1999

Location management and fault tolerance issues in mobile networks

Govindarajan Krishnamurthi
Iowa State University

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Location management and fault tolerance issues in mobile networks

by

Govindarajan Krishnamurthi

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Major Professor: Arun K. Somani

Iowa State University

Ames, Iowa

1999

Graduate College
Iowa State University

This is to certify that the Doctoral dissertation of

Govindarajan Krishnamurthi

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

Signature was redacted for privacy.

For the Graduate College

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

This is a very difficult part to write. I have to thank many people who have helped me in various stages of life. First, I wish to thank my advisor Prof. Arun K. Somani. His guidance was invaluable to me in completing my dissertation. His advise to me as a friend as well as an advisor cannot be measured. I would also like to thank Prof. Murat Azizoglu, whose guidance is something I can always trust and count on.

My former roommate Suresh Sriramulu, who is always there to help me. His friendship is invaluable to me and I shall always cherish it. Sathyadev Uppala, is another friend of mine with whom I have spent innumerable hours discussing matters dealing with philosophy. My former labmate Prof. Suresh Subramaniam is another person with whom my discussions have been quite enjoyable. My labmates in the Dependable Computing and Networking Laboratory, made my life in school a very pleasurable one. Iam also thankful to Stefano Chessa with whom I have collaborated in research.

My sister Kala and her family has always been a source for love and affection for me. My future wife Prasanthi, whom I love very much, is a person whom I cannot forget mentioning here. Whatever little that I have achieved in life, would not have been possible, without the sacrifices made by my parents. What they mean to me cannot be put into words. I love them from the bottom of my heart. Their blessings have been a source of strength to me. I dedicate this work to them.

ABSTRACT

Personal Communication Services (PCS) networks has the capability of providing communication services to people not in the proximity of wired networks. An important issue in managing PCS networks is that of maintaining a handle on the location of mobile users. The location information of the mobile users is stored in databases distributed in the backbone wireline network. The databases are queried to locate the position of mobiles during call delivery. We develop new protocols for location management in mobile networks, with the added constraint of load balance among the databases. We then analyze the effect of failures of some these databases on the performance of the PCS network. We also propose new protocols to recover from database failures in the network.

1 INTRODUCTION

Cellular communication technology has advanced rapidly in the last decade with significant developments in the capabilities of mobile networks. The third generation Personal Communication Services (PCS) networks, will be capable of providing different types of services to a large population.

The essential ingredient of the PCS network is the presence of nodes whose position is not static. This freedom given to nodes gives rise to problems which are unique to mobile networks. This has led to several research efforts being undertaken in various research organizations. In this chapter we describe the PCS Network and various issues associated with PCS networks.

The PCS network is the state of art architecture in mobile networks. It was established for providing voice telephony for mobile users[1]. Along with this main service other related services like voice mail, call forwarding, calling number identification, call waiting and conference calls are other services which are provided by the PCS network.

1.1 PCS Network Architecture

In this section we describe the PCS Network architecture. We also describe the model we assume in the rest of this report.

Most PCS networks use a cellular architecture for bandwidth efficiency [1], [2]. In this architecture, each cell has a base station to which the mobiles of the cell communicate through a wireless link. A set of base stations is controlled by a Base Station Controller (BSC). The primary function of a BSC is to manage the radio resources of

[1]We refer to mobile hosts/users as mobiles in the rest of the dissertation.

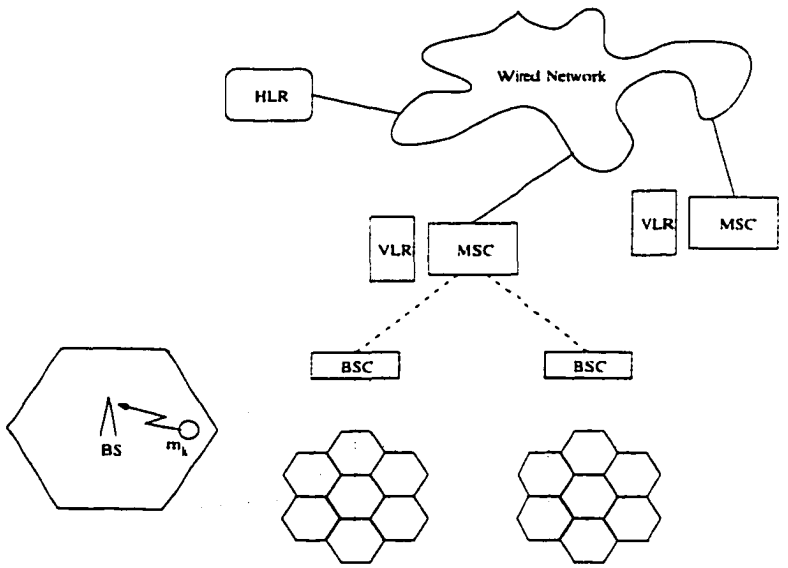

Figure 1.1 PCS Architecture based on the IS-41 standard

its base stations, by performing hand-off and allocating radio channels. Each BSC is connected to a Mobile Switching Center (MSC) through a wired network. A MSC typically provides switching functions and coordinates location registration and call delivery. The MSC has access to the location information databases in the network, which are used to store location and service information for each registered mobile of the PCS network. PCS architectures which adopt the IS-41/GSM standards ([3]/[4]) (Figure 1.1) use a two level hierarchy of such databases for location management. These are the Home Location Register (HLR) and the Visitor Location Register (VLR). The HLR is a centralized global database in which information about all mobiles registered in the PCS network is stored. The VLR is a local database usually associated with the MSC and stores information about mobiles visiting the MSC's Location Area (LA). The VLR therefore serves the purpose of a cache. Several architectures for LIDs can be found in the literature. These include both centralized and distributed architectures. Each architecture has its advantages and disadvantages. A centralized database is simpler to implement and manage. It also has a better connection delay performance, since the lo-

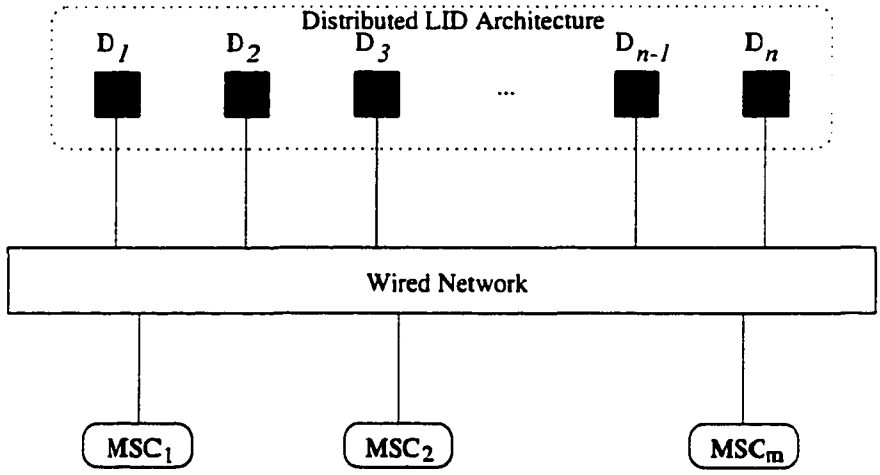

Figure 1.2 A distributed location database architecture

cation information can be accessed by a single query. However a centralized architecture is neither scalable nor fault tolerant. Furthermore as the number of mobiles increases, the increasing load on the LID negatively affects the Quality of Service (QoS) provided by the network.

A distributed database architecture provides fault tolerance, scalability and modularity, at the expense of increased control traffic and connection delay. In this report we assume the logical setup for the general distributed LID architecture shown in Figure 1.2. There are n databases with identical storage and access capabilities. These databases are connected to each other and to the MSCs through a wired network.

A single fault model (network link(s) or the HLR) is assumed in the dissertation. The HLR is assumed to be fail-stop and have a stable storage which can be recovered after a failure (this can be easily achieved using a redundant array of disks [5]). The HLR's volatile memory is periodically backed-up to the disk and the HLR logs the transactions on the disk between backups [6]. This ensures that we can retrieve the contents of the volatile memory of the HLR at the time of the HLR's failure from the contents of the disk backup and the log. Once a backup is done the log is deleted and a fresh logging process is initiated.

1.1.1 Research Issues in Mobile Networking

In this subsection we describe some of the research issues in the area of mobile networking.

- Channel Allocation Problem: The process of allocating wireless bandwidth to a mobile when the mobile wishes to communicate. Several research efforts have been undertaken to address this issue. A comprehensive survey of this field of research can be found in [7].

- Mobility Management Problem: In PCS networks, a location tracking mechanism is needed to locate the position of the mobile in order to establish connections. Current methods require a mobile to report its location to the network when necessary. Some of the location management schemes use a time-based update strategy. Other schemes require a mobile to update its location only if its present location is at least a pre-determined distance away from its location at the previous update. The network stores the location of the mobile in location-information-databases (LIDs) and this information is retrieved during call delivery. The issue of when to update, where to update and how to retrieve the location information efficiently forms the location management problem. A related issue is that of *handoffs*. When a mobile moves from one cell to another while communicating the call has to be maintained. Some research efforts in this area are [8] and [9].

- Power Issues: A mobile uses a battery as its power source. Therefore, optimal usage of this power is an issue to consider. A research effort in this area is [10].

- Fault Tolerance Issues: Issues pertaining to recovery of databases in case of link and database failures are issues which are being investigated.

- Ad-Hoc Mobile Networks: A relatively new research issue in the field of mobile networks is that of Ad-Hoc networks. These networks are devoid of base stations and routers. Mobiles in such networks communicate directly with each other. Routing

algorithms in standard networks depend on addressing schemes and the existence of routing tables in designated routers. Such algorithms are not applicable in Ad-Hoc networks due to the highly dynamic topology of the network and the high rate of mobility of the hosts. Routing protocols for such networks is an active research issue.

- Mobile IP: This research deals with adapting the traditional IP to wireless networks.

- Security: Security is an important issue as in any network. PCS networks employ data encryption techniques to protect user information and sensitive network control information against eavesdropping.

1.1.2 Factors Deciding the Merit of a PCS network

Various factors come into play in deciding the success of a PCS network. They are,

- Cost of the mobile phone: This is a very important factor in deciding the success of the network. Mobile phones take an intermediate position between simple phones and complex computers. The cost of phones have been reducing with the increasing number of service providers.

- Cost of the service provided: The goal of the PCS providers is to provide a service at a rate comparable to that provided by regular telephone companies. A plan mooted is to charge according to the mobile's location. In this scheme, a static mobile in its home area will be charged like a normal phone call. A mobile in a different area but not moving rapidly is charged like a pay phone. Only a rapidly moving mobile will be charged the present high rate. This brings down the cost and thus makes the service more popular.

- Range of services provided: As the range of services increases the appeal of the system also increases.

- Service Area: PCS promoters speak of making mobile networks universal. This is a tough goal to achieve. This is due to the number of service providers and the limitations of the wireless media to barriers like buildings and tunnels.

- Roaming: Ideally a customer should be able to use his mobile service anywhere. This requires the technology to coordinate home and visited location areas and also the cooperation between different service providers.

- Setup Time and Call Blocking: Customers like calls to be setup in times comparable to ordinary phones. Congested control channels and limited wireless bandwidth may cause long delays in setting up the call. The delay in locating the called mobile also plays a part in increasing the setup time. Some of these calls therefore may be blocked due setup time violations. Calls can also be blocked due to the unavailability of wireless bandwidth to service the request.

- Call Dropping: Call Dropping is a very undesirable feature. Call dropping occurs when no channel is available during handoff.

- Network Security: Unlike wired telephony, in PCS networks, the system has to take far more responsibility for protection against unauthorized use of the network. This is because data is transfered through air. First generation wireless networks were vulnerable to fraudulent use. Third generation wireless networks incorporate more robust authentication technology to prevent unauthorized access. Each system establishes procedures to verify that a terminal is authorized to use the system. These procedures are analogous to the password controls imposed on computer users and the personal identity numbers used in ATM machines.

We now present the standard IS-41 protocols for Location Management and Call Delivery.

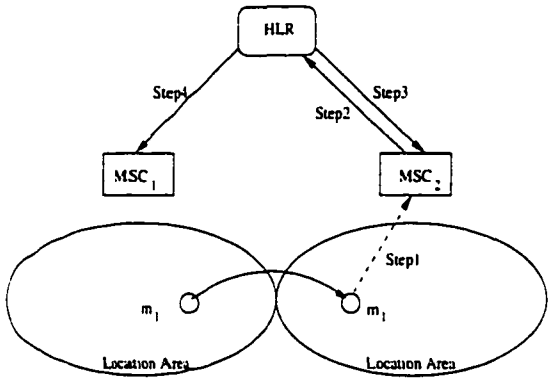

Figure 1.3 Processing a Location Update

1.2 Location Update and Call Delivery Protocols

The Location Update (LU) Protocol is used to update a mobile's position in the HLR when the mobile moves into a different LA. We describe the operation of LU and Call Delivery processes in the context of the IS-41 architecture. When a mobile m_1 moves from a LA with MSC MSC_1 to another LA with MSC MSC_2, it sends a LU to MSC_2 (Step 1 in Figure 1.3 (a)). MSC_2 delivers the message to the HLR (Step 2 in Figure 1.3 (a)). The HLR then sends a confirmation to MSC_2 on receipt of the message. The HLR also sends a message to MSC_1 which deletes m_1's entry in VLR_1 (Steps 3 and 4 in Figure 1.3). MSC_2 also updates VLR_2 with m_1's entry.

The Call Delivery Protocol is used to establish a call between two mobiles. When mobile m_1 calls another mobile m_2 it sends a Call Delivery Request (CDR) to MSC_1 (Step 1 in Figure 1.4). MSC_1 then queries the local VLR database about the position of the called mobile. If the information is retrieved MSC_1 can then establish a connection between the two mobiles. If the information is not present in the VLR, MSC_1 queries the HLR for the called mobile's information (Step 2 in Figure 1.4). The HLR determines the LA the called mobile is currently residing in and sends a route request message to the MSC of the called mobile's LA (MSC_2) (Step 3 in Figure 1.4). MSC_2 then determines a Temporary Location Directory number for m_2 (if the called mobile can receive a call)

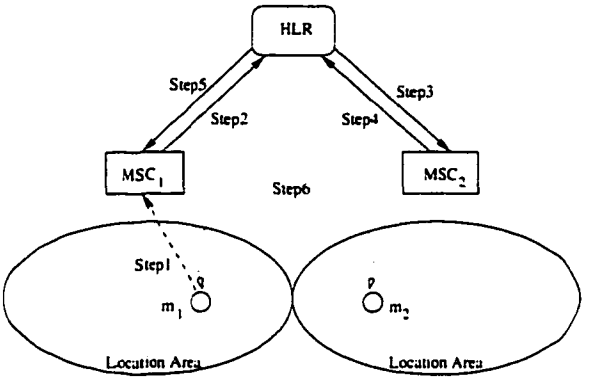

Figure 1.4 Processing a Call Delivery Request

and transfers the information to the HLR (Step 4 in Figure 1.4). The HLR delivers this information to MSC_1 (Step 5 in Figure 1.4), and establishes a connection between the mobiles (Step 6 in Figure 1.4). A similar protocol involving the HLR is used when a call originates from a fixed host. If a mobile calls a fixed host (a host in a wired network) the MSC establishes the connection with the fixed host. This operation does not depend on the state of the HLR.

1.2.1 Distributed Location Update and Query Protocols

When a mobile leaves a LA and enters another LA, it initiates a Location Update (LU) message. This message updates the position of the mobile in k of the n LIDs. Here k is the replication factor which is chosen to achieve a trade-off between the query delay and the storage costs. When a mobile places a call for another mobile, the MSC checks whether the called mobile is in its LA. This is done by querying a local database or cache if such a database/cache is available. If the called mobile is not in the same LA, the MSC queries the LIDs for the location of the called mobile. Once it receives the information, the MSC routes the call.

The research issues described in the rest of this thesis pertains to location management and fault recovery. The rest of the thesis is organized as follows. In Chapter 2, we

describe some of the previous research efforts in the field of location management and fault recovery. In Chapter 3, we derive bounds that are satisfied by any *load balanced* (we also define our notion of load balance in Chapter 3), location management algorithm. We the present a construction of a load balanced location management algorithm which achieves the optimality bounds. We present algorithms which are static, hybrid and dynamic in nature. In chapter 4, we study the factors which decide the number of replicas for a mobile's location information. In chapter 5, we derive bounds satisfied by any load balanced location management algorithm. We also study the effect of network and database faults on the performance of optimal load balanced location management algorithms described in Chapter 3. We study fault recovery issues in mobile networks in Chapter 6. We present a fast recovery protocol for database and link failures in mobile networks based on the IS-41 architecture. We then extend the protocol to the distributed architecture described in Figure 1.2. We conclude the thesis in Chapter 7 and give some directions for future research.

2 PREVIOUS RESEARCH IN LOCATION MANAGEMENT AND FAULT RECOVERY

Several schemes for location management in PCS networks have been developed in the literature. A review of some of these schemes can be found in [1] and [2]. Two schemes with a centralized database are described in [3] and [4]. In [13], tree based architecture is proposed wherein each node of the tree is a LID. Each LID contains information about all mobiles in its subtree. Another tree based distributed architecture is presented in [12]. However, unlike the architecture in [13], a mobile can be located at any node and not only at the leaf nodes. In [14], LAs which are frequented more often by a mobile are partitioned from those which are less frequented by the mobile. A LU is generated only if a mobile moves out of a partition into another partition. In [15], a dynamic distributed architecture is presented. The architecture extends the IS-41 architecture by adding another hierarchy of databases called the directory registers (DRs). Other hierarchical architectures can be found in [16], [17] and [18].

