upon receipt of the new cluster information re-broadcast it so that every node in the network have the new cluster information. And, the non-boundary nodes just listen and update their tables and do not re-broadcast.

3. Implementation Details

- Detection of a new link : Each host periodically broadcasts a beacon which includes its identifier. If a host h receives a beacon from another host h' which is not in its current neighbor set, it means that there is a prospective new link to

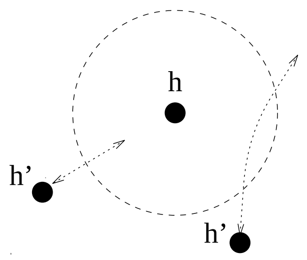


Fig. 55. Movements That Cause Unnecessary Link Creations/Deletions

be created. However, the **Switch ON** procedure is not immediately initiated. Only after a certain number of successive beacons is received from the same host is the **Switch ON** procedure initiated. This is to avoid unnecessary oscillations due to the host h' moving in and out of host h's vicinity. Fig. 55 shows the scenarios where the movement of h' could cause a sequence of unnecessary link creations/deletions.

- Detection of a link break : If a host h does not receive a certain number of consecutive beacons from its neighbor h', it will assume that either h' has moved out of its vicinity or that h' is disconnected. Host h will then follow the procedure for host disappearance as explained in Section C.2.

E. Performance Evaluation

1. Complexity

This section compares the cluster-based approach's worst-case performance with the performance of *Distributed Bellman-Ford* (DBF) [11], *Ideal Link State* (ILS) [25], *Diffusing Update Algorithm* (DUAL) [25], NP [16] and flooding. The ILS protocol [25] requires that each topology change be transmitted to every node. The DUAL protocol [25] is a distance-vector loop-free algorithm based on internodal coordination spanning multiple hops. DUAL is known to be the lowest complexity distance-vector algorithm. NP protocol [16] is source-initiated routing protocol that provides loop-free routing only to desired destinations in a dynamic network. Flooding does not have any control overhead due to topology updates/maintenance. Everytime a node wants to send a packet to a destination, the node broadcasts the packet to its neighbors, who in turn broadcast the packet to all their neighbors, except the neighbor from which it was received. This process goes on till the message packet reaches the intended destination.

The performance metrics are the time complexity (TC) and the communication complexity (CC) [25]. Time complexity is defined as the number of steps required for the network to reconverge after a topology change. The number of messages required to accomplish the reconvergence is called the communication complexity. The assumptions made while making the comparisons are same as in [25]. They are as follows:

- The routing algorithm behaves synchronously, so that every host in the network executes a step of the algorithm simultaneously at fixed points in time.
- At each step, the host receives and processes all the inputs originated during the preceding step and, if required, sends update messages at the same step.

We borrow the complexity computations of DBF, ILS, and DUAL from [25]. Table XII lists the protocols with the complexities. The complexity parameters are as follows:

- N: Number of nodes in the network.
- E: Number of links in the network.
- d: Diameter of the network. The diameter of a network is defined as the length of the longest shortest path in hops between any two nodes [25].

- $\mathcal{D}$: Maximum degree of a node.
- $\mathcal{B}$: Upper bound on the number of *unique* boundary nodes in the network. Overlapping clusters may have more than one boundary node between them. However, only one of them will be considered as the boundary node and will be used to pass messages between clusters. The other boundary nodes are considered as non-boundary nodes. The procedure to select a boundary node has been described in Section C.
- x: Number of nodes affected by the topological change.
- l: Diameter of the affected network segment.

Table XII. Complexity Comparison

Protocol	**TC**	**CC**
DBF [11]	O(N)	O(N^2)
ILS [25]	O(d)	O(E)
DUAL [25]	O(x)	O($\mathcal{D}$x)
NP [16]	O(l)	O(x)
Cluster	O(d)	O($\mathcal{B}$ +$\mathcal{D}$)
Flooding	0	0

Since, flooding does not have any topology update overhead, the time complexity and communication complexity of flooding is zero. The complexities of DUAL and NP will be high if $x \approx N$ (This is true in the situations when a node fails or switches off.), i.e., when most of the nodes in the network are affected by the topological change. In such cases, the diameter of the affected segment, $l \approx d$. The performance of the cluster-based approach depends on number of boundary nodes and the maximum degree of a node. We resort to simulations to determine the variation of number of boundary nodes, cluster size, with degree of the network. We will show through simulations that even for low nodal degrees, the number of boundary nodes in a network is much less than the total number of nodes in the network. We also determine the *routing overhead* of the cluster-based approach.

2. Simulations

Simulations are performed to determine average cluster size, and number of boundary nodes for random graphs. The routing overhead of the cluster-based approach is also determined. *Routing overhead* is ratio of the path length between a source and a destination as determined by the cluster-based approach and the actual shortest path length between them.

Random graphs are generated using the *random graph generator* function presented in the Appendix E. The clusters are determined using the **Switch ON** procedure described in Section C.1. Input to the simulations are (i) N (number of nodes), and (ii) D (average degree in the network). As shown in Fig. 56, the average cluster size increases as N increases. It also increases when D increases. Fig. 56 shows that there is a large region of values of N and D where the average cluster size is greater than 2. In these scenarios, clustering will benefit. Fig. 57 and Fig. 58 illustrate the variation of number of clusters and number of boundary nodes with degree, respectively. Note that the number of clusters and boundary nodes in the network decrease as degree increases. Also note that they increase as number of nodes in a network increases. The maximum number of boundary nodes for a given N occurs when D is low. However, the maximum number of boundary nodes is much less than N. For example, for N=10, the maximum number of boundary nodes is 5 (with $D = 2$).