Caching schemes [19] and [20], pointer forwarding schemes [21], [22], search updating schemes [23], and local anchor schemes [24] can be used to reduce the number of updates and the search time. In [25], feedback from the network about its current load is used by a mobile in deciding on whether to update its location information. In [26] a location management scheme based on a genetic algorithm is presented.

In [27] a "per-user profile replication" for mobiles is proposed in the context of the IS-41 architecture, in which a mobile's update is replicated not only in the HLR, but also in VLR's of LA's where *frequently* calling mobiles reside.

In [28], load balancing in quorum systems is discussed and some necessary and suf-

ficient conditions for perfect balancing are given. In [29], a load balancing location management protocol is proposed, in which n databases are partitioned into subsets of cardinality of $2\sqrt{n} - 1$. These subsets are chosen such that any pair of subsets have at least one common database. Upon receiving a location update request, the MSC uses a hash function which takes into the account the mobile's ID and its current location (MSC_{id} of the MSC where the updating mobile is located) in selecting a subset for update. Upon receiving a call delivery request, the MSC uses the hash function with its own ID and the called mobile's ID to choose a subset for query. The construction of the subsets guarantees that the MSC is able to find a database with the desired location information.

In [30], a "crumbling walls" approach [31] is used to determine the updated databases. In this work, apart from a dedicated set of location databases, several general purpose servers are used for a variety of tasks including location management. During low to moderate load situations these general purpose servers perform other tasks. During high load situations, they are diverted to handle location management. [30] shows that in most load cases the cardinality of the update databases varies between $\log n - \log(\log n)$ and $n/\log n$. One drawback of this scheme is its reliance on the existence of general purpose servers which can be used at times of high load. The algorithms in [29] and [30] achieve load balance in both updates as well as queries. However, they assume the network to be fault free.

Database recovery issues are addressed in [32], [33], [35] and [34]. However, these research efforts work in the framework of the IS-41 architecture and do not address the issue of query delay in the presence of failures in a distributed database framework, that we propose to do in this thesis. In [3] during the normal operation of the HLR, the HLR's volatile memory is backed up on disk periodically. When recovering from a failure the HLR recovers the data from the non-volatile backup and sends an Unreliable Roamer Data Directive to its associated VLRs. The VLRs remove records of the mobiles associated with the HLR. Subsequently, when a mobile confirms its location by making a call or through a LU the HLR reconstructs its database in an incremental fashion. [32]

also recommends periodic location updates from mobiles to reduce the recovery period. During the time the HLR is unable to determine a mobile's location all calls to the mobile are lost. An analysis of [3] and [32] is presented in [33]. It is shown that when the system is at least moderately reliable (period between HLR failures > 100 time units) the call origination rate and the rate of LA crossing has only a minor effect on the system cost provided the frequency of the frequency of the periodic update is comparable to the value of the call inter-arrival time. The IS-41 standard [32] recovery process has the following drawbacks. A periodic location update from each mobile is sent to the HLR irrespective of the state of the HLR, requiring the usage of wireless bandwidth. This problem is magnified if the population of mobiles is large. This also results in a power drain for the mobiles. To be effective the frequency of these periodic updates must be comparable to the frequency of incoming calls. Any reduction in the recovery period can only be achieved by increasing the frequency of the periodic location update which translates to a greater use of the wireless bandwidth. Frequent update messages also increases the load on the HLR. [33] assumes that the call arrival and mobility processes are Poisson. [34] analyzes the [32] recovery process using more generalized arrival and mobility patterns.

An aggressive restoration procedure for the HLR is proposed in [35]. Once the HLR becomes functional after a failure, it requests the location information from the VLR's. To reduce the amount of information to be exchanged, the HLR requests only the information about mobiles which have moved after its last checkpoint. If the HLR is not checkpointed frequently, this approach may require a high communication overhead. The clocks of the HLR and the VLR also need to be synchronized. This approach is attractive only if very few records of information are sent from the VLR's to the HLR.

The IS-41 standard [32], recovery procedure for VLR failures is as follows. On becoming functional after a failure, a VLR either waits for the mobiles to advertise their location or waits for the HLR to inform a mobile's location through call delivery. In [35], gives schemes for checkpointed as well as non checkpointed VLRs. If the VLR is checkpointed and the VLR receives a call delivery message from the HLR, the VLR assumes

that the position of the mobile is current and tries to connect the call to the mobile. However, if the mobile is not found, the rest of the LAs in the VLR's domain is paged. If the VLR is not checkpointed, then on receiving a call delivery message all the LAs in the VLRs domain is paged. In another scheme for non-checkpointed VLR recovery described in [35], unless a mobile has advertised its position, any call delivery message to the mobile is dropped. Paging is expensive both with respect to the bandwidth used as well as time consumed. The scheme therefore is not very practical.

Other interesting research issues in the are of mobile networks are addressed in [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49] and [50].

3 OPTIMAL LOAD BALANCED LOCATION MANAGEMENT

3.1 Introduction

An important issue in the selection of location update and query protocols is *load balancing*. In a load balanced location management algorithm, the number of location updates and queries that a LID receives per unit time must be equal to the those of other LIDs *on the average*. This is a practical requirement to avoid performance bottleneck in the distributed database architecture. The load balancing requirement limits the space of update and query protocols that can be used. For example if an update protocol always chooses a subset of LIDs deterministically, it will not balance the update load. Similarly, if the query protocol searches the LID in a fixed order, say $D_1, D_2, \ldots$, it will not balance the query load. Therefore, a key notion in load balancing is the randomization of the updated LID as well as the query order. In particular we will insist that the location management policy be load balanced *at all times*. This, in turn leads to the requirement that the probability that the database D_i is selected for a given location update be $\frac{k}{n}$, where k is the number of databases the information about a mobile is stored in. While the balancing of the query load is slightly more difficult to quantify, it can be seen that the order in which the LIDs are queried must be a random permutation π of the set $\{1, \ldots, n\}$. Note that it is not necessary to have a uniform probability distribution over all such permutations, it is only necessary to balance the query load among the databases.

An interesting question that arises in this context is the existence of an optimal load

balancing location management policy. We will address this question for some practical performance metrics in the remainder of this chapter[1].

3.2 Performance Bounds

We now obtain bounds to the performance of any load-balanced algorithm for three performance metrics: worst-case query delay, average query delay and call blocking probability. The worst-case query delay refers to the maximum number of databases the algorithm should query in order to retrieve the location information about the called mobile. Average query delay refers to the expected number of databases queried. Some real time systems, however, may restrict the maximum number of queries that can be made. In such cases the call is blocked if the number of queries needed exceeds the maximum permissible by the system. The call blocking probability is used to analyze the performance of these systems.

We define the following quantities that will be used in the analysis.

- Let n be the number of LIDs in the system. Let the databases be labeled as $D_1, D_2, \ldots, D_n$.
- Let the replication factor k $(1 \leq k \leq n)$ be the number of databases a mobile's location information is updated in.
- Let X_i be a binary random variable indicating whether D_i has the location information of a given mobile. We define the storage vector as $\mathbf{X} = (X_1, X_2, \ldots, X_n)$.
- Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be the query order of the databases for accessing the location information of a given mobile. π is a permutation of $\{1, 2, \cdots, n\}$, and D_{π_j} is the jth database to be queried if the location information has not been found in $D_{\pi_1}, D_{\pi_2}, \ldots, D_{\pi_{j-1}}$ (while this implies a sequential search, the databases could be searched in parallel).

[1] A portion of this chapter was published as [51]

- Let Y_i be the binary random variable indicating whether the ith queried database has the required information, i.e. $Y_i = X_{\pi_i}$.
- Let T denote the number of queries to find the location information. We refer to T as the *query delay*. It is implicitly assumed that each database query results in a constant time delay, which is normalized to unity.

3.2.1 Lower Bound to Worst-Case Query Delay

In this section, we obtain a lower bound to the worst-case query delay of any load-balanced location management algorithm.

From the above definitions, the storage vector is a random binary n-vector with Hamming weight k, i.e. $\sum_{i=1}^{n} X_i = k$ (a mobile is updated in k LIDs). Since $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_n)$ is a permutation of the storage vector $\mathbf{X}$, $\mathbf{Y}$ is also a binary n-vector with Hamming weight k. Using the definition of Y_i, the query delay can be written as

$$T = \min\{i : Y_i = 1\}.$$

A load-balanced update mechanism must satisfy $E[X_i] = E[X_j]$, for all i, j. In conjunction with the Hamming weight constraint, this implies that

$$P(X_i = 1) = \frac{k}{n} \qquad i = 1, 2, \ldots, n. \tag{3.1}$$

Since $\mathbf{Y}$ is a permutation of $\mathbf{X}$, we also have

$$P(Y_i = 1) = \frac{k}{n} \qquad i = 1, 2, \ldots, n. \tag{3.2}$$

Note that a location update algorithm may be viewed as a probability distribution on the storage vector $\mathbf{X}$, while a query algorithm is a probability distribution on π. Thus a load-balanced update must satisfy the condition given in (3.1) on the marginal (Bernoulli) distribution. The proposition below provides a lower bound to the maximum value of the query delay T.

Proposition 1: *For any load-balanced update algorithm and any query algorithm, if j is an integer such that $P(T > j) = 0$, then $j \geq \lceil \frac{n}{k} \rceil$.*

Proof: Let j be an integer such that $P(T > j) = 0$. Then

$$P(\sum_{i=1}^{j} Y_i \geq 1) = 1 \,. \tag{3.3}$$

The event in (3.3), is the union of the events $\{Y_i = 1\}$, $i = 1, 2, \ldots, j$. Then by union bound

$$1 \;\leq\; \sum_{i=1}^{j} P(Y_i = 1) = \frac{jk}{n} \tag{3.4}$$

where we have also used (3.2). The claim follows follows from the last inequality. ■

It follows from this proposition that the worst-case delay T_{max} of any load-balanced location update algorithm is at least $\lceil \frac{n}{k} \rceil$. We will show in Section 3.3 that this bound is achievable.

3.2.2 Lower Bound to Average Query Delay

In this section, we obtain a lower bound to the expected value of the query delay for any load-balanced algorithm.

Proposition 2: *Given a system with n databases and a replication factor k, the average query delay $E[T]$ for a load-balanced location management algorithm satisfies $E[T] \geq \frac{1}{2}(\alpha + 1)(1 + \frac{\beta}{n})$, where $\alpha = \lfloor \frac{n}{k} \rfloor$ and $\beta = n - k\alpha$.*

Proof: Let $p_j = P(T = j)$ for $j = 1, 2, \ldots, n$. (Note that the actual span of the delay distribution is no more than $n - k + 1$, and no less than $\lceil \frac{n}{k} \rceil$. It is notationally convenient to extend this span to n.) We then have

$$\begin{aligned} p_j &= P(\sum_{i=1}^{j-1} Y_i = 0, Y_j = 1) \\ &\leq P(Y_j = 1) \\ &= \frac{k}{n} \end{aligned}$$

for any load-balanced algorithm. Next we construct the following linear program in the variables $(p_1, p_2, \ldots, p_n)$.

$$\Theta = \min \sum_{j=1}^{n} j p_j$$

subject to

$$0 \leq p_j \leq \frac{k}{n} \qquad j = 1, 2, \ldots, n$$

$$\sum_{j=1}^{n} p_j = 1 \, .$$

The objective function is minimized by the probability distribution

$$\begin{aligned} p_j &= \frac{k}{n} \qquad j = 1 \, , \, 2, \ldots, \alpha \\ p_{\alpha+1} &= \frac{\beta}{n} \qquad j = \alpha + 1 \\ p_j &= 0 \qquad j > \alpha + 1 \end{aligned}$$

with the resulting value

$$\Theta = \frac{1}{2}(\alpha + 1)\left(1 + \frac{\beta}{n}\right) \, . \tag{3.5}$$

Since $E[T] \geq \Theta$ for any load balancing algorithm, the result is established. ■

3.2.3 Lower Bound to Call Blocking Probability

In a large PCS network supporting real-time traffic, it may not be acceptable to have large delays in establishing connections. In such a situation, it may be necessary to block a connection request once the normalized query delay has exceeded a certain threshold m. We now obtain a lower bound to the call blocking probability P_B with any load-balanced location management algorithm.

Proposition 3: *For any load-balanced location management algorithm with a query delay threshold m, the blocking probability P_B satisfies*

$$P_B \geq \left(1 - \frac{mk}{n}\right)^+$$

where $a^+ = \max(0, a)$.

Proof: We use the upper bounds on the delay distribution that were found in the proof of Proposition 2 to obtain

$$\begin{aligned} P_B &= P(T > m) \\ &= 1 - \sum_{j=1}^{m} p_i \\ &\geq 1 - \frac{mk}{n} . \end{aligned} \tag{3.6}$$

■

Note that the lower bound is nontrivial only if $m \leq \lfloor \frac{n}{k} \rfloor$. We will show in the next section that no blocking occurs for higher thresholds. More generally, we will find that the lower bound to the blocking probability is achievable.

3.3 A Construction for Optimal Location Management

There are two main issues in the design of a load balancing location management algorithm. First, given the number of LIDs n and the replication factor k, an ideal algorithm should achieve the best possible performance. Second, for a given n, the replication factor k should be chosen so as to minimize an appropriate cost function. This cost function must balance the tradeoff between storage and performance requirements.

The lower bounds obtained in the previous section for three important performance metrics point out the limits to achievable performance with a given (n, k). We now construct a simple algorithm that meets these bounds exactly.

Update Strategy: For a given (n, k) let us write

$$n = k\alpha + \beta$$

where $\alpha = \lfloor \frac{n}{k} \rfloor$ and $\beta = n - k\alpha$. We view the databases $D_1, D_2, \ldots, D_n$ to be arranged on a logical ring[2]. Let the first database for update be selected randomly, using a uniform

[2]This topology does not impose any constraints on the physical topology. The databases can be connected via an arbitrary physical topology, such as the telephone network.

distribution over the n databases. The remaining $k - 1$ databases to be updated are selected deterministically, at "distances" α and $\alpha + 1$ from the previous updated database on the logical ring. The "distance" is defined on the logical ring, i.e. D_{i+1} is at a distance of 1 from D_i. Let $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_k)$ be the placement vector, where $\gamma_i \in \{1, 2, \ldots, n\}$ is the index of the ith database updated by the algorithm. We define the displacement vector $\mathbf{a} = (a_1, a_2, \ldots, a_k)$ as a binary vector with Hamming weight β. Having selected the first database index γ_1 uniformly, the update strategy selects the remaining $k - 1$ databases as

$$\gamma_{i+1} = \gamma_i \oplus (\alpha + a_i), \qquad i = 1, \ldots k - 1 \tag{3.7}$$

where $\oplus$ denotes modulo n addition defined over the set $\{1, 2, \ldots, n\}$.

The displacement vector $\mathbf{a}$ can be fixed a priori, or it can be randomly generated at each update. As long as its Hamming weight is β, the resulting update algorithm possesses the optimality properties which will be obtained below. Note that when k is a divisor of n, the algorithm places the location information at equidistant databases on the logical ring with a random "phase".

Figure 3.1 depicts a sample outcome of this update algorithm with $n = 10$ and $k = 3$. The shown update could occur, for example, with $\mathbf{a} = (0, 1, 0)$ and $\gamma_1 = 4$.

The MSC, when performing the update for a mobile, also deletes the previously updated location information of the mobile. The MSC can learn about the databases with the previous location information from the mobile[3]. Note that the same set of databases cannot be used repeatedly for updates, as some of the mobiles may be queried more frequently or may require more updates than others. In this case the databases which store information about these active mobiles may have a higher load than databases which store information of less active mobiles. The MSC therefore generates a fresh random number to decide the first database for update upon receiving a location update request. The pseudo-code of an Update Strategy which starts at a random LID and updates $k - \beta$ databases at intervals of $\lfloor \frac{n}{k} \rfloor$ LIDs and β LIDs at intervals of $\lceil \frac{n}{k} \rceil$ databases is given in

[3]In the worst case it might be necessary to query the LIDs for the information.

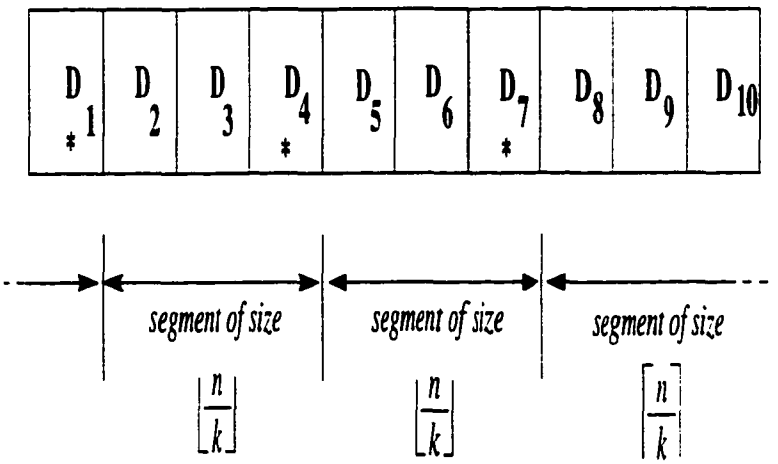

Figure 3.1 An example of the Update Strategy with $n = 10$, $k = 3$ ("*" indicates updated databases).

Table 3.1.

Query Strategy: Our query strategy starts at a randomly selected database and searches the databases contiguously until the location information is found. The algorithm chooses the initial database index π_1 using a uniform distribution on $\{1, 2, \ldots, n\}$ and employs a sequential search with order $D_{\pi_1}, D_{\pi_{1\oplus 1}}, \ldots, D_{\pi_{1\oplus(n-1)}}$. The pseudo-code of the Query Strategy is given in Table 3.2. Note that the search in the pseudo-code extends to only $\lceil \frac{n}{k} \rceil$ databases, a fact which will be proved in Proposition 5 below. To accommodate the presence of database faults, we can modify the query strategy to use more queries than the upper limit $\lceil \frac{n}{k} \rceil$. Similarly, we can set the maximum query delay to a value less than $\lceil \frac{n}{k} \rceil$, in which case some of the calls may be blocked.

Proposition 4: *The proposed location management algorithm is load-balanced for any displacement vector* **a** *with Hamming weight* β.

Proof: By unfolding the recursion in (3.7), one finds

$$\gamma_{i+1} = \gamma_1 \oplus m_i, \qquad i = 1, \ldots, k-1 \tag{3.8}$$

Table 3.1 Pseudo-code for Update Strategy

```
Update(m_i : mobile id);
    γ1 : integer;
    count : integer;
    n : integer; /* Number of LIDs */
    k : integer; /* Replication Factor */
    β : integer;
begin
  β = n − k⌊n/k⌋;
  γ1 := prev_starting_LID; /*from mobile or search */
   DELETE old information from LIDs;
  γ1 := uniform_random[1, n]; /*new starting LID */
     count := 0;
     while count ≤ k − β
      UPDATE D_{γ1⊕(count*⌊n/k⌋)};
      count := count + 1;
     done;
     count := 1;
     while count ≤ β
      UPDATE D_{(γ1⊕(k−β)⌊n/k⌋⊕(count*⌈n/k⌉))};
      count := count + 1;
     done;
end
```

where $m_i = i\alpha + \sum_{j=1}^{i} a_j$ are distinct integers in $\{1, \ldots, n\}$. The probability that any database D_i contains the location information is then obtained as

$$\begin{aligned} P(X_i = 1) &= P\Big(i \in \{\gamma_1, \gamma_1 \oplus m_1, \ldots, \gamma_1 \oplus m_{k-1}\}\Big) \\ &= P\Big(\gamma_1 \in \{i, i \oplus (n - m_1), \ldots, i \oplus (n - m_{k-1})\}\Big) \\ &= \frac{k}{n} \end{aligned} \tag{3.9}$$

which shows that the update strategy is load-balanced.

As each database is equally likely to contain the desired information and each database is equally likely to be the starting point of the query sequence, the query algorithm is load-balanced as well. ■

Proposition 5: *The worst-case query delay for the proposed location management*

Table 3.2 Pseudo-code for Query Strategy

```
Query(m_i : mobile id);
    π_1 : integer;
    count : integer;
    n : integer; /* Number of LIDs */
    k : integer; /* Replication Factor */
begin
  π_1 := uniform_random[1,n]; /* 1st LID queried */
  count = 0;
  while count < ⌈n/k⌉
    if (info in D_{π_1 ⊕ count})
      return info;
    else
      count := count + 1;
  done;
end
```

algorithm is $T_{max} = \lceil \frac{n}{k} \rceil$.

Proof: We consider the cases $\beta = 0$ and $\beta \geq 1$ separately. For $\beta = 0$, consecutive updates are separated by $\alpha = \frac{n}{k}$ databases, and the query strategy has to query no more than α databases. For $\beta \geq 1$, the number of queries is at most $\alpha + 1 = \lceil \frac{n}{k} \rceil$. Thus, in general, $T_{max} = \lceil \frac{n}{k} \rceil$. ■

Proposition 5 shows that the location management algorithm meets the lower bound of Proposition 1, and hence is optimal with respect to the worst case query delay. The next two propositions establish analogous results in average query delay and call blocking probability.

Proposition 6: *The proposed location management algorithm is optimal with respect to the average query delay.*

Proof: Let T be the query delay, and let $p_i = P(T = i)$ be the delay distribution for the proposed algorithm. By Proposition 5, $p_i = 0$ for $i > \alpha + 1$. For $1 \leq i \leq \alpha$, we have $p_i = \frac{k}{n}$. This is because in order to need exactly i queries to find the information,

the first query had to have been made i locations before an updated database. For each i, there are k such locations. However to require $\alpha + 1$ queries to reach the information, there are only β databases where the first query can be made. Therefore $p_{\alpha+1} = \frac{\beta}{n}$. Hence,

$$\begin{aligned} E[T] &= \sum_{i=1}^{\alpha} i \frac{k}{n} + (\alpha + 1) \frac{\beta}{n} \\ &= \frac{1}{2}(\alpha + 1)\left(1 + \frac{\beta}{n}\right). \end{aligned} \tag{3.10}$$

By Proposition 2 the optimality of the algorithm with respect to the average query delay is established. ■

It is shown in the Appendix that when a fully random update and query algorithm is use, the average query delay is $\frac{n+1}{k+1}$. This delay is greater than the average query delay shown in (3.10) by a factor 2 for large n and k.

Proposition 7: *For a network which blocks connections when the query delay exceeds a given threshold m, the proposed location management algorithm is optimal with respect to the call blocking probability.*

Proof: Let $P_B = P(T > m)$ be the call blocking probability. From the proof of Proposition 6, we have for $m \leq \alpha$

$$P_B = 1 - \sum_{i=1}^{m} p_i = 1 - \frac{mk}{n}. \tag{3.11}$$

For $m > \alpha$, $P_B = 0$. Hence the blocking performance of the algorithm meets the lower bound of Proposition 3, establishing its optimality. ■

Some networks may require the optimal replication of updates for a given performance level. Such an optimization will require the determination of the relative costs of storage vs. delay/blocking performance. Let the cost function be of the form $C = aS + Q$, where S is the storage cost, Q is the query cost, and a is the relative cost factor. We now find the replication factor k that minimizes the cost C for a given blocking probability P_B. Using (3.11) we can write the cost function as

$$C = ak + (1 - P_B)\frac{n}{k} .$$

The value of k which minimizes the cost is

$$k = \sqrt{\frac{(1 - P_B)n}{a}} . \tag{3.12}$$

Thus the optimal replication factor and the minimum cost are both $O(\sqrt{n})$. Similarly we can show using Propositions 1 and 2, that the replication factor which minimizes the cost using worst-case delay and average delay as the performance metric are also of $O(\sqrt{n})$.

3.4 Static and Hybrid Location Management Algorithms

In the previous section, we described a dynamic load balancing algorithm. In this section, we consider two other classes of load balancing *static* and *hybrid* load balanced algorithms. In a static update scheme where the mapping between the mobiles and the databases is fixed load balance is not possible, unless particular care is taken in distributing the location information of mobiles among the databases. The advantage in such a scheme is the low query delay in locating a database with a mobile's location information. In dynamic update algorithms the mapping between the mobiles and the databases where they are to be updated varies with time. Though such algorithms are load balanced, the search delay is high. The load on databases, though balanced, is also high on each database.

Balancing the load among databases can be done in two ways. If the query and update load offered by each mobile is known then the mobiles can be distributed among the databases in such a way that the load is balanced over the databases. However, if load information is not available we can distribute the mobiles location information among the databases and subsequently relocate some of the mobiles if any database becomes overloaded (update and query load over a predefined value). The two schemes

described in the following section are tuned to these two scenarios. We now define some quantities which we will use in the rest of the section.

- Let the number of databases in the network be n. We assume that they are numbered $D_1, \ldots, D_n$. The numbering scheme places no restrictions on how the databases are actually distributed in the network.
- The number of mobiles in the network is M.
- The number of databases each mobile is updated in (replication factor) is k, a network-wide constant.
- Let the call arrival rate of mobile m_i be c_i.
- Let the rate of updates generated by mobile m_i be u_i.
- Let w_i be the load due to a mobile m_i, $1 \leq i \leq M$, on database D_j, $1 \leq j \leq n$, when m_i is updated in D_j.

3.4.1 Static Load Balancing

When prior statistics (mobility or update rates and call arrival rates) of mobiles are known to the system, it may be possible to determine the databases where a mobile's information is stored such that the load is balanced. We show in this section that the problem of distributing mobile's information over databases such that the overall load on the databases is balanced is NP-hard. We then present a heuristic which is near optimal.

In this approach, each MSC is informed about the mapping between mobiles and databases. This presents the ideal scenario as the load is balanced among the databases and the query delay to locate a database with the required mobile's information is also minimum (i.e. one query). The average load on the databases is also a minimum as a database never receives queries for mobiles not updated in it. This, therefore represents the ideal load balancing algorithm with both the average load and the query delay being

minimum. We refer to this as the *Ideal Static (IS) Algorithm* in the rest of this section. As the number of mobiles increases however, the amount of data that needs to be stored at each MSC is very large, and hence may not be feasible. We therefore use a cache at each MSC to store information about mobiles to which calls originate in the MSC's LA. The advantage of this cache scheme is that the caches are always consistent.

Query Strategy: On receiving a call delivery request, the MSC of the calling mobile checks its cache to determine whether the called mobile's entry is present. If the entry is present (i.e the set of databases in which the mobile's location information is updated is present), it queries one of the k databases with the information selected at random, and retrieves the information. If the entry is not present in the cache, then the MSC chooses a database at random (with each database having the same chance of being picked) and begins to query sequentially starting from the picked database until the mobile's information is found. Once the information is retrieved, the MSC establishes the connection with the called mobile if possible. It also proceeds to collect information about the other $k-1$ databases with the called mobile's information to be used in case of a future query. This can be avoided however if the mapping information is stored in the mobile and is transfered to the MSC as part of the call establishment process.

Update Strategy: When a mobile moves from one LA into a new LA, it initiates an LU. If the mobile knows the set of k databases its location information is updated in, then it can transfer the knowledge to the MSC which could update the mobile's new location information in the databases. However, if such the mobile does not know its database set the MSC checks whether the information is present in the cache. If so, the MSC updates the information in the databases. Otherwise, the MSC locates the k databases using the previously defined query strategy to locate the databases and then updates the information.

The initial distribution of the mobiles among databases is done based on the notion

that a database gets queries only for mobiles updated in the database. This assumes that the entire mapping between mobiles and the databases they are updated in is available at the MSCs. By using a cache, we deviate from this assumption. While searching for a mobile not in its cache, the MSC may query databases not having the required mobile's location information. This adds extra load on these databases. Therefore, the average load on databases is higher than what is expected from the IS algorithm. As the size of the cache increases the difference between the optimal average and the observed load on the database becomes smaller.

Proposition 1 *The mapping of mobiles to databases such that the load on the databases is balanced is NP-hard.*

Proof: Let the load on database D_j be $W(j)$. $W(j)$ is given by

$$W(j) = \sum_{i \,:\, m_i \text{ is updated in } D_j} w_i \,.$$

The load offered by a mobile to the database it is updated in w_i, is given by $w_i = u_i + \frac{c_i}{k}$. This is because whenever a mobile sends its update, the update is received by all the databases in which the mobile is replicated in. The query load however is shared by the k databases which store its update. The problem reduces to the determination of a mapping between mobiles and the databases which minimizes $\max_{1 \le j \le n} W(j)$. This problem is NP-hard as shown below.

Consider the case when $k = 1$. In this case each mobile is updated in just one database. This is a variation of the Multiprocessor Scheduling problem : Given M tasks with processing times w_i, $1 \le i \le M$ and n processors, schedule tasks on the processors by assigning one processor to each task such that the processing time is minimized. This Multiprocessor Scheduling problem is NP-hard [52]. Since $k = 1$ is a special case of the problem at hand the proposition is proved. ■

3.4.2 A Mapping Heuristic

In this section, we present a heuristic which maps the mobiles to the databases and bound the worst-case average database load of the heuristic.

- Sort the mobiles in the non-increasing order of their loads, and consider them in the sorted order.
- For the mobile under consideration choose the k_i least loaded databases. Here k_i is the replication factor of mobile m_i. (We assume that $k_i = k$, $1 \leq i \leq M$ in our numerical results. However, the heuristic works with arbitrary values of k_i.)
- Update database loads, and repeat until all mobiles are allocated a database set.

The complexity of the first step of the heuristic is $O(m \log m)$ [53]. By using Fibonacci Heaps it can be shown that the amortized cost of the second step is $O(k + \log n)$ [53] where k is $O(\sqrt{n})$ [51]. We have the complexity of the heuristic therefore as $O(m(\log m + k + \log n))$. The advantage of this scheme is that once the mobiles are distributed we have near optimal load balance. If a mobile's entry is in cache, the location information of the mobile can be located among the databases in one query. This is to be compared with the $O(\sqrt{n})$ average number of queries for the dynamic load balancing algorithm [51], which achieves perfect load balance.

Proposition 2: *The performance of the heuristic is within twice the optimal performance*

Proof: The optimal load on each database for a load balanced algorithm is $\geq \frac{\sum_i k_i w_i}{n}$, $1 \leq i \leq M$. Therefore if L^o_{max} represents the maximum loaded database then we have

$$L^o_{max} \geq \frac{\sum_i k_i w_i}{n}, \ 1 \leq i \leq M \tag{3.13}$$

and

$$L^o_{max} \geq w_{max} \tag{3.14}$$

where $w_{max} = \max_i w_i,\ 1 \leq i \leq m$. Let L^h_{max} and L^h_{min} represent the load of the most and least heavily loaded database, respectively, when using the heuristic. We then have

$$L^h_{max} - L^h_{min} \leq w_{max} \tag{3.15}$$

and

$$L^h_{min} \leq \frac{\sum_i k_i w_i}{n}, 1 \leq i \leq M. \tag{3.16}$$

From (3.13), (3.14), (3.15), (3.16) we obtain the result. ■

3.4.3 Hybrid Load Balancing

The heuristic described in the previous section cannot be used unless statistics about each mobile is available at each MSC. In this section we describe a scheme which functions when such information is not available. The update strategy in this scheme is as follows.

Update Strategy: The update scheme is similar to the update scheme of [51] described in Section 3.3. However it differs from the one in [51] in that the mapping between mobiles and databases is maintained here. This implies that when a mobile's location information is updated, it is done so in a static set of databases. On receiving a LU request from a mobile the MSC tries to locate the mobile's entry in its cache. If the entry is present then the MSC updates the mobile's new location information in the databases in which the mobile is currently updated in. If the mobile's entry is not present in cache then the MSC initiates a query similar to the one described in [51] to locate a database with the information. Once the information is retrieved the MSC updates the mobile's new location information in the databases. The pseudo-code for the Update Strategy is given in Table 3.3.

Query Strategy: To exploit the update strategy described, we use a cache associated with each MSC to store mapping information of mobiles called from the MSC's LA.

Table 3.3 Pseudo-code for Update Strategy of Hybrid Algorithm

```
Update(m_i : mobile id);
    start : integer;
    count : integer; /* A counter */
    n : integer; /* Number of Databases */
    k : integer; /* Replication Factor */
    β : integer;
begin
  β = n − k⌊n/k⌋;
  start := prev_starting_database; /*obtained from the mobile or cache or by search */
    if β = 0 then
      count := 0;
      while count < k
        UPDATE D_{start⊕(count*n/k)}
        count := count + 1;
      done;
    else
      count := 0;
      while count ≤ k − β
        UPDATE D_{start⊕(count*⌊n/k⌋)};
        count := count + 1;
      done;
      count := 1;
      while count ≤ β − 1
        UPDATE D_{(start⊕(k−β)⌊n/k⌋⊕(count*⌈n/k⌉))} ;
        count := count + 1;
      done;
end
```

For such mobiles the query algorithm needs only one query to find the database with the location information. On receiving a call delivery request, the MSC of the calling mobile determines whether the called mobile's entry is in its local cache. In the event that the called mobile's entry is present in cache the MSC chooses one of the k databases randomly (with each database having the same chance of being picked) to query. If the mobile's entry is not in cache, the MSC uses the query strategy of [51] described in Section 3.3. The MSC then updates the cache with the called mobile's entry. In this paper the cache is used primarily to reduce the delay in locating the database with the

mobile's information. The functionality of the cache can however be increased as suggested in [20] to further enhance the performance of the algorithm. The pseudo-code of the Query strategy is given in Table 3.4.

The network fixes the maximum number of updates and queries a database can receive in a certain amount of time. We call this as the *threshold* for each database. Due to the uneven distribution of highly active mobiles over the databases some of the databases will receive a higher share of the load. This might lead to threshold violations (state of being overloaded) for these databases. We now describe the procedure that the hybrid algorithm adopts to deal with threshold violations.

Dealing with Threshold Violations: Before describing the threshold violation heuristic we wish to point out the following. By the nature of the updating scheme once the starting database is chosen for update the other $k-1$ updates are deterministic. Therefore, the updating scheme partitions the databases into database subsets of size k such that if one database in a subset receives an update from a mobile, every database in the subset receives an update from the mobile.

Since the mapping between mobiles and databases is not dynamic, some of the databases may become temporarily overloaded due to the random nature of a mobile's movement and call arrival pattern. Once a database becomes overloaded, it sends a request to the MSC of its most active mobile and requests the relocation of the mobile's location information. The MSC broadcasts a message to the databases in the system requesting the load information. The MSC deletes the mobile's entry from the subset of databases and uses a first fit heuristic to choose a subset for update, i.e., the MSC chooses the first subset that does not become overloaded after receiving an update. If the MSC finds no subset of databases which can take on the extra load then the threshold is increased.