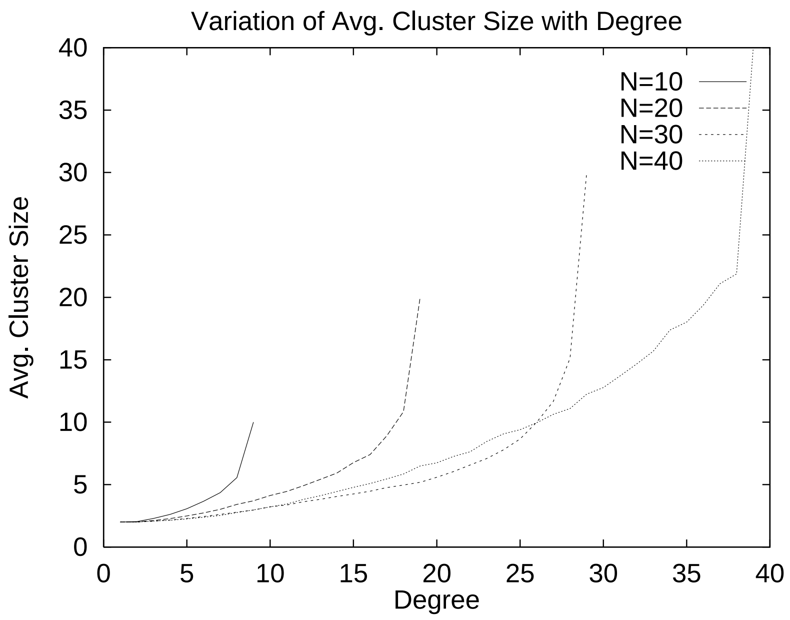


Fig. 56. Variation of Average Cluster Size with Degree

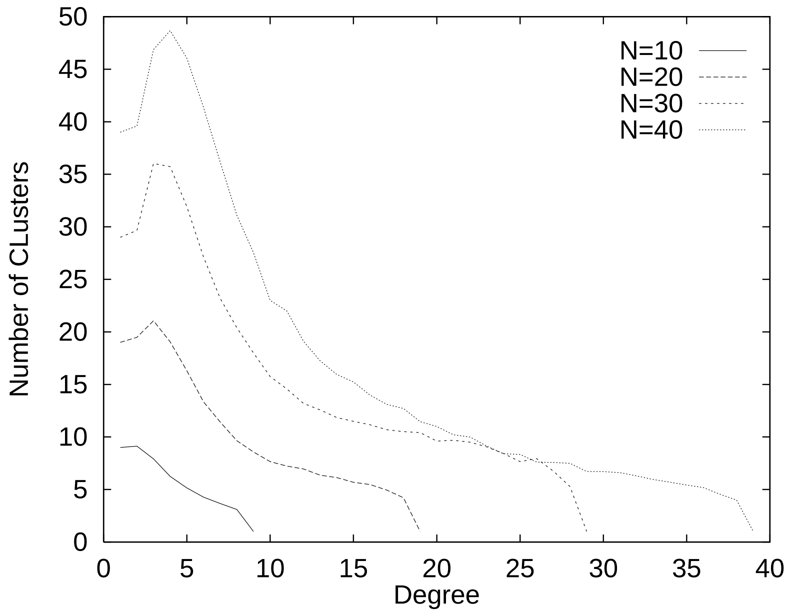


Fig. 57. Variation of Number of Clusters with Degree

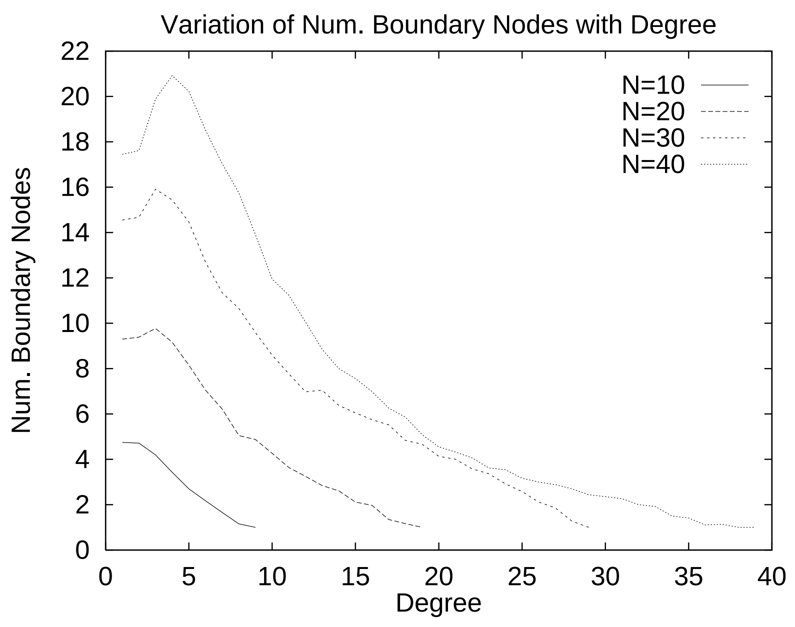


Fig. 58. Variation of Number of Boundary Nodes with Degree

In other words, in such a network, if cluster-based approach is used, the number of nodes taking part in the topology update protocol will be less than 50% of the total number of nodes in the network. Fig. 59 and Fig. 60 illustrate the variation of average path length of the cluster-based approach and flooding with degree for N=10 and N=30 respectively. The average path length is computed as the average of the path lengths between each source and destination in the network. Flooding always determines the shortest path between two nodes. Note that the average path length determined by the cluster-based approach is higher than the average path length determined by flooding. The routing overhead determined as the ratio of the path lengths determined by clustering and flooding is observed to be less than 2 for both the cases considered (N=10 and N=30). Compared to savings in network load due to updates, the routing overhead of the cluster-based approach is not high.

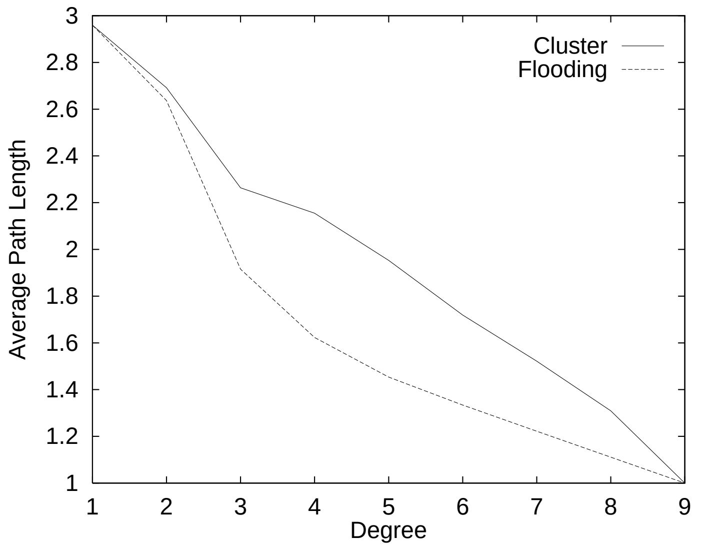


Fig. 59. Variation of Average Path Length with Degree: N=10

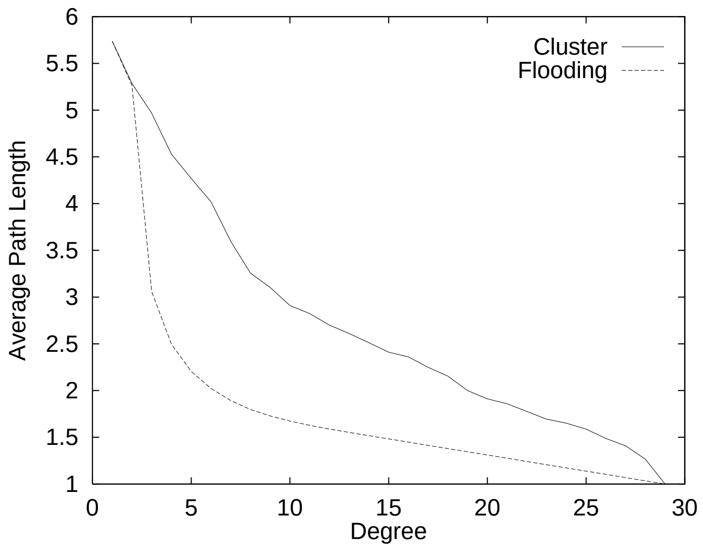


Fig. 60. Variation of Average Path Length with Degree: N=30

F. Other Clustering Approaches

The problem of clustering in networks has been discussed earlier in literature [23, 34, 41, 66, 70]. Our work differs from the earlier works in the following respect:

- Clustering was proposed as a hierarchical approach in earlier literature to reduce the amount of routing information stored at individual hosts. The entire network is thought of as a tree of hierarchies, in which each node at a higher level is made up of one or more nodes from lower levels. Each host has to take part in two updating procedures: one local within its cluster, and the other global with other distant nodes. In this paper, clustering is restricted to a single level. The main advantage behind using cluster-based approaches is that the way we maintain the clusters, which, limits the number of nodes taking part in the topology-update operation, thereby, reducing the network load during topology updates.