Choice of Threshold: An interesting question that arises is the choice of the threshold value. By having a very high threshold we ensure that the chance of a database

being overloaded is less. A high threshold implies that the databases are allowed to tolerate larger loads. This increases the query and update processing delays. We thus start with a small threshold value and increase it when it becomes apparent that the system is unable to function at that threshold, i.e., there relocating MSC does not find a subset where the mobile can be relocated into. The value of the threshold thus decides the frequency that mobiles need to be relocated from one database to another. The higher the value the greater the chance that the mapping between mobiles and databases is static.

Cache Inconsistency: As described earlier, on receiving a call delivery request, the MSC of the calling mobile first checks whether the called mobile's entry is present in its cache. If the mobile's entry is present then it chooses one of the k databases with equal probability as the database to query. In the algorithm described, a cache inconsistency occurs in the event of the called mobile being relocated from the set of databases indicated in the cache. In this event the MSC uses the query strategy of [51] described in Section 3.3 to locate the database with the mobile's entry. The MSC then updates the called mobile's entry in its cache.

Cache Size: Cache size is another important issue to consider in designing the architecture. A higher cache size improves the query delay in the system but in a system with high load increasing cache load could cause to an increase in the query delay due to cache inconsistencies.

We analyze the performance of the two algorithms described in this section as well as the dynamic algorithm [51] described in Section 3.3 and present a comparison between the algorithms in the next section.

3.5 Performance Comparison

In this section, we compare through simulation the performance of the hybrid schemes with respect to average load on each database and average query delay. The simulation

was run on Sparc Ultra 1 machines. The simulated network consists of 20 databases and 100 Location Areas. We assumed 5000 mobiles in the system. Calls are assumed to arrive with an exponential rate uniformly varying between 5 and 10 arrivals per unit time. Each mobile is assumed to move from one LA to another with an exponential rate uniformly varying between 0.1 and 1 movements per unit time. When a mobile moves, each LA has an same probability of being the new LA. In the simulation we do not assume that a mobile knows the databases in which its location information is updated.

3.5.1 Average Query Delay

The delay in locating the database with a mobile's information is affected by many criteria such as the cache size, the distribution of the updates among the databases and the threshold level.

The dynamic algorithm does not use caches and achieves perfect load balance on the average. The average query delay in the scheme is shown to be $\frac{n+k}{2k}$. The IS algorithm has a query delay of unity. We use these values as a figure of comparison for the performance of the static scheme and the hybrid load balancing scheme described in the previous section.

Increasing Threshold: The performance of the three algorithms is presented in Figure 3.2. As the acceptable threshold increases the average query delay decreases for the hybrid scheme. This is so because as the threshold increases the mapping between databases and mobiles approaches a static mapping. The average query delay for the static scheme is higher than that of the hybrid or the fully dynamic scheme. This is because the cache size is very low to exploit the locality of reference in calls. This coupled with the non-regular placing of the updates among the n databases in the static scheme causes the average query delay is high (the worst-case number of queries is $n - k + 1$).

Increasing Cache Size: Increasing the cache size helps the algorithms exploit the locality of reference in calls made. The effect is more significant in the static algorithm

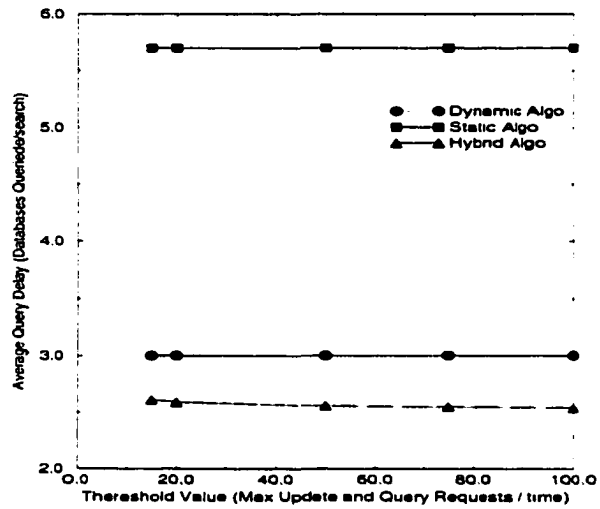

Figure 3.2 Effect of Choice of Threshold on Average Query Delay

than in the hybrid algorithm as the caches in the static algorithm are always consistent. The effect of increasing cache size on the two schemes is described in Figure 3.3. As the size of the cache increases the average query delay for the static algorithm decreases and tends to a delay of one as the cache size approaches the size of the mobile population.

3.5.2 Average Load on Databases

The dynamic algorithm of [51] is load balanced on the average. However the average load on the databases is high. This is due to the number of messages sent during each update and query. During each update the databases are first queried to locate the information, the information is then deleted and then a fresh set of updates is generated. On average, $O(\sqrt{n})$ databases are queried by a MSC while servicing a call delivery request. We now discuss the effect of cache size and the threshold level on the average load on the databases for the proposed static and hybrid scheme. We compare it with the the average load per database for the IS algorithm.

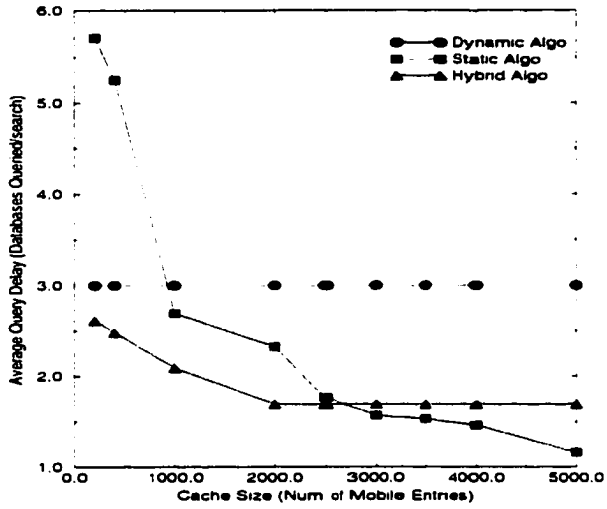

Figure 3.3 Effect of Cache Size on the Average Query Delay

Effect of Threshold Level: Figure 3.4 compares the average load on databases for the three algorithms. The cache size used in this simulation was 200. The average load for the IS algorithm is based on the fact that each database receives updates and query requests for only mobiles updated in them. For the static algorithm however, only those mobiles whose entries are found in the cache have a query delay of one. The other mobiles have to be searched. The observed average load therefore can be seen as the direct implication of the extra queries made to locate mobiles in the static algorithm. From Figure 3.3 the average query delay is 5.70. The average load therefore on each database is about five times the load on the databases for the IS algorithm as now the algorithm performs five times more number of queries.

The hybrid algorithm performs better as the threshold level increases. This is due to the fact that there are fewer threshold violations and hence fewer relocations of mobiles. This therefore reduces the extra load on the databases.

Effect of Cache Size: The size of the cache is an important consideration which decides the performance of the static algorithm and the hybrid algorithm. The perfor-

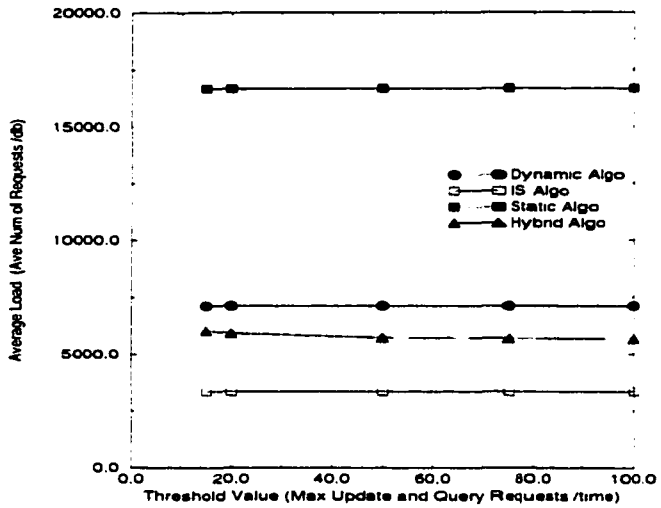

Figure 3.4 Effect of Threshold Level on Average Load on Database

mance is shown in Figure 3.5. As described previously, as the cache size increases the average query delay decreases for both the static and hybrid algorithms. This is because the number of called mobiles whose entries will be found in the cache increases. As the caches are always consistent, obtaining the location information of such mobiles requires only one query. As the cache size increases the average load on the databases approaches the average load for the IS algorithm.

The increase in cache size reduces the load on the databases as the number of queries needed to locate the databases reduces as more and more mobiles are located in the cache.

From the discussion above, we see that the hybrid algorithm proposed in the paper performs significantly better than [51] in terms of both average load on the database as well as the average query delay. For lower cache values, the hybrid algorithm performs better than the static algorithm. The advantage of the hybrid algorithm is that it needs no prior knowledge of the loads offered by each mobile.

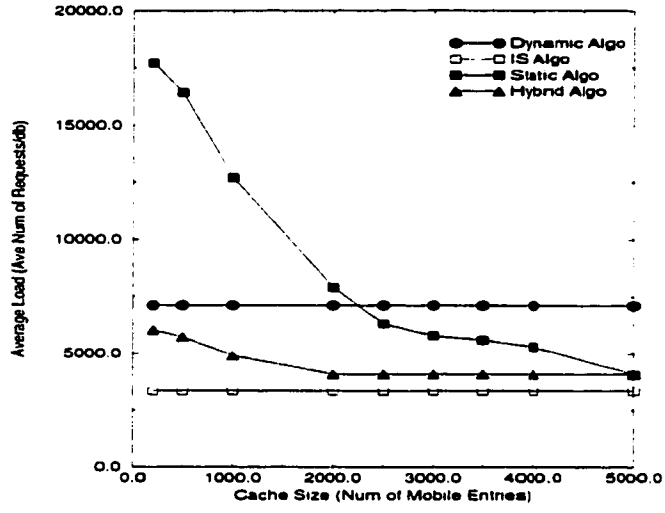

Figure 3.5 Effect of Cache Size on Average Load on Database

3.6 Summary

Efficient management of location information is an important aspect to consider in the design of future mobile networks, especially since the number of mobiles in the network is poised to increase at a high rate in the near future. We have established fundamental bounds to the performance of load-balanced distributed location management algorithms in terms of three important performance metrics. We then proposed an algorithm which simultaneously meets these bounds while satisfying the load balancing requirements. The strong optimality of this algorithm as well as its feasibility for implementation make it a candidate for location management in future networks. Another appealing feature of the proposed solution is that it balances the storage and query load at all times in a real-time environment. An alternative approach would be to balance in a time-average sense, which could result in temporary bottleneck in database access. We have also presented static and hybrid location management algorithms where in the mapping between the mobiles and the LIDs where their information is updated in does not change or changes less frequently. We have compared the performance of the hybrid

algorithm with the static algorithm and the dynamic algorithm.

The determination of the replication factor needs to take into consideration the relative costs of storage and query delay, fault tolerance, and possibly other factors. The cost minimization approach can be extended to include the other design constraints, such as the frequency of updates and queries a database is allowed to participate in. We address this issue in the next chapter.

Table 3.4 Pseudo-code for Query Strategy of Hybrid Algorithm

```
Query(m_i : mobile id);
    start : integer;
    count : integer;
    n : integer; /* Number of Databases */
    k : integer; /* Replication Factor */
begin
  if(m_i in cache)
   begin
    count := uniform_random[0,k]; /*index of the dbase to query */
     Query(D_{start⊕(count*n/k)});
     if(m_i in D_{start⊕(count*n/k)}) /*cache hit */
      return info;
     else /*cache inconsistency */
      begin
       start := uniform_random[1,n]; /* the id of first dbase for query */
       count = 0;
       while count < ⌈n/k⌉
          if (info in D_{π1⊕count})
            return info;
          else
            count := count + 1;
       done;
      end
   end
  else /* entry not in cache */
   begin
    start := uniform_random[1,n]; /* the id of first dbase for query */
    count = 0;
     while count < ⌈n/k⌉
     if (info in D_{π1⊕count})
      return info;
     else
       count := count + 1;
      done;
    end
end
```

4 OPTIMAL REPLICATION OF LOCATION INFORMATION

4.1 Introduction

To have the same replication factor for all mobiles in the system is not optimal. This is because of the difference in call arrival and mobility patterns of the mobiles in the network. Each mobile therefore, has to be replicated according to its activity state. A static partitioning of the mobiles according to their activity states is also not feasible unless the activity statistics of the mobile is known in advance. Therefore any replication protocol must allow for changes in the activity state of mobiles. Having different replication factors for each mobile is also not optimal as frequent messages may have to be generated indicating a change in activity state.

4.2 Dynamic Load Balanced Location Management Algorithm

In this section we present a dynamic update and query strategy (referred to as the Dynamic Algorithm) which is optimal with respect to three performance metrics namely, worst-case query delay, average query delay and call blocking probability. The optimality of these strategies is proved in the previous chapter. In the scheme, given the number of LIDs (n) in the network and the replication factor (k) for a mobile, update and query strategies which balance the load as well as optimizes sequential query delay are described. The work does not address the issue of *optimal replication factor* which we address in the rest of this chapter. We also analyze a parallel querying algorithm in this

chapter.

As described previously, to have the same replication factor for the entire population of mobiles is not optimal. This is because of the difference in the activity rates of mobiles. Some mobiles are more active than others (less active mobiles are referred to as relaxed mobiles). We denote the replication factor of highly-active mobiles by s and that of relaxed mobiles by t. The replication factor should chosen to balanced update and query costs.

Update Strategy: We now recapitulate the optimal update strategy described in Chapter 3. In this description we use k to refer to the replication factor. k is either s or t depending on whether a mobile is highly-active or relaxed. For a given (n, k) let us write

$$n = k\alpha + \beta$$

where $\alpha = \lfloor \frac{n}{k} \rfloor$ and $\beta = n - k\alpha$. We view the databases $D_1, D_2, \ldots, D_n$ to be arranged on a logical ring[1]. Let the first database for update be selected randomly, using a uniform distribution over the n databases. The remaining $k - 1$ databases to be updated are selected deterministically, at distances α and $\alpha + 1$ from the previous updated database on the logical ring. More specifically, let $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_k)$ be the placement vector, where $\gamma_i \in \{1, 2, \ldots, n\}$ is the index of the ith database updated by the algorithm. We define the displacement vector $\mathbf{a} = (a_1, a_2, \ldots, a_k)$ as a binary vector with Hamming weight β. Having selected the first database index γ_1 uniformly, the update strategy selects the remaining $k - 1$ databases as

$$\gamma_{i+1} = \gamma_i \oplus (\alpha + a_i), \qquad i = 1, \ldots k-1 \tag{4.1}$$

where $\oplus$ denotes modulo n addition defined over the set $\{1, 2, \ldots, n\}$.

The displacement vector $\mathbf{a}$ can be fixed a priori, or it can be randomly generated at each update. As long as its Hamming weight is β, the resulting update algorithm

[1]This topology does not impose any constraints on the physical topology. The databases can be connected via an arbitrary physical topology, such as the PSTN.

possesses the optimality properties which will be obtained below. Note that when k is a divisor of n, the algorithm places the location information at equidistant databases on the logical ring with a random "phase".

The MSC, when performing the update for a mobile, also deletes the previously updated location information of the mobile. The MSC can identify the databases with the previous location information by getting the information from the mobile or by querying the databases. The MSC thereby receives information about the state of the mobile (highly-active or relaxed). This information is used in deciding the replication factor of the new update. The state of the mobile is maintained in the new update. The MSC generates a fresh random number to decide the first database for update upon receiving a location update request.

Query Strategy: The average sequential query delay of the query scheme in Chapter 3 is of $O(\sqrt{n})$. To avoid this delay we parallelize the queries. The load balancing aspect of the query strategy is not affected by the parallel queries [51]. We now describe the modified query strategy.

On receiving a request to locate a mobile's location information, the MSC of the calling mobile assumes that the called mobile is highly-active and queries $\lceil \frac{n}{s} \rceil$ databases, $D_{\pi_1}, D_{\pi_{1\oplus 1}}, \ldots, D_{\pi_{1\oplus\lceil \frac{n}{s} \rceil}}$ in parallel, where $\oplus$ is modulo addition over the set $\{1, 2, \ldots, n\}$. D_{π_1} is chosen uniformly over the n databases in the network. If the called mobile is highly-active, the query strategy terminates with this round of queries as any two successive LIDs with updates for a highly-active mobile is separated by a maximum of $\lceil \frac{n}{s} \rceil$ LIDs. However, if the called mobile were relaxed, then there is a chance that the information is not retrieved in the first round of queries. We therefore use a second round of $\lceil \frac{n}{t} \rceil - \lceil \frac{n}{s} \rceil$ parallel queries to locate relaxed mobiles whose location information is not retrieved in the first round. Specifically, in the second round $D_{\pi_{1\oplus\lceil \frac{n}{s} \rceil\oplus 1}}, \ldots, D_{\pi_{1\oplus\lceil \frac{n}{t} \rceil}}$ are queried in parallel. As the worst-case query delay for relaxed mobiles is $\lceil \frac{n}{t} \rceil$, we are guaranteed to locate all relaxed mobiles. The pseudo-code of the parallel query strategy is given in Table 4.1.