- The cluster creation and maintenance algorithms have not been discussed in most of the literature where if it is discussed, it either is specifically for regular graph structures [41, 70], or employs a cluster controller (or leader) [66]. In this work we create and maintain a small number of clusters (cliques) in an arbitrary graph. The cluster graph is created using a sequence of **Switch ON** procedures (one procedure call for each node being added). The cluster is maintained in the face of different network events by calling the appropriate algorithms as explained in this work.

- Cluster overlapping in some approaches requires each node to be included in more than one cluster [66, 70]. However, in this work we do not require all the nodes to be included in more than one cluster.

- Unlike the previous approaches, we require our clustering algorithms to create and maintain clusters such that they satisfy the cluster-connectivity criterion (Definition 4). Since, we require the network to be cluster-connected, we can apply any routing protocol directly by just replacing the nodes by clusters. Thus, we can enjoy the advantages of a chosen routing protocol (loop-free routes, etc.), and also the cluster-based approach (low topology update overhead, etc.).

G. Summary

Proposed in this chapter is a new methodology for routing in mobile wireless networks. Simple distributed algorithms are proposed for cluster creation and maintenance. This chapter shows that routing protocols based on clusters could obtain performance improvements over previous approaches. Cluster-based protocols allow the network to enjoy the liberty of maintaining routes between all pairs of nodes at all times, without causing much network overhead. Thus, a compromise on routing optimality as suggested in [16] to avoid network congestion might not be required.

Quick reconvergence in some protocols like DSDV [61] is obtained by quick re-broadcast by each and every recipient of the broadcast, causing degradation of the availability of the wireless medium. However, in our approach, re-broadcast is done **only** by the *boundary* nodes. Nodes other than *boundary* nodes just listen and update their tables.

Similar to [16, 22] the cluster-based approach does not guarantee shortest path. This is due to the fact that the clusters are created using the first-fit approach, which does not produce the maximum clusters in the graph. However, it has been shown that the routing overhead of the cluster-based approach is not high.

CHAPTER VI

CONCLUSION

This chapter contains a summary of results presented in this dissertation, followed by suggestions for future work.

A. Summary of Results

We classify the mobile wireless networks into infrastructure networks and dynamic networks. Infrastructure networks are typically two-tiered network composing of a static backbone network and a peripheral wireless network. Dynamic networks on the other hand comprises of only mobile hosts that communicate with one another using wireless links. This dissertation studies the following performance issues in mobile wireless networks – recovery issues in infrastructure networks, location management issues in infrastructure networks, and routing in dynamic networks.

Mobility of users and limited wireless bandwidth bring forth interesting dimensions into design of recovery protocols in a mobile wireless networks. Work in this dissertation is the first effort to study the effect of mobility and wirelessness on the design and performance of recovery protocols. Presented in this dissertation are recovery schemes for a mobile wireless environment. The recovery schemes are a combination of a state-saving and a handoff scheme. Each combination provides some level of availability and requires some amount of resources: network bandwidth, memory, and processing power. A fundamental relationship between the performance of recovery schemes, failure rate, mobility and wireless bandwidth is established. Through analysis, it is shown that there can be no single recovery scheme that performs well for all mobile environments. However, we determine the optimal recovery scheme for each environment, where an environment is determined by the mobility, wireless bandwidth and the failure rate.

Location management is one of the most important issues in infrastructure mobile wireless networks. In order to communicate with an user, one needs to know the user's location. Thus, the network faces a problem of continuously keeping track of the location of each and every user. This problem becomes noticeable when the network sizes are large. This dissertation presents centralized and distributed location management schemes.

Centralized schemes like IS-41 use home location servers for location management. The location management schemes in IS-41 are inefficient because they incur a very heavy load on the network and the home location servers due to location updates and searches. Forwarding technique is used to lower the network load due to updates in the network and at the home location servers due to user mobility. However, forwarding increases the search cost (call set-up time). Two heuristics are presented to limit the search cost. A search-update (caching) strategy is presented to further reduce the search cost and also the call-delivery rate at the home location server. The performance of the schemes depend on (i) call-mobility pattern of the user, and (ii) cost of forwarding. Analytical models are built to compare the performance of the proposed schemes with the IS-41 scheme. It is determined that the proposed schemes perform significantly better than the IS-41 scheme for most call-mobility patterns and forwarding costs. Although beneficial, forwarding and search-updates complicate the maintenance, and fault-tolerance issues. To overcome these problems, this dissertation also presents cost-effective techniques for fault tolerance and forwarding pointer maintenance.

The bottleneck in the centralized schemes are the home location servers. Instead of a home location server, a hierarchy of location servers is proposed for location management. The signalling load is now distributed over various location servers. A suite of location management schemes for such a network architecture is presented. Each scheme is a combination of a search strategy, update strategy and a search-update strategy. A static location management scheme requires a single combination to be executing always. It is observed that there is no combination that outperformed others for all call-mobility patterns. Since, the user behavior (call-mobility pattern) is not always available to the system designer, there is need for adaptive location management. An adaptive scheme is presented that is based on the assumption that the past history of the system reflects the behavior in the future. Results indicate that the adaptive scheme performs better than the static scheme for a wide range of call-mobility patterns.

The conventional routing protocols were not designed for dynamic networks (e.g., ad-hoc networks, packet radio networks) where the topological connectivity is subject to frequent, unpredictable change. In addition to host and link failures, changes in topology can occur due to host mobility and disconnections. Due to limited bandwidth available on the wireless links, the amount of information exchanged or propagated during topology updates has to be kept low in such networks. To achieve this goal, we propose a cluster-based methodology for routing in dynamic networks. The basic idea behind the protocol is to divide the graph into number of overlapping clusters. The main advantage of our approach is that it limits the number of nodes taking part in the topology-update operation, thereby, reducing the network load during topology updates. We propose simple protocols for cluster creation and maintenance. Compared to existing and conventional routing protocols, the proposed cluster-based approach incurs lower overhead during topology updates and also provides quicker reconvergence. Although, the cluster-based approach does not guarantee shortest path, it is determined using simulations that the routing overhead[1] of the cluster-based approach is small.

B. Future Work

The field of mobile computing is still relatively new. There are many challenging and interesting areas of future research that can stem out of the work presented in this dissertation. The following presents a summary.

Chapter II dealt with application-level recovery protocols in a mobile environment. An opportunity for further research exists in other fault-tolerance issues in mobile computing, such as recovery from failure of a base station, fault-tolerant broadcast/multicast protocols, and development of new and efficient distributed recovery schemes. Although, the analysis presented in this research is used to analyze recovery protocols, we feel that this analytical framework could also be used with some variations to analyze the effect of mobility, wireless bandwidth and disconnections on the performance of file systems, and database systems.

Chapter III presented schemes to improve the performance of location management schemes in a personal communication network. Chapter IV presented location management schemes for a network comprising of a hierarchy of location servers. The schemes were analyzed using analytical modeling or simulations. It would be interesting to determine the performance of these schemes using real call-mobility traces. Secondly, in this work we did not consider variance in the *call set-up* times. From a user's perspective, apart from the mean call set-up time, the variance is also a key parameter. Future work should involve development of schemes that reduce

[1]Routing overhead is ratio of the path length between the source and the destination as determined by the cluster-based approach and the actual shortest path length between them.

the variance. It was shown in Chapter IV that a simple adaptive scheme performs better than static schemes for non-uniform call-mobility patterns. A potential area of research is to study other sophisticated adaptive location management schemes that is able to better predict user behavior.

A cluster-based approach for routing in dynamic networks was presented in Chapter V. Protocols for cluster creation and maintenance were presented. A number of different issues remain to be studied: (a) Extensions of these protocols to support concurrent events. (b) Load balancing among multiple boundary nodes. In our work, there is only one unique boundary node selected between any two clusters for propagating update information. (c) Algorithms to create and maintain clusters such that there is always more than one boundary node between any two overlapping clusters. This will add robustness during a boundary node failure. (d) Generalization of 1-clusters to k-clusters (where $k > 1$). This will require design of more complex algorithms for cluster creation and maintenance. The interesting issue will be to determine if there is any performance improvement using k-clusters (where $k > 1$) instead of 1-clusters.

REFERENCES

[1] A. Acharya, B. R. Badrinath, "Checkpointing Distributed Applications on Mobile Computers," *Proc. International Conference on Parallel and Distributed Information Systems,* Austin, Texas, pp. 73-80, September, 1994.

[2] S. Alagar, R. Rajagopalan, S. Venkatesan, "Tolerating mobile support station failures," *Proc. IEEE Conference on Fault Tolerant Systems,* pp. 225-231, December, 1995.

[3] R. Alonso, H. Korth, "Database System Issues in Nomadic Computing," *Proc. ACM SIGMOD International Conference on Management of Data,* pp. 388-392, May, 1993.

[4] B. Awerbuch, D. Peleg, "Concurrent Online Tracking of Mobile Users," *Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols,* Zurich, Switzerland, pp. 221-233, September, 1991.

[5] B. R. Badrinath, T. Imielinski, A. Virmani, "Locating Strategies for Personal Communication Networks," *Proc. IEEE GLOBECOM Workshop on Networking of Personal Communication,* December 1992.

[6] A. Bakre, B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," *Proc. International Conference on Distributed Computing Systems,* Oct., 1994.

[7] B. Bakshi, P. Krishna, N. H. Vaidya, D. K. Pradhan, "Improving Performance of TCP over Wireless Networks," Technical Report # TR-96-014, Department of Computer Science, Texas A&M University, May, 1996.

[8] H. Balakrishnan, S. Seshan, E. Amir, R. H. Katz, "Improving TCP/IP Performance over Wireless Networks," *Proc. ACM Conference on Mobile Computing and Networking,* pp. 2-11, November 1995.

[9] D. M. Balston, R. C. V. Macario, *Cellular Radio Systems,* Artech House, Boston, Massachusetts, 1994.

[10] Amotz Bar-Noy, H. Kessler, "Tracking Mobile Users in Wireless Communication Networks," *Proc. IEEE, INFOCOM*, pp. 1232-1239, May, 1993.

[11] D. Bertsekas, R. Gallager, *Data Networks,* Second Edition, Prentice Hall Inc., Englewood Cliffs, New Jersey, 1992.

[12] Pravin Bhagwat, Charles. E. Perkins, "A Mobile Networking System Based on Internet Protocol (IP)," *Proc. USENIX Symposium on Mobile and Location-Independent Computing,* pp. 69-82, August 1993.

[13] A. Borg, J. Baumbach, S. Glazer, "A Message System Supporting Fault-tolerance," *Proc. Ninth ACM Symposium of Operating Systems and Principles,* pp. 90-99, October, 1983.

[14] R. Caceres, L. Iftode, "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments," *IEEE Journal on Selected Areas in Communications,* Vol. 13, No. 5, Oct., 1994.

[15] Cellular Digital Packet Data System Specification: Release 1.1, CDPD Forum Inc., January, 1995. (Contact CDPD Forum Executive Director, email: info@forum.cdpd.net)

[16] M. Scott Corson, A. Ephremides, "A Distributed Routing Algorithm for Mobile Wireless Networks," *ACM Journal on Wireless Networks,* Vol.1, No.1, pp. 61-81,

1995.

[17] W. Diepstraten, G. Ennis, P. Bellanger, "DFWMAC - Distributed Foundation Wireless Medium Access Control," *IEEE Document P802.11-93/190,* Nov., 1993. (Contact Wim Diepstraten, email: Wim.Diepstraten@utrecht.ncr.com)

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

[19] S. Dolev, D. K. Pradhan, J. L. Welch, "Modified Tree Structure for Location Management in Mobile Environments," *Proc. IEEE INFOCOM,* pp. 530-537, May, 1995.

[20] G. H. Forman, J. Zahorjan, "The Challenges of Mobile Computing," *IEEE Computer,* Vol. 27, No. 4, pp. 38-47, April, 1994.

[21] R. J. Fowler, "The Complexity of Using Forwarding Address for Decentralized Object Finding," *Proc. ACM Symposium on Principles of Distributed Computing,* pp. 108-120, 1986.

[22] E. Gafni, D. P. Bertsekas, "Distributed Algorithms for Generating Loop-free Routes with Frequently Changing Topology," *IEEE Transactions on Communications,* Vol. COM-29, No. 1, pp. 11-18, January, 1981.

[23] J. J. Garcia-Luna-Aceves, "Analysis of Routing Strategies for Packet Radio Networks," *Proc. IEEE INFOCOM,* pp. 292-302, May, 1985.

[24] J. J. Garcia-Luna-Aceves, "A Unified Approach to Loop-free Routing Algorithm Using Distance Vectors or Link States," *Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols,* pp. 212-223, September, 1989.

[25] J. J. Garcia-Luna-Aceves, "Loop-Free Routing Using Diffusing Computations," *IEEE Transactions on Networking,* Vol. 1, No. 1, pp. 130-141, February, 1993.

[26] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman and Company, New York, New York, 1979.

[27] Alan Gibbons, *Algorithmic Graph Theory,* Cambridge University Press, Cambridge, Massachusetts, 1985.

[28] D. J. Goodman, "Trends in Cellular and Cordless Communications," *IEEE Communications Magazine,* pp. 31-40, June 1991.

[29] J. S. M. Ho, I. F. Akyildiz, "Local Anchor Scheme for Reducing Location Tracking Costs in PCNs," *Proc. ACM Conf. on Mobile Computing and Networking,* pp. 181-193, November, 1995.

[30] C. Hedrick, Routing Information Protocol, RFC 1058, June, 1988. (Available from nic.ddn.mil)

[31] T. Imielinski, B. R. Badrinath, "Wireless Computing: Challenges in Data Management," *Communications of ACM,* pp. 19-28, October, 1994.

[32] J. Ioannidis, D. Duchamp, G. Maguire, "IP-based Protocols for Mobile Internetworking," *Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols,* pp. 235-245, September, 1991.

[33] J. Ioannidis, G. Q. Maguire Jr., "The Design and Implementation of a Mobile Internetworking," *Proc. Winter USENIX,* pp. 489-500, January, 1993.

[34] K. Ishida, "Space-Time Tradeoff in Hierarchical Routing Schemes," *Responsive Computer Systems,* edited by H. Kopetze and Y. Kakuda, Springer Verlag, New York, pp. 147-163, 1993.

[35] J. M. Jaffe, F. M. Moss, "A Responsive Routing Algorithm for Computer Networks," *IEEE Transactions on Communications,* pp. 1758-1762, July, 1982.

[36] R. Jain, Y. B. Lin, S. Mohan, "A Caching Strategy to Reduce Network Impacts of PCS," *IEEE Journal on Selected Areas in Communications,* Vol. 12, No. 8, pp. 1434-1445, October, 1994.

[37] R. Jain, Y. B. Lin, "A Forwarding Strategy to Reduce Network Impacts of PCS," *Proc. IEEE INFOCOM,* pp. 481-489, March, 1995.