Table 4.1 Pseudo-code for Parallel Query Strategy

```
Query(m_i : mobile id);
    π_1 : integer;
    n : integer; /* Number of Databases */
    s : integer; /* Replication Factor of Highly-Active mobiles*/
    t : integer; /* Replication Factor of relaxed mobiles */
begin
 π := uniform_random[1,n]; /* 1st db queried */
 Query D_{π_1} through D_{pi_1 ⊕ ⌈n/s⌉} in parallel;
 if info found then
  return;
 else
  Query D_{π_1 ⊕ ⌈n/s⌉ ⊕ 1} through D_{π_1 ⊕ ⌈n/t⌉}
end
```

State Transition for Mobiles: In the previous section, we described update and query strategies which depended on partitioning the population of mobiles based on their activity. This partitioning can be static. However, this solution is not effective, as the behavior of a mobile can change radically over time. We therefore use the startegy described below to take care of the transition of activity states for the mobiles.

The MSC of a mobile's current LA changes the state of the mobile depending on the rate of activity of the mobile. To achieve this, the MSC maintains statistics about the activity of the mobiles in its LA. Once a mobile moves to a different LA, the mobile's statistics are transferred to the new MSC. Using the statistics two strategies can be adopted to decide when a mobile m_i's activity state changes from highly-active to relaxed and vice-versa. We can designate m_i as highly-active if the rate of calls received by the mobile is greater than some threshold T_1 The advantage of this scheme is that hot mobiles now have higher location update replication thus reducing the number messages to find the mobile.

Another scheme would be to designate m_i as highly-active when its Call to Mobility Ratio (CMR) is greater than a threshold T_2. The advantage of this strategy is to reduce the number of transition updates to the HLR databases thus reducing its load. In

the next section, we present a methodology to partition the population of mobiles into highly-active and relaxed mobiles.

The state transition procedure (Table 4.2) for a mobile is similar to a regular location update except that a query is not done. This is because mapping between mobiles in the MSC's LA and the databases the mobiles are updated in and the state of the mobile are known to the MSC. When the state of a mobile changes, the MSC generates a pseudo-LU. This pseudo-LU serves the purpose of increasing/decreasing the number of LIDs the mobile is updated in. The MSC deletes the mobile's present location information from the databases. The MSC then updates the mobile's information in the appropriate number of databases. The state variable which denotes the mobile's state is updated to register the change in state.

The Issue of Load Balance: The proof of load balance of the Query and Update strategies is given in the previous chapter. The transition procedure essentially is a new update with different replication factor and hence can be shown using a similar argument to be load balanced.

4.3 Analysis

It is shown in the previous chapter that when queries are made sequentially to locate a mobile's information, the minimum average number of queries needed to locate a mobile's information for a given (n, k) is $\frac{1}{2}(\alpha + 1)(1 + \frac{\beta}{n})$, where $\alpha = \lfloor \frac{n}{k} \rfloor$ and $\beta = n - k\alpha$. We now analyze the location management algorithm when the queries are made in at most two rounds of parallel queries.

Let $W^r_{LU}(W^h_{LU})$ be the average work (average number of updates/queries) during the LU of a relaxed (highly-active) mobile. $W^r_A(W^h_A)$ be the average work (average number of queries received) to locate a relaxed (highly-active) mobile. Similarly, let W_T be the average work during a state transition. We have the following equations to determine the work done for the different procedures in the protocol.

During a highly-active (relaxed) to relaxed (highly-active) transition s (t) deletes are performed and t (s) updates are performed on the LIDs. During an LU process of a highly-active mobile the mobile's information is first located, and then s LIDs are deleted and s LIDs are updated with the new information. While locating a highly-active mobile $\lceil \frac{n}{s} \rceil$ queries are made. We therefore have the following equations.

$$W_T = s + t \tag{4.2}$$

$$W_A^h = \left\lceil \frac{n}{s} \right\rceil \tag{4.3}$$

$$W_{LU}^h = 2s + W_A^h \tag{4.4}$$

While locating a relaxed mobile the Query strategy might need two rounds of parallel queries. In the first round $\lceil \frac{n}{s} \rceil$ queries are made and with a certain probability P the second round is executed to locate a relaxed mobile. We therefore have the following equations

$$W_A^r = \left\lceil \frac{n}{s} \right\rceil + \left(\left\lceil \frac{n}{t} \right\rceil - \left\lceil \frac{n}{s} \right\rceil \right) \cdot P \tag{4.5}$$

$$W_{LU}^r = 2t + W_A^r \tag{4.6}$$

We now evaluate the probability P that the Query strategy uses a second round of queries to locate the information of a relaxed mobile (m_i). Let the random starting database chosen in the Query strategy be D_{π_1}. If $\{D_{i_1}, \ldots, D_{i_t}\}$ represents the t LIDs the relaxed mobile is updated in, the Query strategy terminates in one round of parallel queries if $\{D_{\pi_1}, D_{\pi_1 \oplus 1}, \ldots, D_{\pi_1 \oplus \lceil \frac{n}{s} \rceil}\} \cap \{D_{i_1}, \ldots, D_{i_t}\}$ is not empty. If $\lceil \frac{n}{s} \rceil \leq \lfloor \frac{n}{t} \rfloor$, the probability that the Query strategy terminates in one round is given by $\frac{t}{n} \lceil \frac{n}{s} \rceil$. Therefore, $P = 1 - \frac{t}{n} \lceil \frac{n}{s} \rceil$. If $\lceil \frac{n}{s} \rceil \leq \lfloor \frac{n}{t} \rfloor$, we have

$$1 - \frac{t}{s} - \frac{t}{n} \leq P \leq 1 - \frac{t}{s} \tag{4.7}$$

Given a partition of the population of the mobiles in the network in relaxed and highly-active mobiles, we define the following quantities:

- $\lambda_A^r(\lambda_A^h)$: The rate of incoming calls for relaxed (highly-active) mobiles.

- $\lambda^r_{LU}(\lambda^h_{LU})$: The rate of LUs for relaxed (highly-active) mobiles.
- λ_T: The rate of transition updates.
- The function *Cost* denotes the average cost of the algorithm in terms of the queries/updates made.

Using (4.2) - (4.6), *Cost* is given by

$$\begin{aligned} Cost(s,t) &= \lambda^h_A \cdot W^h_A + \lambda^r_A \cdot W^r_A \\ &+ \lambda^h_{LU} \cdot W^h_{LU} + \lambda^r_{LU} \cdot W^r_{LU} + \lambda_T \cdot W_T \\ &= (\lambda^h_A + \lambda^h_{LU}) \cdot \left\lceil \frac{n}{s} \right\rceil \\ &+ (\lambda^r_A + \lambda^r_{LU}) \left(\left\lceil \frac{n}{s} \right\rceil + \left(\left\lceil \frac{n}{t} \right\rceil - \left\lceil \frac{n}{s} \right\rceil \right) \cdot P \right) \\ &\quad + 2\lambda^h_{LU} \cdot s + 2\lambda^r_{LU} \cdot t + \lambda_T \cdot (s+t) \\ &= A \cdot \left\lceil \frac{n}{s} \right\rceil + B \left(\left\lceil \frac{n}{s} \right\rceil + \left(\left\lceil \frac{n}{t} \right\rceil - \left\lceil \frac{n}{s} \right\rceil \right) \cdot P \right) \\ &+ C \cdot s + D \cdot t \end{aligned} \tag{4.8}$$

where $A = \lambda^h_A + \lambda^h_{LU}, B = \lambda^r_A + \lambda^r_{LU}, C = 2\lambda^h_{LU} + \lambda_T, D = 2\lambda^r_{LU} + \lambda_T$. s and t are the replication factors for highly-active and relaxed mobiles respectively. Let us denote by s_m, t_m the integer values of s and t for which *Cost* is minimized ($1 \leq t_m \leq s_m \leq n$). The function *Cost* is mathematically intractable. We therefore find simpler functions which serve as upper and lower bounds to the function *Cost*. We will show that the upper and lower bounds of these bounding functions are of $O(\sqrt{n})$ and thus conclude that minimum of *Cost* is of $O(\sqrt{n})$. Consider the function

$$Cost'(s,t) = A\frac{n}{s} + B\left(\frac{n}{t} - \frac{n}{s} + \frac{n \cdot t}{s^2}\right) + C \cdot s + D \cdot t. \tag{4.9}$$

Using (4.7), it can be shown that

$$Cost' - 2B \leq Cost \leq Cost' + A + 2B \tag{4.10}$$

Finding the minimum of *Cost'* involves the solving a third degree equation with many parameters. The solutions are thus not very easily manageable. We therefore determine

two functions f' and f'' which are upper and lower bounds of $Cost'$ and whose minima give a reasonable approximation of the minimum of $Cost$. Once the parameters of the system are known, it is not difficult to find s_m, t_m such that the cost is minimized. The functions f' and f'' are used to determine the order of the minimum cost.

Upper Bound to the Minimum of Function Cost: Choose $f'(s,t) = A\frac{n}{s} + C \cdot s + B\frac{n}{t} + D \cdot t$. f' can be written as functions of s and t as follows $f'_1(s) + f'_2(t)$. We use such a construction for f' to simplify the minimization process. The minimum of f' now is the sum of the minimum of f'_1 and f'_2. Let the s' and t' be the values of s and t which minimizes f'. It is possible to show that $s' = \sqrt{n\frac{A}{C}}$, $t' = \sqrt{n\frac{B}{D}}$ and the minimum value of f' is $f'(s',t') = 2\sqrt{n \cdot A \cdot C} + 2\sqrt{n \cdot B \cdot D}$. Using the fact that $-\frac{n}{s} + \frac{n \cdot t}{s^2} \leq 0$, for all t, s such that, $1 \leq t \leq s \leq n$, we have $f' \geq Cost'$, for all t, s, such that $1 \leq t \leq s \leq n$. Observe that $s' > t'$ if $\sqrt{\frac{A \cdot D}{B \cdot C}} > 1$. Now the values of s and t represent the replication factors of highly-active and relaxed mobiles respectively, and hence are integers. We choose $s'_m = \lceil s' \rceil$ and $t'_m = \lceil t' \rceil$ as the integer values of s' and t'. The values of s'_m and t'_m need not minimize f' over integers as we are only interested in an upper bound. Now

$$f'(s'_m, t'_m) - f'(s', t') \leq C + D. \tag{4.11}$$

Using (4.10), (4.11) and the fact that $Cost'(s,t) \leq f'(s,t)$ for $1 \leq t \leq s \leq n$, we have an upper bound to the minimum of the function $Cost$ which is

$$Cost(s_m, t_m) \leq 2\sqrt{n \cdot A \cdot C} + 2\sqrt{n \cdot B \cdot D} + A + 2B + C + D. \tag{4.12}$$

Lower Bound to the Minimum of Function Cost: Consider the function $f''(s,t) = A\frac{n}{s} + B\frac{n}{3t} + C \cdot s + D \cdot t$. Now if $s \geq \frac{3}{2}t$, then

$$\begin{aligned} Cost'(s,t) &\geq A\frac{n}{s} + B\left(\frac{n}{t} - \frac{2n}{3t}\right) + C \cdot s + D \cdot t \\ &\geq A\frac{n}{s} + B\frac{n}{3t} + C \cdot s + D \cdot t = f''(s,t). \end{aligned} \tag{4.13}$$

If $s < \frac{3}{2}t$, as $s \geq t \geq 1$, we have

$$Cost'(s,t) \geq A\frac{n}{s} + B\frac{n \cdot t}{s^2} + C \cdot s + D \cdot t$$

$$\geq \quad A\frac{n}{s} + B\frac{4n}{9t} + C \cdot s + D \cdot t \geq f''(s,t). \tag{4.14}$$

From (4.13) and (4.14), it follows that $f'' \leq Cost'$.

Let the minimum of f'' be attained at $s = s''$ and $t = t''$ respectively. Using an argument similar to the one used in minimizing f' we can show that the minimum value of f'' is $2\sqrt{n \cdot A \cdot C} + \sqrt{\frac{4}{3}n \cdot B \cdot D}$, is obtained for $s'' = \sqrt{\frac{n \cdot A}{C}}, t'' = \sqrt{\frac{n \cdot B}{3D}}$. $s'' > t''$ if $\sqrt{\frac{3A \cdot D}{B \cdot C}} > 1$. An interesting observation here is that if $s' > t'$ then $s'' > t''$. The minimum f'' over the set of Reals is at most the minimum of f'' over the set of Integers. Therefore, by (4.10) and the fact that $Cost'(s,t) \geq f'(s,t)$ for $1 \leq t \leq s \leq n$, we have

$$Cost(s_m, t_m) \geq 2\sqrt{n \cdot A \cdot C} + \sqrt{\frac{4}{3}n \cdot B \cdot D} - 2B. \tag{4.15}$$

Using (4.12) and (4.15) and the condition that $s' > t'$, we see that the minimum of $Cost$ is $O(\sqrt{n})$. The proof depends on the condition that $s' > t'$. The condition implies that

$$\sqrt{\frac{A \cdot D}{B \cdot C}} \geq 1 \iff \sqrt{\frac{(\lambda_A^h + \lambda_{LU}^h)(2\lambda_{LU}^r + \lambda_T)}{(\lambda_A^r + \lambda_{LU}^r)(2\lambda_{LU}^h + \lambda_T)}} \geq 1. \tag{4.16}$$

Therefore for the condition $s' > t'$ to be true the ratio of the CMR of highly-active mobiles to the CMR of relaxed mobiles (assuming that $\lambda_T << \lambda_{LU}^h, \lambda_{LU}^r$). Therefore the condition $s' > t'$ is true if the CMR of the highly-active mobiles is greater than the CMR of the relaxed mobiles. (4.16) also gives us an idea about how to partition the system into highly-active and relaxed mobiles.

From our proof that both the lower and upper bounds are of $O(\sqrt{n})$ we have shown that the minimum of the function $Cost$ is of $O(\sqrt{n})$.

4.4 Discussion

In this section we present a discussion of our analysis and the interpretation of our results. We then compare the performance of the Dynamic algorithm described with the load balanced location algorithm presented in [29] (referred to as the P&S algorithm). In the following discussion we use the following sets of values of $\{\lambda_A^h, \lambda_{LU}^h, \lambda_A^r, \lambda_{LU}^r, \lambda_T\}$:

{10,1,2,1,0.5} and {5,0.8,2,1.1,0.3} (Both the sets satisfy (4.16)). Figure 4.1 depicts (4.8) for $n = 50$ and the two sets of parameters. Table 4.3 we compare the cost of the Dynamic algorithm and P&S algorithm. The table shows an upper bound of the cost of the Dynamic algorithm for the LU and Query for cold mobiles as we assume that the algorithm always uses a second round of queries to locate a cold mobile. The cost of each operation in the P&S algorithm depends only on the number of databases n, while the cost of the Dynamic algorithm depends on the values of s and t. We select arbitrary

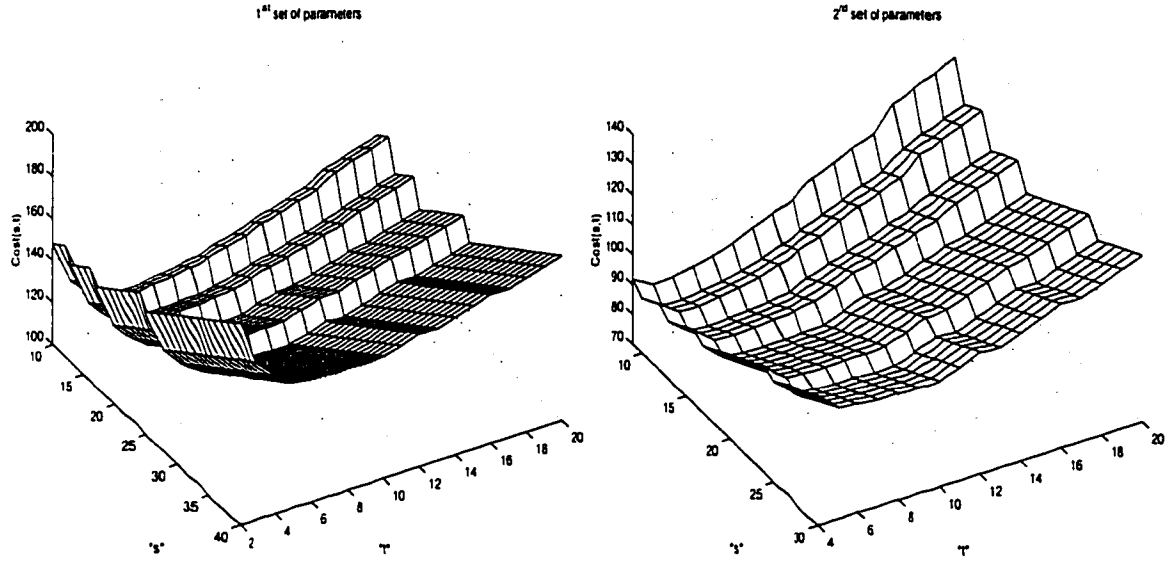

Figure 4.1 Dynamic Algorithm Cost with $n = 50$ using the two sets of parameters

values for s and t in the comparison of the two schemes. We choose $s = \sqrt{n}, t = 0.7\sqrt{n}$ for the comparison. The minimum cost for the Dynamic Algorithm therefore is at most this cost. From the table it is clear that the P&S algorithm has a higher cost than the Dynamic algorithm. This higher cost of the P&S algorithm can also be interpreted as a higher average load on the databases. This is because more databases are queried and updated for each LU and call delivery request respectively, in the P&S algorithm.

Once the parameters of the system are known it is very easy to find integers s_m and t_m which minimize (4.16). In Table 4.4 we present the values of s_m and t_m which

minimize (4.16) and we compare the minimum costs of the Dynamic algorithm and the P&S algorithm. Figure 4.2 is a graphical representation of Table 4.4. It is clear from the figure that the Dynamic scheme performs significantly better than the P&S scheme[2]. The performance improvement becomes more significant as the number of LIDs increase. This higher cost of the P&S algorithm can also be interpreted as a higher average load on the databases. This is because more databases are queried and updated for each LU and call delivery request respectively, in the P&S algorithm.