[38] Pankaj Jalote, *Fault Tolerance in Distributed Systems,* Prentice Hall Inc., Englewood Cliffs, New Jersey, 1994.

[39] David B. Johnson, "Routing in Ad Hoc Networks of Mobile Hosts," *Proc. of Workshop on Mobile Computing Systems and Applications,* pp. December, 1994.

[40] J. Jubin, J. D. Tornow, "The DARPA Packet Radio Network Protocols," *Proceedings of the IEEE,* Vol.75, No. 1, pp. 21-32, January, 1987.

[41] F. Kamoun, "Design Considerations for Large Computer Communication Networks," UCLA-ENG-7642, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, March, 1976.

[42] P. R. Karn, H. E. Price, R. J. Diersing, "Packet Radio in the Amateur Service," *IEEE Journal on Selected Areas in Communications,* Vol. 3, No. 3, pp. 431-439, May, 1985.

[43] K. Keeton, B. Mah, S. Seshan, R. H. Katz, D. Ferrari, "Providing Connection-Oriented Network Services to Mobile Hosts," *Proc. of the USENIX Symposium on Mobile and Location-Independent Computing,* pp. 83-102, August 1993.

[44] R. Koo, S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Transactions on Software Engineering,* Vol. SE-13, No. 1, pp. 23-31, January, 1987.

[45] P. Krishna, N. H. Vaidya, D. K. Pradhan, "Location Management in Distributed Mobile Environments," *Proc. IEEE International Conference on Parallel and Distributed Information Systems,* pp. 81-89, September, 1994.

[46] P. Krishna, N. H. Vaidya, D. K. Pradhan, "Static and Adaptive Location Management in Distributed Mobile Environment," *Computer Communications* (Special Issue on Mobile Computing), Vol. 19, No. 4, March, 1996. (Also available from http://www.cs.tamu.edu/people/pkrishna)

[47] P. Krishna, M. Chatterjee, N. H. Vaidya, D. K. Pradhan, "A Cluster-based Approach for Routing in Ad-Hoc Networks," *Proc. USENIX Symposium on Location Independent and Mobile Computing,* pp. 1-8, April, 1995.

[48] P. Krishna, Nitin H. Vaidya, D. K. Pradhan, "Recovery in Distributed Mobile Environments," *Proc. IEEE Workshop on Advances in Parallel and Distributed Systems,* pp. 83-88, October, 1993.

[49] P. Krishna, N. H. Vaidya, D. K. Pradhan, "Forwarding Pointers for Efficient Location Management in Distributed Mobile Environments," Technical Report # 94-061, Dept. of Computer Science, Texas A&M University, September, 1994.

[50] P. Krishna, N. H. Vaidya, D. K. Pradhan, "Efficient Location Management in Mobile Wireless Networks," Technical Report # 96-030, Department of Computer Science, Texas A&M University, June, 1996.

[51] W. C. Y. Lee, *Mobile Cellular Communications Systems,* McGraw Hill, New York, New York, 1989.

[52] Y. B. Lin, "Failure Restoration of Mobility Databases for Personal Communication Networks," *ACM-Baltzer Journal of Wireless Network,* Vol.1, pp. 365-372, 1995.

[53] Jean-Paul Linnartz, *Narrowband Land-Mobile Radio Networks,* Artech House, Boston, Massachusetts, 1993.

[54] C. Lo, R. Wolff, "Estimated Network Database Transaction Volume to Support Wireless Personal Data Communications Applications," *Proc. International Conference on Communications, ICC,* pp. 1257-1263, May, 1993.

[55] J. M. McQuillan, D. C. Walden, "The ARPA Network Design Decisions," *Computer Networks,* Vol.1, No.5, pp. 243-289, August, 1977.

[56] J. M. McQuillan, I. Richer, E. C. Rosen, "The New Routing Algorithm for ARPANET," *IEEE Transactions on Communications,* Vol. 28, No. 5, pp. 711-719, May, 1980.

[57] Kathleen S. Meier-Hellstern, G. P. Pollini, D. J. Goodman, "The Use of SS7 and GSM to Support High Density Personal Communications," *Proc. International Conference on Communications,* Paper 356.2, 1992.

[58] S. Mohan, R. Jain, "Two User Location Strategies for Personal Communication Services," *IEEE Personal Communications,* Vol. 1, No. 1, pp. 42-50, 1994.

[59] Shree Murthy, J.J. Garcia-Luna-Aveces, "A Routing Protocol for Packet Radio Networks," *Proc. ACM International Conference on Mobile Computing and Networking,* pp. 86-95, November, 1995.

[60] L. J. Ng, R. W. Donaldson, A. D. Malyan, "Distributed Architectures and Databases for Intelligent Personal Communication Networks," *Proc. IEEE International Conference on Selected Topics in Wireless Communication, ICWC* pp. 300-304, 1992.

[61] C. Perkins, P. Bhagwat, "Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for Mobile Computers," *Proc. ACM SIGCOMM Symposium on Communication , Architectures and Protocols,* pp. 234-244, September, 1994.

[62] C. Perkins, "IP Mobility Support," Internet Draft, IETF Mobile IP Working Group, October, 1994. (Available from http://snapple.cs.washington.edu:600/mobile/mobile_www.html).

[63] D. K. Pradhan, N. H. Vaidya, "Roll-Forward Checkpointing Scheme:A Novel Fault-Tolerant Architecture," *IEEE Transactions on Computers,* Vol.43, No. 10, pp. 1163-1174, October, 1994.

[64] D. K. Pradhan, *Fault Tolerant Computer System Design,* Prentice Hall Inc., Englewood Cliffs, New Jersey, 1996.

[65] D. K. Pradhan, P. Krishna, N. H. Vaidya, "Recoverable Mobile Environment: Design and Tradeoff Analysis," *Proc. International Symposium on Fault-Tolerant Computing,* June, 1996. (Also available from http://www.cs.tamu.edu/people/pkrishna)

[66] C. V. Ramamoorthy, A. Bhide, J. Srivastava, "Reliable Clustering Techniques for Large, Mobile Packet Radio Networks," *Proc. IEEE INFOCOM,* pp. 218-226, May, 1987.

[67] S. Rangarajan, A. Dahbura, "A Fault-Tolerant Protocol for Location Directory Maintenance in Mobile Networks," *Proc. International Symposium on Fault-Tolerant Computing,* pp. 164-173, June, 1995.

[68] M. Satyanarayanan, J. J. Kistler, L. B. Mummert, M. R. Ebling, P. Kumar, Q. Lu, "Experience with Disconnected Operation in a Mobile Environment," *Proceedings USENIX Symposium on Mobile & Location-Independent Computing,* pp. 11-28, August, 1993.

[69] M. Schwartz, T. E. Stern, "Routing Techniques used in Communication Networks," *IEEE Transactions on Communications,* pp. 539-552, April, 1980.

[70] Nachum Shacham, "Organization of Dynamic Radio Network by Overlapping Clusters: Architecture, Considerations, and Optimization," *Performance,* December, 1984.

[71] N. Shivakumar, J. Widom, "User Profile Replication for Faster Location Lookup in Mobile Environments," *Proc. ACM International Conference on Mobile Computing and Networking,* pp. 161-169, Nov., 1995.

[72] M. Spreitzer, M. Theimer, "Providing Location Information in a Ubiquitous Computing Environment," *Proc. Symposium on Operating System Principles,* December, 1993.

[73] R. E. Strom, S. Yemini, "Optimistic Recovery in Distributed Systems," *ACM Transactions on Computer Sysstems,* pp. 204-226, Aug. 1985.