Adapting to Changes in the Replication Factor: The replication factor s and t could change because the system is scaled or the rates change. In the case the replication factors change, the reconfiguration of the system is handled in the following way. For a predetermined period of transition the MSC's generate pseudo-location updates for the mobiles in their LA using the new value of s (if the mobile is hot) or t (if the mobile is cold). To locate a mobile a MSC uses the minimum of the two values of s as the temporary working value of s and the minimum of the two values of t as the temporary working value of t. A regular LU of a mobile is served with the new value of s or t. After the transition phase, the system uses the new values of s and t. Such reconfigurations can be done during periods when the traffic is low.

The dynamic replication algorithm described in this paper has the following advantages. Unlike [29] the algorithm does not use a centralized database to store the status of mobiles. It also does not use a centralized server to collect statistics about mobiles in the network like the algorithm described in [27]. The algorithm uses the MSCs in the system to collect statistics about mobiles in the system. This is practical as all calls in an LA are routed by the MSC. The dynamic algorithm described chooses the optimal replication factor for both highly-active and relaxed mobiles by involving various network parameters like call arrival rates, mobility rates and state transition rates. The algorithm is fully adaptive to the changes in network parameters. Further the algorithm is very simple to implement thus making it a feasible choice for use in real-life mobile

[2]In the calculation of the costs, a query is assumed to be initiated to locate databases for deletion at the time of a LU.

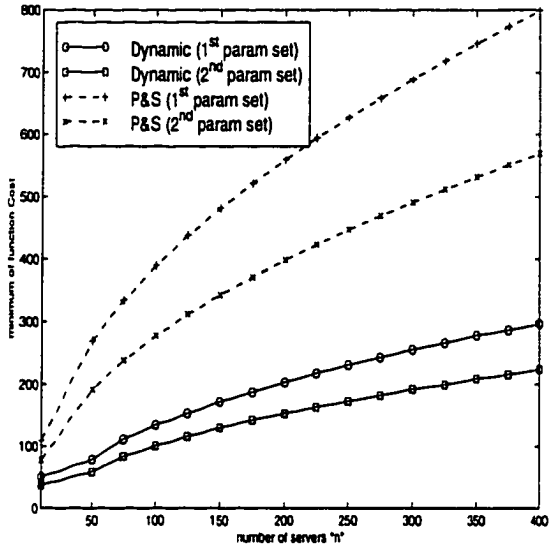

Figure 4.2 Comparison between the Dynamic Algorithm and P&S Algorithm

networks.

4.5 Summary

In this chapter, we presented the issues involved in optimally replicating mobiles based on their activity. We showed using analysis that the average cost of the algorithm is of $O(\sqrt{n})$ where n is the number of databases in the system. We presented a methodology to partition the system into highly-active and relaxed mobiles. [29] uses a centralized database to store the status of the mobiles and also uses a static approach in deciding the degree of replication for hot and cold mobiles. The dynamic algorithm described in this chapter avoids this centralized database and uses the MSCs to make the decision. The approach presented is adaptive to changes in system parameters in deciding the degree of replication, thus minimizing the cost.

Networks are prone to failures, and it is important that any location management algorithm performs well in the presence of failures. In the remaining portion of this work

we address this issue. In the next chapter, we analyze the effect of LID/failures on the performance of load balanced location management algorithms.

Table 4.2 Transition from cold to hot and from hot to cold

```
Transition(m_i : Mobile ID; state : old state of mobile);
    γ1 : integer;
    count : integer;
    n : integer; /* Number of Databases */
    k : integer; /* Replication Factor */
    β : integer;
begin
DELETE mobile's old information;
state_i = state; /*update m_i's state */
if state is highly-active then
  k = s; /* New replication factor */
else
  k = t; /* New replication factor */
  β = n − k⌊n/k⌋;
   γ1 := prev_starting_database; /*from mobile or search */
   DELETE old information from databases;
   γ1 := uniform_random[1, n]; /*new starting database */
    if β = 0 then
      count := 0;
      while count < k
        UPDATE D_{γ1 ⊕ (count * n/k)}
        count := count + 1;
      done;
    else
      count := 0;
      while count ≤ k − β
        UPDATE D_{γ1 ⊕ (count * ⌊n/k⌋)};
        count := count + 1;
      done;
      count := 1;
      while count ≤ β − 1
        UPDATE D_{(γ1 ⊕ (k−β)⌊n/k⌋ ⊕ (count * ⌈n/k⌉))};
        count := count + 1;
      done;
end
```

Table 4.3 Comparison of Individual Procedure Costs

	Ave Dynamic Cost	$(s = \sqrt{n}, t = 0.7\sqrt{n})$	P&S cost
LU highly-active	$2s + \lceil \frac{n}{s} \rceil$	$\leq 3\sqrt{n} + 1$	$10\sqrt{n} - 5$
LU relaxed	$\leq 2t + \lceil \frac{n}{t} \rceil$	$\leq 2.42\sqrt{n} + 1$	$6\sqrt{n} - 3$
Query highly-active	$\lceil \frac{n}{s} \rceil$	$\sqrt{n} + 1$	$2\sqrt{n} - 1$
Query relaxed	$\leq \lceil \frac{n}{t} \rceil$	$\leq 1.42\sqrt{n} + 1$	$2\sqrt{n} - 1$
State transition	$s + t$	$1.7\sqrt{n}$	$2\sqrt{n} - 1$

Table 4.4 Comparison of Overall Costs

	Dynamic 1^{st} param set			Dynamic 2^{nd} param set			P&S Cost	
n	$AveCost$	s_m	t_m	$Cost$	s_m	t_m	(1st param)	(2nd param)
10	50.4	5	3	37.3	5	3	85.2	57.5
25	78.4	13	5	58.8	9	5	210.3	141.9
50	111.0	17	7	100.2	13	6	261.1	176.3
100	152.5	20	10	129.6	20	10	341.8	230.7
200	217.7	29	13	172.5	25	13	464.0	313.2
400	305.0	40	20	230.5	40	20	643.7	434.5

5 PERFORMANCE OF LOCATION MANAGEMENT ALGORITHMS IN THE PRESENCE OF FAILURES

5.1 Introduction

Any network is susceptible to failures. Hence, a location management algorithm must have in built mechanisms to perform in the presence of failures. In this chapter we derive bounds for three standard performance metrics satisfied by any load balanced location management algorithm. We then present robust update and query strategies. We analyze the performance of these robust strategies with respect to worst-case query delay, average query delay and call blocking probability. We then analyze the performance of the robust version of the parallel querying strategy described in Chapter 4.

5.2 Performance Bounds

In this section we analyze the performance of load balanced location management algorithms in the presence of faults using three standard performance metrics, namely, worst-case query delay, average query delay, and call blocking probability. The metric worst-case query delay refers to the maximum number of LIDs the query algorithm has to query to locate the called mobile's location information. Average query delay similarly, denotes the average number of LIDs queried. Real life networks may have an upper bound on the number of queries possible after which the call is dropped. This performance of algorithms is measured by the third metric. We now define the quantities that will be used in the analysis.

- Let n be the number of LIDs in the system. Let the databases be labeled as $D_1, D_2, \ldots, D_n$.
- Let the replication factor k be the number of databases a mobile's location information is updated in.
- Let X_i be a binary random variable indicating whether D_i has the location information of a given mobile. We define the storage vector as $\mathbf{X} = (X_1, X_2, \ldots, X_n)$.
- Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be the query order of the databases for accessing the location information of a given mobile. π is a permutation of $\{1, 2, \cdots, n\}$, and D_{π_j} is the jth database to be queried if the location information has not been found in $D_{\pi_1}, D_{\pi_2}, \ldots, D_{\pi_{j-1}}$ (while this implies a sequential search, the databases could be searched in parallel).
- Let Y_i be the binary random variable indicating whether the ith queried database has the required information, i.e. $Y_i = X_{\pi_i}$.
- Let there be t LID failures in the system, $0 \leq t < k$. The probability that any database is failed is given by $\frac{t}{n}$. The failures can be either LID failures or link failures which manifest themselves as LID failures, as the LID which is connected by the link is unreachable. We assume that the queried LID being failed or alive, does not have any effect on whether the LID has the required information, i.e., we assume probabilistic independence of "failure" and the "updated LID" events.
- Let T denote the number of queries to find the location information. We refer to T as the *query delay*. It is implicitly assumed that each database query results in a constant time delay, which is normalized to unity.

From the above definitions, the storage vector is a random binary n-vector with Hamming weight k, i.e. $\sum_{i=1}^{n} X_i = k$. Since $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_n)$ is a permutation of the storage vector $\mathbf{X}$, $\mathbf{Y}$ is also a binary n-vector with Hamming weight k. The query delay

can be written as

$$T = \min\{i : Y_i = 1\}.$$

A load balanced update mechanism must satisfy $E[X_i] = E[X_j]$, for all i, j. In conjunction with the Hamming weight constraint, this implies that

$$P(X_i = 1) = \frac{k}{n} \qquad i = 1, 2, \ldots, n. \tag{5.1}$$

Since $\mathbf{Y}$ is a permutation of $\mathbf{X}$, in the fault free case we have

$$P(Y_i = 1) \;=\; \frac{k}{n} \qquad i = 1, 2, \ldots, n \tag{5.2}$$

Note that a location update algorithm may be viewed as a probability distribution on the storage vector $\mathbf{X}$, while a query vector is a probability distribution on π. Thus a load balanced update must satisfy the condition given in (3.1) on the marginal (Bernoulli) distribution. The theorem below provides a lower bound to the maximum value of the query delay T.

5.2.1 Lower Bound to Worst-Case Query Delay

In this section, we obtain a lower bound to the worst-case query delay for any load balanced location algorithm in the presence of t faults.

Theorem 1: *For any load balanced algorithm and any query algorithm, if j is an integer such that $P(T > j) = 0$ and there are t LID failures in the network, then $j \geq \lceil \frac{n^2}{k(n-t)} \rceil$.*

Proof: The event that a LID is failed and the event having the required information are independent. The probability of a LID being failed is $\frac{t}{n}$. Let j be an integer such that $P(T > j) = 0$. Then

$$P(\sum_{i=1}^{j} Y_i \geq 1) = 1\,. \tag{5.3}$$

The event in (5.3), is the union of the events $\{Y_i = 1\}$, $i = 1, 2, \ldots, j$. Then by union bound

$$1 \leq \sum_{i=1}^{j} P(Y_i = 1) = \frac{jk(n-t)}{n^2} \tag{5.4}$$

where we have also used (5.2) and the independence notion of the failed LID and LID with information events. The result follows from the last inequality, and observing that j is a positive integer. ■

5.2.2 Lower Bound to Average Query Delay

In this section, we obtain a lower bound to the expected value of the query delay for any load balanced algorithm, in the presence of t faults. We again make use of the independence between the failed LIDs and the LIDs with the information about the called mobile.

Theorem 2: *Given (n, k), the average query delay E[T], for a load balanced location management algorithm in the presence of t failures satisfies $E[T] \geq \frac{1}{2}(\zeta + 1)(1 + \frac{\nu}{n^2})$. where $\zeta = \lfloor \frac{n^2}{k(n-t)} \rfloor$ and $\nu = n^2 - k(n-t)\zeta$.*

Proof: Let $p_j = P(T = j)$, the probability that the query algorithm finds the information in the jth query, $j = 1, 2, \ldots, n$. We have

$$\begin{aligned} p_j &= P(\sum_{i=1}^{j-1} Y_i = 0, (Y_j = 1, \text{LID not failed})) \\ &\leq P(Y_i = 1, \text{LID not failed}) \\ &= \frac{(n-t)k}{n^2} \end{aligned}$$

for any load balanced algorithm, in the presence of t failures. We now construct the following linear program in the variables $(p_1, p_2, \ldots, p_n)$.

$$\Theta = \min \sum_{j=1}^{n} j p_j$$

subject to

$$0 \leq p_j \leq \frac{(n-t)k}{n^2} \qquad j = 1, 2, \ldots, n$$

$$\sum_{j=1}^{n} p_j = 1 \, .$$

The objective function is minimized by the probability distribution

$$\begin{aligned} p_j &= \frac{(n-t)k}{n^2} \qquad j = 1 \, , \, 2, \ldots, \zeta \\ p_j &= \frac{\nu}{n^2} \qquad j = \zeta + 1 \\ p_j &= 0 \qquad j > \zeta + 1 \end{aligned}$$

with the resulting value

$$\Theta = \frac{1}{2}(\zeta + 1)\left(1 + \frac{\nu}{n^2}\right) \, . \tag{5.5}$$

Since $E[T] \geq \Theta$ for any load-balancing algorithm, the result is established. ■

5.2.3 Lower Bound to Call Blocking Probability

In a large PCS network supporting real-time traffic, it may not be acceptable to have large delays in establishing connections. In such a situation, it may be necessary to block a connection request once the query delay has exceeded a certain threshold m. We now obtain a lower bound to the call blocking probability P_B with any load balanced location management algorithm, in the presence of t faults.

Theorem 3: *For any load-balanced location management algorithm, with a query delay threshold r, the blocking probability P_B satisfies*

$$P_B \geq \left(1 - \frac{r(n-t)k}{n^2}\right)^+$$

where t is the number of faults in the network and $a^+ = \max(0, a)$.

Proof: We use the upper bounds on the delay distribution that were found in the proof of Theorem 2 to obtain

$$\begin{aligned} P_B &= P(T > r) \\ &= 1 - \sum_{j=1}^{r} p_i \\ &\geq 1 - \frac{r(n-t)k}{n^2} . \end{aligned} \tag{5.6}$$

■

Note that the lower bound is nontrivial only if $r \leq \lfloor \frac{n^2}{k(n-t)} \rfloor$.

5.3 Fault Tolerant Load Balanced Location Management

Location management algorithms in mobile networks are used to update the location information of mobiles in LIDs and the LIDs are queried during call delivery. Failed LIDs during call delivery could lead to calls being blocked. Intermediate link failures can also manifest themselves as LID failures. The location management process therefore, has to be robust to tolerate these failures. In [51], a dynamic location management algorithm is presented. The algorithm is load balanced and is shown to be optimal with respect to worst-case query delay, average query delay and call blocking probability in the absence of database failures. We first give a brief description of this algorithm. We adapt the algorithm to accommodate failures in the network. We then analyze the performance of this robust algorithm in the presence of database failures and compare the results with the bounds derived in Section 5.2.

5.3.1 Distributed Optimal Load Balanced Location Management

In this section we give a brief description of the load balanced algorithm presented in Chapter 3. The algorithm uses the following update and query strategies.

Update Strategy: For a given (n, k) let

$$n = k\alpha + \beta$$

where $\alpha = \lfloor \frac{n}{k} \rfloor$ and $\beta = n - k\alpha$. We view the databases $D_1, D_2, \ldots, D_n$ to be arranged on a logical ring[1]. Let the first database for update be selected randomly, using a uniform distribution over the n databases. The remaining $k - 1$ databases to be updated are selected deterministically, at "distances" α and $\alpha + 1$ (D_{i+1} is at a "distance" of 1 from D_i on the logical ring) from the previous updated database on the logical ring. Let $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_k)$ be the placement vector, where $\gamma_i \in \{1, 2, \ldots, n\}$ is the index of the ith database updated by the algorithm. We define the displacement vector $\mathbf{a} = (a_1, a_2, \ldots, a_k)$ as a binary vector with Hamming weight β. Having selected the first database index γ_1 uniformly, the update strategy selects the remaining $k - 1$ databases as

$$\gamma_{i+1} = \gamma_i \oplus (\alpha + a_i), \qquad i = 1, \ldots k-1 \tag{5.7}$$

where $\oplus$ denotes modulo n addition defined over the set $\{1, 2, \ldots, n\}$.

The displacement vector $\mathbf{a}$ can be fixed a priori, or it can be randomly generated at each update. As long as its Hamming weight is β, the resulting update algorithm possesses the optimality properties which will be obtained below. Note that when k is a divisor of n, the algorithm places the location information at equidistant databases on the logical ring with a random "phase".

The MSC, when performing the update for a mobile, also deletes the previously updated location information of the mobile. The MSC can identify the databases with the previous location information by getting the information from the mobile or by

[1]This topology does not impose any constraints on the physical topology. The databases can be connected via an arbitrary physical topology, such as the telephone network.

querying the databases. Note that the same set of databases cannot be used repeatedly for updates, as some of the mobiles may be queried more frequently or may require more updates than others. In this case the databases which store information about these active mobiles may have a higher load than databases which store information of less active mobiles. The MSC therefore generates a fresh random number to decide the first database for update upon receiving a location update request.

Query Strategy: Our query strategy starts at a randomly selected database and searches the databases contiguously until the location information is found. The algorithm chooses the initial database index π_1 using a uniform distribution on $\{1, 2, \ldots, n\}$ and employs a sequential search with order $D_{\pi_1}, D_{\pi_{1\oplus 1}}, \ldots, D_{\pi_{1\oplus(n-1)}}$. The pseudo-code of the Query Strategy is given in Table 3.2. To accommodate the presence of database faults, we can modify the the query strategy to use more queries than the upper limit of $\lceil \frac{n}{k} \rceil$ specified by the query strategy to locate a mobile's location information.

The algorithm is load balanced and the worst-case query delay, average query delay and the call blocking probability are $\lceil \frac{n}{k} \rceil$, $\frac{1}{2}(\alpha + 1)(1 + \frac{\beta}{n})$, and $1 - \frac{rk}{n}$ $(r < \lceil \frac{n}{k} \rceil)$, respectively. The proofs of these results and their optimality are given in [51].

5.4 Robust Location Management

In this section, we extend the query and update strategies of [51], to accommodate database failures.

Robust Update Strategy: The robust update strategy differs from the update strategy described in [51], in the way the updates are handled when a LID chosen for update is failed or unreachable. We adopt the following approach. The update mapping (the LIDs to be updated) is generated as described in [51]. If some of the LIDs thus chosen for update are failed, a fresh update mapping is generated. This approach assumes that a usable mapping will be found soon. This is true when the number of failures t is low when compared to k and n. This approach ensures that k LIDs are

successfully updated, i.e. k replicas of the location information of the mobile generating the location update, are available for the query algorithm, given the prevailing failed set of LIDs. However, if a usable update mapping cannot be generated, i.e. the number of failures in the system is very high, we use the approach described in [56]. Here, the MSC retries the update periodically, with the retry period decided based on the activity of the mobile. These retries are continued until the LID registers the update. A similar approach can be adopted when deleting information from a failed LID. The robust update strategy is shown in Table 5.1.

The approach of generating useable mappings, also assumes that the LIDs are not failed for long durations of time. If the failure period is high, then it might be a better approach to use the retry strategy described in [56], and use the first generated update mapping. This is because LIDs which are not failed, but which are part of a set of LIDs in which there are one or more failed LIDs (unuseable mapping) will neverbe used for the duration of the failure.

Robust Query Strategy: The first LID D_i is chosen at random. Successive LIDs $D_{i\oplus 1}, D_{i\oplus 2}, \ldots$ are queried till the required information is retrieved. To save on network traffic we assume that a LID responds to a query only if it has the required location information. It is however interesting to note that even if a LID does send a negative response when it does not have the desired information, the query delay is not reduced. This is because, the replication factor k for the queried mobile is not known to the querying MSC. To make the replication factor of every mobile registered in network known to every MSC in the system is very expensive as the status of the mobile can change very often, thus leading to increased network traffic. The robust query strategy is illustrated in Table 5.2.

5.4.1 Performance Analysis of the Robust Algorithm

The robust load balanced algorithm is shown to be load-balanced in the absence of faults [51]. Given a set of t faults, $1 \leq t \leq n$, the load is not balanced. This is because,

the updated LID "next"to a failed LID on the logical ring of LIDs $D_1, D_2, \ldots, D_n$, with the required information receives more load than other updated LIDs which are not adjacent to a failed LID with information, as it receives its share of queries as well as the queries to the LID which is failed. But this is a specific instance. If we assume that failures distribute uniformly among the LIDs over time, the algorithm is still balanced on the average.

In the presence of failures, a random query strategy in which queried LIDs are chosen at random, might seem a better approach. However, the average as well as the worst-case delay of such a strategy can be shown to be larger than the robust algorithm even in the absence of failures [51]. We now analyze the performance of the robust algorithm proposed, in terms of worst-case, average query delay and call blocking probability.

Proposition 1: *For a given (n, k), the worst-case query delay for proposed algorithm in the presence of t failed LIDs is*

$$T_{max} = \begin{cases} (t+1)\lceil \frac{n}{k} \rceil & t < \beta \\ \beta\lceil \frac{n}{k} \rceil + (t+1-\beta)\lfloor \frac{n}{k} \rfloor & \beta \leq t < k-1 \\ n & t \geq k-1 \end{cases}$$

Proof: The query algorithm picks a LID at random and queries successive LIDs until the required information is found. The maximum number of queries is needed is when t successive LIDs with information LIDs are failed and the query strategy starts in the segment before the first of these failed LIDs. The worst-case number of queries is made when the maximum number of segments of size $\lceil \frac{n}{k} \rceil$ are queried. Observing that for a given (n, k), there are β updated LIDs separated by $\lceil \frac{n}{k} \rceil$ LIDs and $k - \beta$ updated LIDs separated by $\lfloor \frac{n}{k} \rfloor$ LIDs we have

$$T_{max} = \begin{cases} (t+1)\lceil \frac{n}{k} \rceil & t < \beta \\ \beta\lceil \frac{n}{k} \rceil + (t+1-\beta)\lfloor \frac{n}{k} \rfloor & \beta \leq t < k-1 \end{cases}$$

When $t = k - 1$, the worst-case delay is n. The worst-case query delay occurs when $k - 1$ LIDs with information are failed, and the query algorithm queries all these failed LIDs. Using similar reasoning, when $t \geq k$, the worst-case query delay is n. ■

In Figure 5.1, we illustrate the worst-case query delay in a network with 12 LIDs and the replication factor for the queried mobile being 4. D_1, D_3, D_7 and D_{10} have the required location information. Assume that at the time of query D_4 and D_7 are not reachable/failed. The maximum number of queries needed in this case is when the starting LID for query is D_2. The number of queries needed in this case (worst-case) is 9.

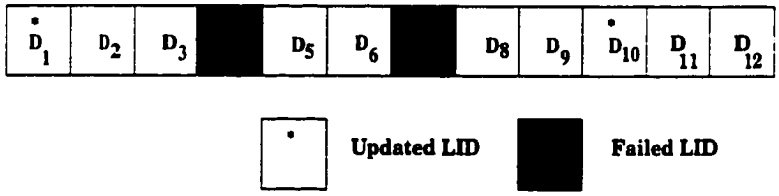

Figure 5.1 Example to illustrate the worst-case query delay with $n = 12$, k =4, $t = 2$.

In Figure 5.2, we compare the worst-case query delays of the robust algorithm, an ideal algorithm which achieves the worst-case delay described in Theorem 1, and a random algorithm in which each LID chosen for query is picked at random. When using this fully random query strategy, the worst-case number of queries is $n - (k - t) + 1$. From the figure, we see that when the failures are low, the robust algorithm performs comparably with the optimal algorithm. When number of concurrent failures in the system is very high then the worst-case query delay of the robust algorithm is much higher than that achieved by the ideal algorithm. However, it is important to note that such an ideal algorithm might not exist. The robust algorithm performs significantly better than the random algorithm in all failure scenarios.

If the failed set of LIDs contains the LIDs with information, the query algorithm fails to retrieve the information. We analyze this case later in this section.

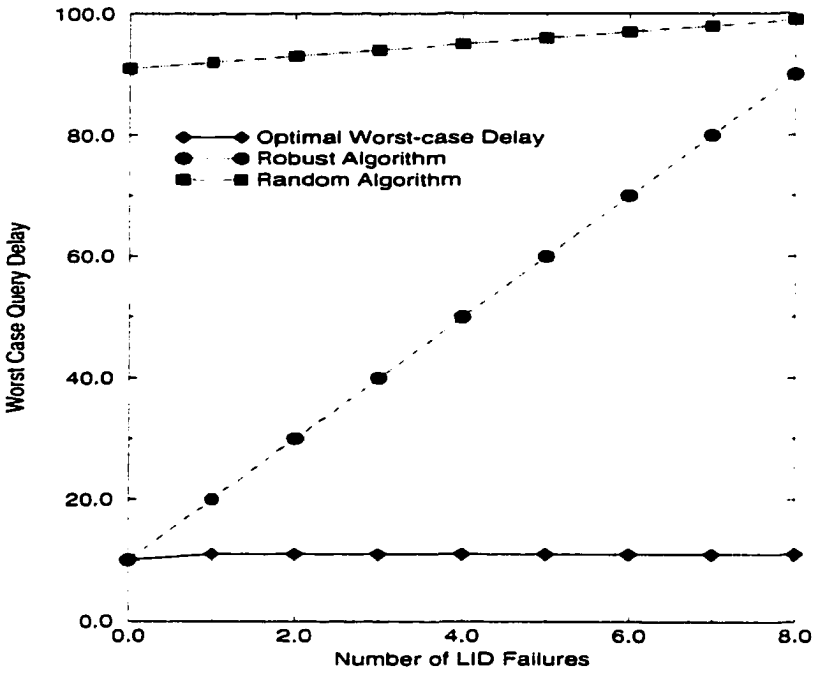

Figure 5.2 Worst-case query Delay Comparison: $n = 100$, $k = 10$, $0 \leq t \leq k-2$

Proposition 2: *For a given (n, k), the average query delay for the proposed algorithm in the presence of t $(0 \leq t < k)$ failed LIDs is $\frac{1}{2}(\alpha+1)(1+\frac{\beta}{n}) + \sum_{m=0}^{t} \frac{\binom{n-(m+1)}{t-m}}{\binom{n}{t}} \frac{mn}{k}$, where α and β are as defined before, and m is the number of "successive" LIDs with information queried, that are failed.*

Proof: Let D_i be the first LID to be queried. If the required information is not available in D_i, $D_{i\oplus 1}, D_{i\oplus 2}, \ldots$ are queried until the information is found. From the description of the update strategy, it can be seen that the number of LIDs the query algorithm queries depends on the number of "successive" LIDs with information the algorithm encounters. Let the first m $(0 \leq m \leq t)$ LIDs with information that are queried be failed. The distance between "successive" updated LIDs is $\lceil \frac{n}{k} \rceil$ or $\lfloor \frac{n}{k} \rfloor$ with probability $\frac{\beta}{k}$ and $1 - \frac{\beta}{k}$ respectively. The expected number of LIDs queried is therefore,

$m(\alpha + \frac{\beta}{k}) = \frac{mn}{k}$. Let ϵ be the random variable denoting the number of LIDs queried before the first failed LID with information. The expected value of ϵ is $\frac{1}{2}(\alpha+1)(1+\frac{\beta}{n})$. The probability that the query algorithm queries m successive LIDs with information that are failed, is $\frac{\binom{n-(m+1)}{t-m}}{\binom{n}{t}}$. Observing that the probability that we start in the segment before the first of these m failures is $\frac{1}{k}$, and that there k LIDs where this string of m failed LIDs can begin, we have the expected number of queries as

$$\frac{1}{2}(\alpha+1)(1+\frac{\beta}{n}) + \sum_{m=0}^{t} \frac{\binom{n-(m+1)}{t-m}}{\binom{n}{t}} \frac{mn}{k}. \tag{5.8}$$

■

Figure 5.3 compares the average query delay of the robust algorithm with an ideal algorithm which achieves the query delay presented in Section 5.2. We see that the performance of the robust algorithm is comparable to the optimal bound in all failure scenarios.

Call Blocking Probability: The robust algorithm is shown to be optimal with respect to call blocking probability [51], when there are no failed LIDs, i.e., $t = 0$. In this section we simulate the performance of the algorithm in the presence of failed LIDs. The simulation was performed on SUN Ultra 1 machines running Solaris 2.5.1 and Solaris 2.6. We assumed a population of 10000 mobiles in the system. We also assume that the replication factor for all the mobiles is the same. When each query is made to locate a mobile's location information, we generate a faulty set of t LIDs. The query algorithm then queries the LIDs for the information and the query delay is measured. If the query delay exceeds the maximum prescribed query delay, the call is dropped. The call blocking probability is thus measured over the course of a large number of trials. We present below the results of two such simulations. In Chapter 4 ([54]) it is shown, that the optimal value for the replication factor is $O(\sqrt{n})$. We therefore fix k as 10 in the examples described below.

In Table 5.3, the performance of the robust algorithm is compared with an ideal algorithm which achieves the lower bound for call blocking probability described in

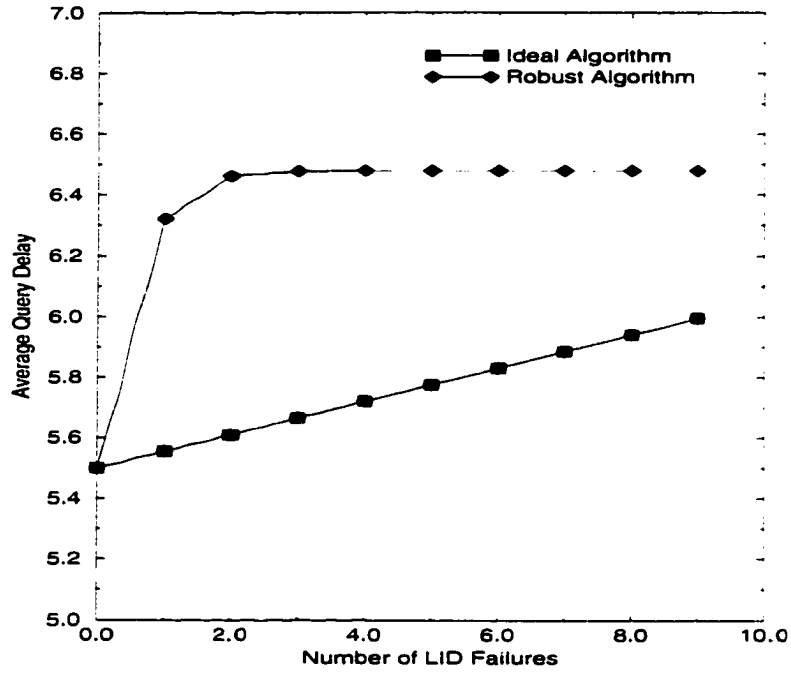

Figure 5.3 Average Query Delay Comparison: $n = 100$, $k = 10$, $0 \leq t < 10$

Section 5.2, for the case when the maximum number of queries acceptable is 10, and $n = 100$, $k = 10$ and $0 \leq t \leq 9$. In Table 5.4, we compare the two algorithms when the maximum acceptable query delay of 6 is lower than $\lceil \frac{n}{k} \rceil$ of 10. From Tables 5.3 and 5.4, it is clear that the robust algorithm achieves the optimal call blocking probability bound described in Theorem 3.

An interesting observation here is that, even though the worst-case query delay of the robust algorithm is higher than the bound derived in Theorem 1, the call blocking probability bound derived in Theorem 3 is being achieved for the test cases. This can be explained as follows. The probability that a large number of failed LIDs with information are queried as n and k increase with respect to t becomes very insignificant, hence making it possible to achieve the bound.

As mentioned before when $k \leq t \leq n$, i.e., there are more number of failures than up-

dates for the queried mobile, there is a chance that the required information is not found and the call is blocked. The probability of calls being blocked due to the information not being found is

$$\frac{\binom{n-k}{t-k}}{\binom{n}{t}}. \tag{5.9}$$

5.4.2 Recovery of LIDs after a failure

Once a LID becomes functional after a failure, due to the mobility of the mobiles its contents might not reflect the current position of its registered mobiles. The LID therefore, enters a period of recovery during which the LID incrementally rebuilds itself with new information. In [56], a fast recovery protocol is presented in which wired network nodes like the MSC are used to retry updates and deletes which are not completed due to the failed state of the LID. We can use a similar approach for LID recovery in the robust location management protocols described in Section 5.4.

5.5 Robust Load Balanced Parallel Querying Algorithms

In [54], a parallel querying load balanced location management algorithm is presented. In this scheme, the mobile population is partitioned into *active* and *relaxed* (less active) mobiles based on their activity. The location information of *active* mobiles is replicated in k_1 LIDs and *relaxed* mobiles are updated in k_2 LIDs, with $k_1 > k_2$. The update strategy in [54] is similar to the strategy of [51]. The query strategy is as follows. On receiving a request to locate a mobile, the MSC assumes that the called mobile is a *active* mobile and queries $\lceil \frac{n}{k_1} \rceil$ successive LIDs $(D_i, D_{i\oplus 1}, \ldots, D_{i\oplus\lceil \frac{n}{k_1} \rceil})$ in parallel, with the first of these LIDs chosen at random. If the called mobile is *active* then it is located in one round of parallel queries. If the called mobile is *relaxed*, then there is a chance that the called mobile is not located in one round of queries. Such mobiles are found in a second round of queries in which LIDs $D_{i\oplus\lceil \frac{n}{k_1} \rceil \oplus 1}, \ldots, D_{i\oplus\lceil \frac{n}{k_2} \rceil}$ are queried in parallel.

This algorithm can be extended to accommodate LID failures using the same strategy described in Section 5.4. In the modified version, the querying algorithm queries $\lceil \frac{n}{k_1} \rceil$

LIDs in parallel in the first round, $\lceil \frac{n}{k_1} \rceil - \lceil \frac{n}{k_2} \rceil$ LIDs in the second round. If the required information is not found, $\lceil \frac{n}{k_2} \rceil$ LIDs are queried in parallel in subsequent rounds, until the desired location information is found. The only communication assumed between the MSC and queried LID is an ACK if the required information is present in the LID. Therefore, an absence of a response from a LID could imply a failed/unreachable LID with/without information or the lack of the desired information. We therefore use $\lceil \frac{n}{k_2} \rceil$ parallel queries in subsequent rounds to increase the chance that called mobile is found irrespective of the mobile's activity state. This pseudo-code of the robust parallel query algorithm is illustrated in Table 5.5. In the next section we analyze the average number of rounds of parallel queries required by the robust parallel querying algorithm to locate *active* and *relaxed* mobiles.

5.5.1 Average Number of Rounds for the Querying Algorithm

In this section, we evaluate the average number of rounds required by the robust parallel query algorithm (RPQA) in the presence of t failed/unreachable LIDs. We denote the average number of rounds for a *relaxed* mobile and *active* mobile by E_t^r and E_t^a respectively. As described in the previous section, a *relaxed* (*active*) mobile's location information is replicated in k_2 (k_1) LIDs.