[74] F. Teraoka, Y. Yokote, M. Tokoro, "A Network Architecture Providing Host Migration Transparency," *Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols,* September, 1991.

[75] R. Thomas, H. Gilbert, G. Mazziotto, “Influence of the Mobile Station on the Performance of a Radio Mobile Cellular Network,” *Proc. 3rd Nordic Seminar,* paper 9.4, September, 1988.

[76] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing and Computer Science Applications,* Prentice Hall Inc., Englewood Cliffs, New Jersey, 1988.

[77] N. H. Vaidya, “On Checkpoint Latency,” *Proc. IEEE Pacific Rim Fault Tolerant Systems*, pp. 60-65, December, 1995.

[78] H. Wada, T. Yozawa, T. Ohnishi, Y. Tanaka, “Mobile Computing Environment Based on Internet Packet Forwarding,” *Proc. USENIX Winter Conf.,* pp. 503-517, January, 1993.

[79] S. F. Wu, C. Perkins, P. Bhagwat, “Caching Location Data in Mobile Networking,” *IEEE Workshop on Advances in Parallel and Distributed Systems,* pp. 71-76, October 1993.

APPENDIX A

NETWORK EVENTS IN MOBILE WIRELESS NETWORK

This appendix presents an overview of the network events that occur in a mobile wireless network (Chapter III).

Switch ON

Each *MSS* periodically transmits a beacon identifying itself and the *registration area* it is located in. Each mobile host generates a *registration* message identifying itself when it switches on (step 1 in Fig. 61) [9]. When a *MSS* receives a *registration* message from a mobile host, it forwards the message to its location server (step 2 in Fig. 61)). The location server updates its database to indicate the current cell location of the mobile host (step 3). The location server determines the *HLS* of the mobile host from the mobile host identifier (step 4). It then informs the *HLS* of the current location of the mobile host (step 5). The *HLS* upon updating the database (step 6) sends back an acknowledgment to the location server (step 7) which in turn sends an acknowledgment to the *MSS* (step 8), thus, completing the registration process.

Switch OFF

Whenever the mobile host switches off, it sends a *de-registration* message (which includes its identifier) to the *MSS*, which forwards it to the current location server. The location server determines the *HLS* of the mobile host from the mobile host identifier. It then informs the *HLS* of the location of the host. Thus, the *HLS* knows the last location of the mobile host before the mobile host switched off.

Handoffs

The mobile host can be *idle* (not engaged in a call or data transfer) or *active* (engaged in a call or data transfer) when the host crosses cell boundary. The problem of

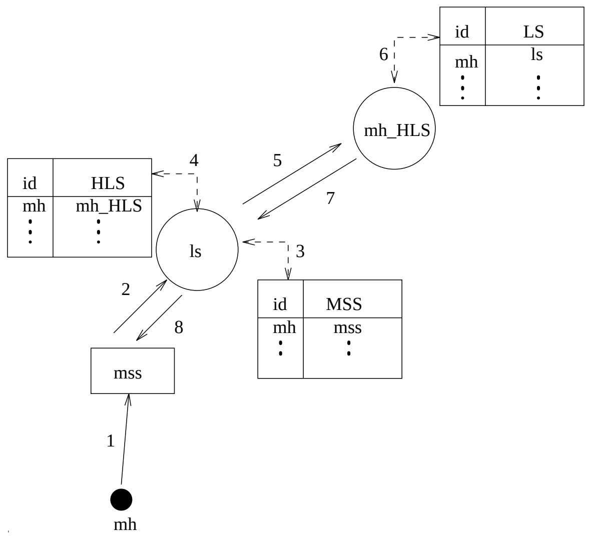


Fig. 61. Mobile Host Switches ON

active handoffs are beyond the scope of this work. Apart from location management, active handoffs involve connection management issues, e.g., connectivity, in-order delivery (in wireless ATM) etc. Here we will describe a procedure for idle handoffs [9]. Please refer to Fig. 62 for this discussion. The mobile host mh upon detecting a

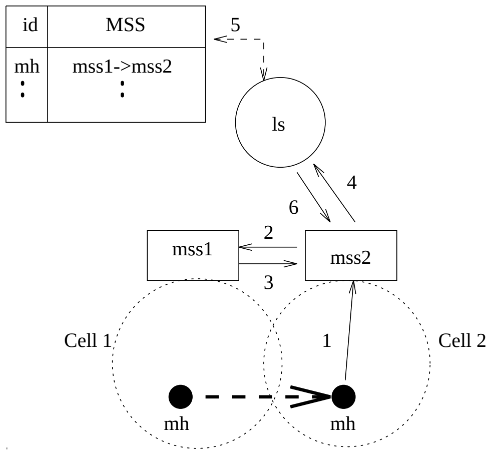


Fig. 62. Handoff

stronger signal from $mss2$ initiates the handoff procedure by sending a *handoff-init* message to $mss2$ (step 1 in Fig. 62). This message includes the identifiers of mh and the old MSS, i.e., $mss1$. The $mss2$ upon receiving the *handoff-init* message requests $mss1$ to transfer the host information (user profile, etc.) (steps 2 and 3). The $mss2$ then sends an update message to the location server (ls) of its registration area (step 4). The ls updates its database and sends back an acknowledgment (steps 5 and 6). The HLS database is not updated during handoffs.

APPENDIX B

SEARCH-UPDATES ANALYSIS FOR MOVEMENT-BASED HEURISTIC

This appendix presents the analysis of search-updates for movement-based heuristic (Chapter III). For the analysis, note the following:

- The moves are such that the number of the forwarding pointers traversed during a search never decreases due to a move.
- HLS update takes place every $\mathcal{M}$ registration area crossings.
- We assume that at any time, the maximum length of the chain of forwarding pointers will be $K = \mathcal{M} - 1$.
- We assume *Jump Updates.*
- The total number of registration areas (location servers) is N.
- The number of forwarding pointers traversed during a search decreases only due to calls. We assume that the probability of a call originating from a location server is equal for all location servers.

The state transition diagram based on above assumptions is as shown in Fig. 63. We model the length of a chain for a host at a particular location server by a markov process. State i, where, $0 \leq i \leq K$ and $i \neq \psi$, represents the state where the length of the chain at the location server is i. The state ψ is the state when the location server has no forwarding pointer for the host, hence, a search for the host originating from the location server requires a HLS query.

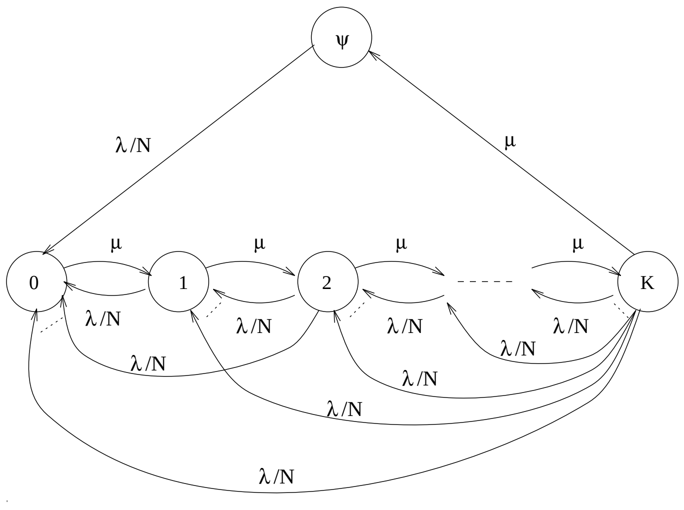


Fig. 63. State Transitions when Search-Updates are used with Movement-based Heuristic

Let the probability of the process being in state i be P_i, $0 \leq i \leq K$. Let the probability of the process being in state ψ be P_ψ. Let r be the call-mobility ratio, defined as λ/μ.

$$\mu P_0 = \frac{\lambda}{N}(P_\psi + \sum_{i=1}^{K} P_i)$$

Simplifying it we get,

$$P_0 = \frac{r}{r+N}$$

For $1 \leq i \leq K$, the state transition equation is as follows;

$$(1 + \frac{ir}{N})P_i = P_{i-1} + \frac{r}{N}\sum_{j=i+1}^{K} P_j$$

Now, the state transition equation for state ψ,

$$P_K = \frac{r}{N}P_\psi$$

The average chain length, $k' = \sum_{i=0}^{K} iP_i$, and the hit probability, $s = 1 - P_\psi$.

For $\mathcal{M} = 3$, the value of k' and s can be obtained as follows:

$$k' = \frac{Nr(3N + 2r)}{(N + r)(2r^2 + 2Nr + N^2)}$$

$$s = \frac{3rN^2 + 4Nr^2 + 2r^3}{(N + r)(2r^2 + 2Nr + N^2)}$$

For $N = 50$, Fig. 64 illustrates the variation of k' when r changes, and Fig. 65

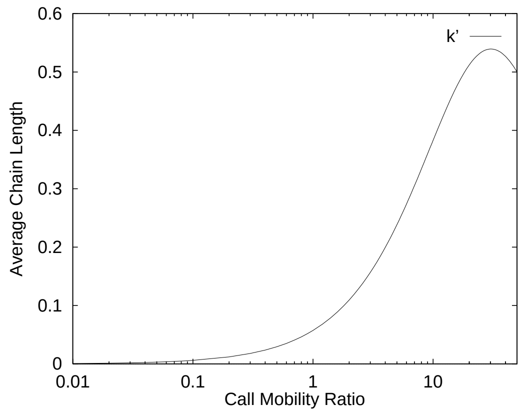


Fig. 64. Variation of k' with r for $\mathcal{M} = 3$

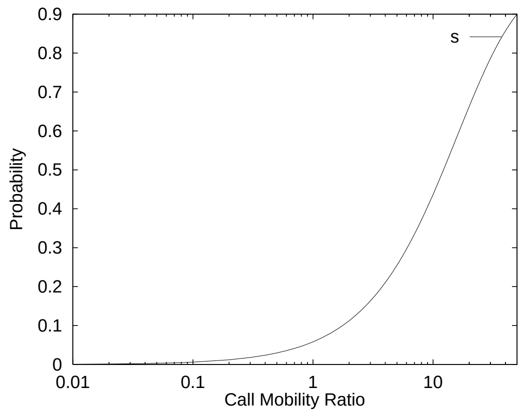


Fig. 65. Variation of s with r for $\mathcal{M} = 3$

illustrates the variation of s with r. As shown in Fig. 65, we see that s increases as r increases. This is becauses, at high values of r, calls are more frequent than moves, and since we are using *jump updates*, the probability of the location information being obtained from the location server itself is high.

APPENDIX C

SEARCH-UPDATES ANALYSIS FOR SEARCH-BASED HEURISTIC

This appendix presents the analysis of search-updates for search-based heuristic (Chapter III). For the analysis, note the following:

- The moves are such that the number of the forwarding pointers traversed during a search never decreases due to a move.
- *HLS* update takes place upon every search, i.e., $\mathcal{S} = 1$.
- We assume *Jump Updates*.
- The total number of registration areas (location servers) is N.
- The number of forwarding pointers traversed during a search decreases only due to calls. We assume that the probability of a call originating from a location server is equal for all location servers.
- The maximum chain length is equal to N, i.e., the number of registration areas in the network. In other words, at least one *HLS* update is assumed to take place before the chain length exceeds N.

The state transition diagram based on above assumptions is as shown in Fig. 66. The state equations for this state diagram are as follows:

$$P_0 = \frac{r}{r + N(1+r)}$$

For $i > 0$,

$$P_i = \frac{1}{1+r} P_{i-1}$$

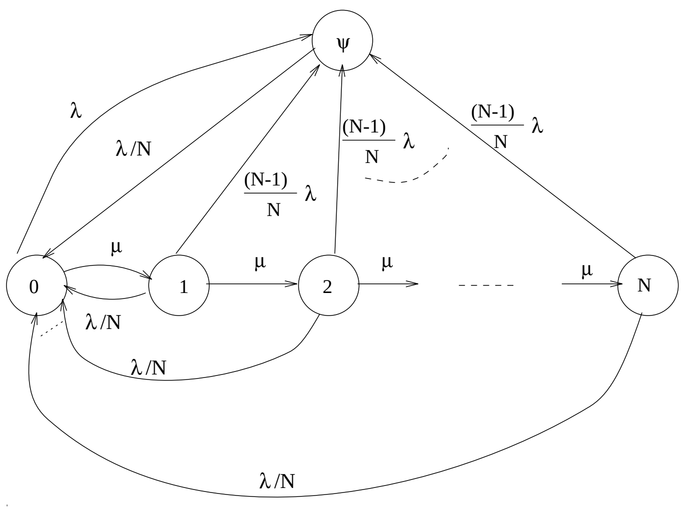


Fig. 66. State Transitions when Search-Updates are used with Search-based Heuristic

$$P_\psi = \frac{(r + N(1+r))(1+r)^N - (1+r)^{N+1} + 1}{(r + N(1+r))(1+r)^N}$$

The average chain length and the hit probability are determined as follows:

$$k' = \sum_{i=0}^{N} iP_i = \frac{(1+r)((1+r)^N + N + 1) + N}{r(1+r)^N(r + N(1+r))}$$

$$s = 1 - P_\psi = \frac{(1+r)^{N+1} - 1}{(1+r)^N(r + N(1+r))}$$

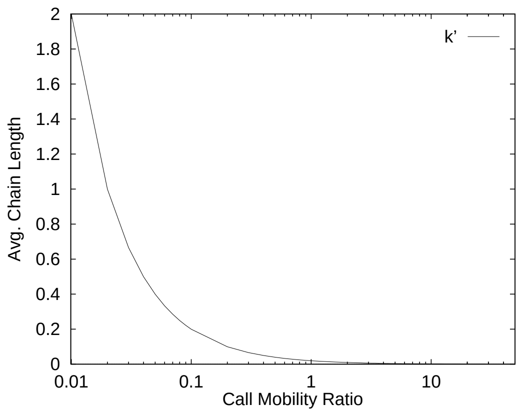


Fig. 67. Variation of k' with r (Search-based) for N=50

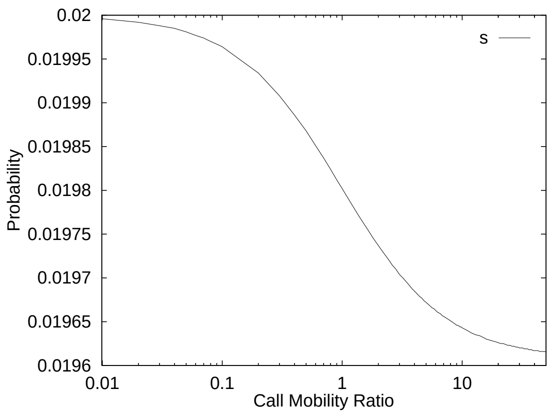


Fig. 68. Variation of s with r (Search-based) for N=50

For $N = 50$, Fig. 67 illustrates the variation of k' when r changes, and Fig. 68 illustrates the variation of s with r. Unlike the movement-based heuristic, the hit probability for search-based heuristic is very low and also reduces with increasing r. This is because in search-based heuristic an HLS update takes upon every search. The forwarding pointer maintenance policy explained in Section 3 requires that the forwarding pointers be purged during every HLS update. Thus, as r increases, the lifetime of forwarding pointer at a location server reduces, thus the hit probability reduces too.

APPENDIX D

PROOF OF CORRECTNESS

This appendix presents an outline of the proof of correctness for the algorithms presented in Chapter V.

For the sake of convenience, let us introduce two terms, namely, *root* node and *affected* node. A *root* node is a node that initiates the cluster update algorithm, whereas *affected* node is a node whose clusters may be affected by the algorithm initiated by the *root* node. For the various types of *events* listed in Chapter V, let us determine the *root* node(s) and the *affected* node(s).

- **Switch ON**: The new node is the *root* node. The neighbors of the new node are the *affected* nodes.
- **Switch OFF**: The node n that is determined using the arbitration procedure explained in Section C.2, is the *root* node. The neighbors of the node n are the *affected* nodes.
- **Connection between nodes A and B**: The node (A or B) with the larger *id* is the *root* node. The common neighbors of A and B are the *affected* nodes.
- **Disconnection between nodes A and B**: The node (say, A) with the larger *id* is the *root* node. The neighbors of the *root* node are the *affected* nodes. The node (say, B) other than the *root* node adds new clusters and does not remove any clusters.

Each algorithm comprises of the following basic steps:

- The *root* node(s) gets from each *affected* node, the *affected* node's neighbor information and its cluster set.
- The *root* node(s) determines the possible clusters using **Create Clusters** function.
- From these clusters, the *root* node(s) determines the *essential* clusters using **Find Essential** function.
- The *root* node(s) adds the *essential* clusters to list of clusters it has obtained from the *affected* nodes. The *root* node(s) then determines and removes the *redundant* clusters using the **Find Redundant** function.
- The new cluster information is then broadcast by the *root* node to the *affected* nodes.

Lemma 1: The root *node has connectivity to each* affected *node through at least one of the clusters returned by the* **Create Clusters** *function.*

Proof: Step 2 of the **Create Clusters** (Table V) function ensures that at least one cluster is created with *root* node and an *affected* node as its members. Thus, the *root* node will have connectivity to each *affected* node through at least one of the clusters. □

As shown in Fig. 69, the *root* node along with the *affected nodes* form a star graph with the *root* node at the center and the *affected* nodes at the fringes. Some of the *affected* nodes may be connected to each other, and they form a *connected* segment. On the other hand, some of the *connected* segments may not connected with other *connected* segments, and they form *disconnected* segments (e.g., A, B and C in Fig. 69(a)), R being the *root* node. In the worst case, a *connected* segment is a node (e.g., C in Fig. 69(a). In such a case, all *affected* nodes are disconnected from each other. When the *root* node R switches ON or moves into the vicinity of the nodes in A, B and C, the *root* node provides connectivity between the *disconnected* segments through it. It should thus be ensured that the *affected* nodes in the *disconnected* segments become cluster-connected after the execution of **Find Essential** function at the *root* node (as in Fig. 69(c)). This is because, even if there is a path between these *disconnected* segments through the *root* node, there may not be any path between them using clusters. This will violate the cluster-connectivity criteria.

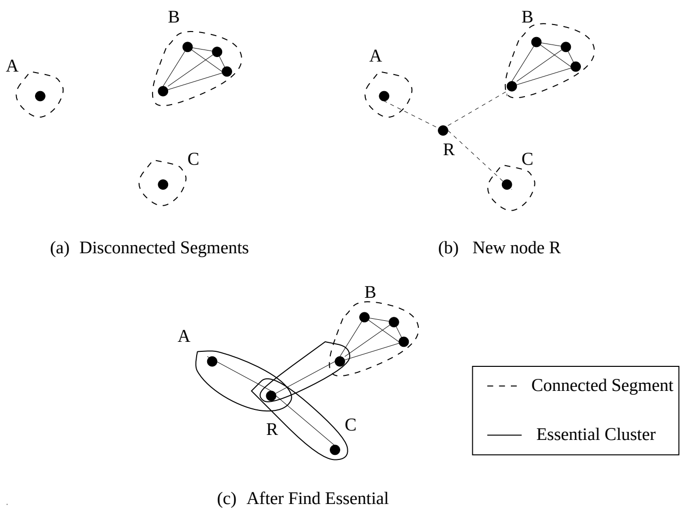


Fig. 69. Clusters formed by **Find-Essential**

Lemma 2: The affected *nodes in the different* disconnected *segments become cluster-connected after the execution of* **Find Essential** *function at the* root *node.*

Proof: Steps 4-9 of the **Find Essential** function ensures that there is an *essential* cluster between at least one node in each *connected* segment and the *root* node. Thus, after the execution of **Find Essential**, there is cluster-connectivity between the *affected* nodes in different *disconnected* segments. □

Lemma 3: Nodes that were cluster-connected before the network event occurred, will remain cluster-connected after the removal of redundant clusters by the root *node.*

Proof: The *root* node executes the **Find Redundant** function to determine *redundant* clusters. This function determines *redundant* clusters based on Definition 6, which ensures that if a cluster is *redundant*, removal of the cluster does not affect the cluster-connectivity of the graph. □

Theorem 1: Given a cluster-connected graph, the graph remains cluster-connected after any network event.

Proof: The proof follows from Lemma 1, Lemma 2 and Lemma 3. □

APPENDIX E

RANDOM GRAPH GENERATOR

This appendix presents the algorithm used to generate random graphs (Chapter V). The random graph generator is based on the 'labeling' algorithm presented in [27]. The input to this graph generator is N and D, where N is the number of nodes in the network, and D is the average degree of the network. We use a 'labeling' algorithm to generate random spanning trees with N nodes. Then we randomly add $(\frac{ND}{2} - (N-1))$ links, so that the average degree in the final network is D. The algorithm is presented in Table XIII.

Table XIII. Random Graph Generator Procedure

```
     Procedure Random_Graph_Generator($N$,$D$);
     Begin;
1.   Node list I = [1...N]; Edge list $T = \emptyset$ ;
2.   Generate a sequence $S$ of $(N-2)$ random labels in the range [1,N];
3.   while $(|S| > 0)$
4.      Look for the smallest label $i_1$ in $I$ that is not in $S$;
5.      $T = T \cup (i_1, s_1)$ ;
6.      Remove $i_1$ from $I$ and $s_1$ from $S$ ;
7.   $T = T \cup (i_1, i_2)$ ;
8.   $remaining = \frac{ND}{2} - (N-1)$ ;
9.   while $(remaining > 0)$
10.     Randomly generate 2 labels (i,j) s.t., $(i,j) \notin T$ ;
11.     $T = T \cup (i,j)$ ;
12.     $remaining = remaining - 1$ ;
     End;
```

VITA

P. Krishna

Academic Degrees:

B.S.(Hons.)

Department of Electrical Engineering

Regional Engineering College, Rourkela, 1990.

M.S.

Department of Electrical Engineering

Texas A&M University, 1992

Field of Specialization:

Computer Science

Permanent mailing address:

Apt. 2, 37/B Southern Avenue

Calcutta 700029

India