In the first two rounds the RPQA queries a total of $\lceil \frac{n}{k_2} \rceil$ LIDs, and in subsequent rounds $\lceil \frac{n}{k_2} \rceil$ LIDs are queried per round. To calculate the average number of rounds to locate the information of a *relaxed* mobile, consider an algorithm A, which queries $\lceil \frac{n}{k_2} \rceil$ in every round. The RPQA will need at most one more round of queries to locate the mobile, as the first two rounds query in the RPQA covers $\lceil \frac{n}{k_2} \rceil$ LIDs, which are covered by one round of queries by algorithm A. Therefore, if E_A is the average number of rounds needed by algorithm A to locate a *relaxed* mobile, we have

$$\begin{aligned} E_t^r &\leq 1 + E_A \\ &= 1 + \sum_{m=0}^{t} \frac{(m+1)\binom{n-(m+1)}{t-m}}{\binom{n}{t}} \end{aligned} \tag{5.10}$$

where m is the number of successive failed LIDs with the required location information queried.

Similarly, if E_B is the average number of rounds of an algorithm B, which queries $\lceil \frac{n}{k_1} \rceil$ every round, the average number of rounds needed to locate the information of a *active* mobile in the presence of $t < k_1$ failures for the RPQA is

$$\begin{aligned} E_t^a &\leq 1 + \frac{\lceil \frac{n}{k_1} \rceil}{\lceil \frac{n}{k_2} \rceil} E_B \\ &= 1 + \frac{\lceil \frac{n}{k_1} \rceil}{\lceil \frac{n}{k_2} \rceil} \sum_{m=0}^{t} \frac{(m+1)\binom{n-(m+1)}{t-m}}{\binom{n}{t}} \end{aligned} \tag{5.11}$$

5.6 Summary

We established fundamental bounds to the performance of load-balanced distributed location management algorithms in presence of database and link failures in terms of three important performance metrics. We studied the performance of a load balanced location management algorithm in the presence of faults. We showed that the algorithm performs better than a totally random query algorithm for the worst-case query metric. We then presented a robust parallel querying load balanced algorithm and studied its performance in the presence of faults. In this work, we have normalized the cost of accessing any LID to unity. Fault tolerant load balanced location management algorithms without this same cost restriction is an open problem.

Fast recovery from LID failures is also an important issue to consider in designing location management protocols. We address this issue in the next chapter.

Table 5.1 Pseudo-code for Robust Update Strategy

```
Update(m_i : mobile id);
    γ1 : integer;
    count : integer;
    n : integer; /* Number of Databases */
    k : integer; /* Replication Factor */
    DO_UPDATE : integer; /* Indicates set of LIDs chosen for update are not failed */
    β : integer;
begin
  β = n − k⌊n/k⌋;
  γ1 := prev_starting_database; /*from mobile or search */
   DELETE old information from databases;
   If LID failed retry until DELETE successful;
   while      DO_UPDATE = 0;
    γ1 := uniform_random[1, n]; /*new starting database */
    if(all LIDs for this γ1 are functional)
      DO_UPDATE = 1 && break;
   done;
    if(DO_UPDATE == 0) /* No fault free mapping available */
      Choose the best mapping and use that as γ1;
    count := 0;
    while count ≤ k − β
      UPDATE D_{γ1⊕(count*⌊n/k⌋)};
      count := count + 1;
    done;
    count := 1;
    while count ≤ β
      UPDATE D_{(γ1⊕(k−β)⌊n/k⌋⊕(count*⌈n/k⌉))};
      count := count + 1;
    done;
end
```

Table 5.2 Pseudo-code for Robust Query Strategy

```
Query(m_i : mobile id);
    π_1 : integer;
    count : integer;
    n : integer; /* Number of Databases */
    k : integer; /* Replication Factor */
begin
  π_1 := uniform_random[1,n]; /* 1st db queried */
  count = 0;
  while count ≤ n
    if (info in D_{π_1 ⊕ count})
      return info;
    else
      count := count + 1;
  done;
  return no information found;
end
```

Table 5.3 Call blocking probabilities: $n = 100$, $k = 10$, max. query delay $= 10$, $0 \leq t \leq 9$

Number of Concurrent Failures t	Optimal Bound	Robust Algorithm
0	0.0	0.0
1	0.01	0.01
2	0.02	0.02
3	0.03	0.03
4	0.04	0.04
5	0.05	0.05
6	0.06	0.06
7	0.07	0.07
8	0.08	0.08
9	0.09	0.09

Table 5.4 Call blocking probabilities: $n = 100$, $k = 10$, max. query delay $= 6$, $0 \leq t \leq 9$

Number of Concurrent Failures t	Optimal Bound	Robust Algorithm
0	0.40	0.40
1	0.41	0.41
2	0.41	0.41
3	0.42	0.42
4	0.42	0.42
5	0.42	0.43
6	0.44	0.44
7	0.44	0.44
8	0.45	0.45
9	0.45	0.45

Table 5.5 Fault-Tolerant Parallel Query procedure

```
ParallelQuery(m_i : mobile id);
    π_1 : integer;
    count : integer;
    n : integer; /* Number of Databases */
    k_1, k_2 : integer; /* Replication Factors */
begin
 π_1 := uniform_random[1,n]; /* starting LID for 1st round */
 count = 1;
 if(Query_in_parallel(π_1, ..., π_{1⊕⌈n/k_1⌉}) == success)
   return mobile information; information found */
 else
  if((Query_in_parallel(π_{1⊕⌈n/k_1⌉⊕1}, ..., π_{1⊕⌈n/k_2⌉}) == success)
    return mobile information; /*information found */
  else
    while(count < k_2 − 1)
      if(Query_in_parallel(π_{1⊕((count)⌈n/k_2⌉)⊕1}, ..., π_{1⊕((count+1)⌈n/k_2⌉)}) == success)
        return mobile information;
        count := count + 1;
    done;
end
```

6 FAST RECOVERY PROTOCOLS FOR LOCATION MANAGEMENT

6.1 Introduction

In PCS networks, a location tracking mechanism is needed to locate the position of the mobile hosts in order to establish connections. Current methods require a mobile to report its location to the network when necessary [1]. Some of the location management schemes use a time-based update strategy. Other schemes require a mobile to update its location only if its present location is at least a pre-determined distance away from its location at the previous update. The network stores the location of the mobile in location-information-databases (LIDs) and this information is retrieved during call delivery. When a LID fails calls may have to be dropped as the location information of mobiles registered in the LID is unavailable. The failure of these databases therefore negatively affects the performance of the network. It is therefore imperative that the up-time of the databases be very high and in case of a failure the recovery period is very small. An intermediate link failure partitions the network resulting in the loss of location updates from mobiles. Any location management protocol must therefore take into account the databases or links being in a state of failure at the time of update. In this chapter, we present a new protocol for location management in presence of database and link failures. We analyze the protocol in terms of the average number of calls lost during the time of failure of the database and also during the time when the database is recovering from failure.

6.2 Fast Recovery Protocol

In this section we present a protocol to recover from HLR failures and intermediate link failures. As described in the previous section, when a mobile moves into a different LA it updates its location in the HLR through a location update message. We assume that the mobile can time-stamp (attach a serial number) the LUs. When a MSC receives a LU from the mobile it attempts to communicate the message to the HLR. If the MSC does not receive an acknowledgment from the HLR (failed HLR or link(s)), the MSC times-out and continually sends a LU retry (LU_r) with a period s (we assume that the latency of the wired network is much smaller than the retry period). The s value in retrying for a mobile should be chosen to trade-off protocol cost and lost incoming calls to the mobile during the recovery period. We discuss the factors which decide the choice of s in Section 6.4. Each LU_r has the same time-stamp as the LU it represents. The LU_r reaches the HLR once the HLR (link) becomes functional. If the HLR had failed, then it initiates the recovery process using the messages given below.

Messages sent while recovering from a HLR failure. The following messages are sent by the HLR to the MSC(s) when it initiates its recovery. In the following discussion we use HLR_{id} to accommodate the possibility of mobiles registered with different HLRs being present in the same LA.

- ALL-FREEZE(HLR_{id}): Broadcast by HLR_{id} to its registered MSCs, when the HLR becomes functional after being in a failed state. This freezes the entries of the mobiles belonging to HLR_{id} in the MSC's VLR. A frozen entry cannot be used until the HLR validates the entry. This message prevents the MSC from routing calls to non-existent mobiles in its LA (when the calls arise from mobiles in its own LA).

- UN-FREEZE(HLR_{id}): Broadcast by HLR_{id} to its registered MSCs at the end of its recovery period (after a HLR failure). This un-freezes the frozen entries of mobiles registered with HLR_{id} in the VLR.

- DEL(HLR_{id},MSC_{id},m_k): Sent by the HLR to MSC_{id} when the HLR determines that m_k does not reside in MSC_{id}'s LA. On receiving this message the MSC removes the entry of mobile m_k (belonging to HLR_{id}) from the VLR. The MSC stops retrying for the mobile.

- UPDATE(HLR_{id},MSC_{id},details of m_k): Sent by HLR_{id} to MSC_{id} when it determines that the mobile resides in the MSC's LA. When this message is received the MSC adds m_k's entry to the VLR if it is not already there. Otherwise the MSC un-freezes the entry in the VLR (This happens if the mobile had moved out of the LA during the failure of the HLR and then returned to the LA during the recovery period of the HLR. This entry can now be used for call delivery).

- UPDATE-FREEZE(HLR_{id},MSC_{id},details of m_k): Sent by HLR_{id} to MSC_{id}, when the HLR receives a LU_r from the MSC. The VLR entry for mobile m_k is updated with the data in the message, but is kept in a frozen state. The MSC stops retrying for the mobile.

When the HLR is in its failed state (*off period*), all incoming calls to a registered mobile needing location information from the HLR are lost. If a mobile moves to a new LA say LA_i during this period, then any update sent from MSC_i to the HLR is also lost. Recall that we have assumed that the HLR has stable storage. The disk therefore can be retrieved after the HLR becomes functional. The HLR after the off period enters a period of time when it rebuilds its database (*recovery period*). The HLR determines whether any of its member mobiles have moved to a different LA during its off period. It then rebuilds itself incrementally with a mobiles' new position when it receives the appropriate message from the mobile. We show that this period is bounded by a time s (LU_r period) after which the location of every mobile is known.

We now describe the behavior of the HLR during its recovery period. We will discuss the number of messages sent, the length of the recovery period, and the number of incoming calls lost in each of these cases, pertaining to a single mobile m_k. Two messages are always sent, ALL-FREEZE at the beginning of the recovery and UN-FREEZE at the

end of the recovery. An UPDATE-FREEZE is sent to the MSCs which retry for mobile m_k during the recovery period (an UPDATE is not sent here as the HLR is unsure about the number of LAs visited by the mobile during its failure). In each of the following cases incoming calls to the mobile arriving during the off and recovery period are lost.

Case 1. *A LU (CDR) arrives from MSC_j at time t_l (t_c) < s after the off period, from m_k (Figure 6.1)*

- Explanation: The mobile has moved to a new LA (has made a call) during the recovery.
- Recovery period for the mobile (t_{rec}) is t_l (t_c).
- UPDATE(MSC_j, HLR_{id}, m_k details) sent to MSC_j. DEL(MSC,HLR_{id},m_k) sent to the MSCs which have retried for m_k during the recovery period and unless the MSC is MSC_j. A DEL message is also sent to the mobile's MSC at the time of the HLR's failure unless the MSC is MSC_j.
- If any retry message is received after t_{rec}, a DEL message is sent to the MSC unless it is MSC_j.

Case 2. *At time s after the off period LU_r(s) has(ve) arrived, and no LU (CDR) has arrived from the mobile (Figure 6.2)*

- Explanation: The mobile had moved to at least one LA(s) during the off-period of the HLR and it did not move or make a CDR during the recovery period.
- The HLR compares the time-stamps, sends an UPDATE to the MSC with the largest time-stamp and multi-casts a DEL message to the other MSCs which have retried for m_k. A DEL message is sent to the MSC of the LA where the mobile resided at the time of the HLR failure, unless it is the same as the MSC where the UPDATE is sent.
- The recovery period lasts for a time s after the off-period.

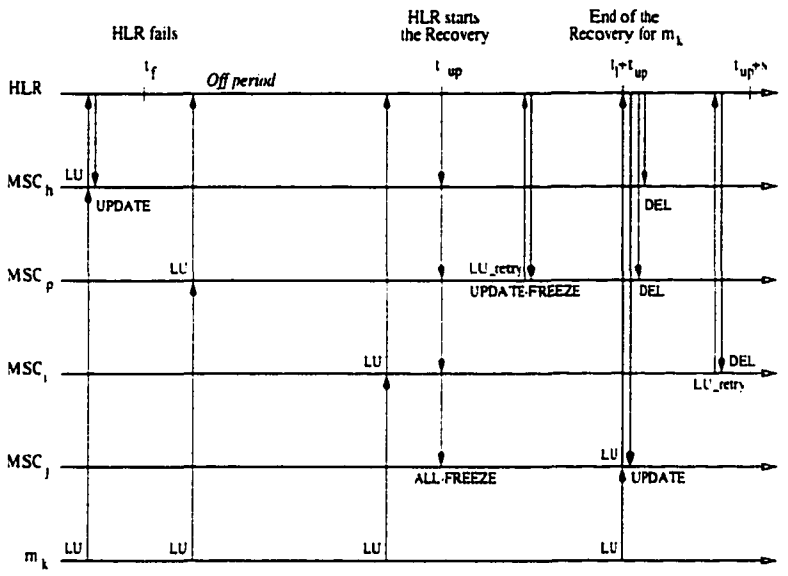

Figure 6.1 Working of the protocol in Case 1

Case 3. *The HLR does not receive any message regarding m_k during a time s after the off-period*

- Explanation: The mobile did not move to a new LA during off-period, nor does it move or make a CDR during the recovery period.
- The recovery period (t_{rec}) for m_k lasts for a time s, which is the maximum time for recovery.

An interesting question that now arises for Case 3 is as follows. How does the HLR know the *retry interval* for every mobile? The answer to this question is clear when we observe that the statistics ($\lambda_a, \lambda_r, \lambda_l, \lambda_c$) used to decide the retry interval of a mobile is also available at the HLR. The HLR maintains the statistics of mobiles in the system, while MSC's maintain the statistics of mobiles in their LA. This is not a large cost for the HLR as this would just imply incrementing counters for the number of CDRs and LUs received for each mobile and this can be made a part of the CDR and LU handling procedures. Therefore, if the HLR does not receive a LU_retry from a mobile within a

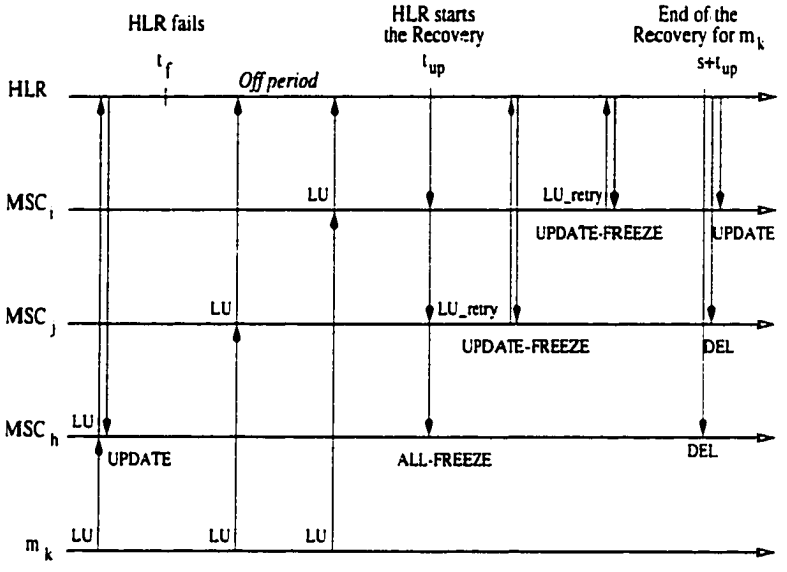

Figure 6.2 Working of the protocol in Case 2

period of time equaling the retry interval for that mobile, the HLR assumes that the mobile has not moved during its failure. The recovery of the HLR for this mobile is now complete.

A LU_r received during the HLR's recovery is logged before sending an UPDATE-FREEZE. If the HLR fails again during the recovery period, MSCs which have not received the UPDATE-FREEZE keep retrying. When the HLR becomes functional again, a new recovery process is started taking its logs from the previous recovery process into account.

6.2.1 Recovery from a Link Failure

If a link between the MSC and the HLR fails[1] we handle it as follows. The HLR now receives a LU_r when it is not in its *recovery period*. The HLR recognizes the failure scenario and initiates the recovery procedure. We present two approaches to recover from a link failure.

[1]In case of a link failure between the mobile's BS and the MSC a similar approach of retrying the message ensures that the LU is delivered at the MSC.

Optimistic approach: If the link failures last for a short period, then very few mobiles update more than once during the time the link is failed. In this case the HLR treats each LU_r as a LU and sends an UPDATE to the MSC (if the time-stamp of the LU is the latest it has received from the mobile). This ensures that the recovery period depends only on the processing time of the messages. The HLR could, however, misdirect a call to a wrong MSC if any mobile does update more than once during the link failure. If the failure time for the link is small then the probability of this misdirection is very low.

Pessimistic approach: If the duration of link failures is long, the HLR adopts the same recovery procedure as it would when recovering from its own failure.

In the case that a link failure occurs during the recovery of the HLR (after a HLR failure), and a mobile is unable to update its position, the HLR has an incorrect entry for the mobile's location at the end of the recovery period. This is however unavoidable as the mobile is disconnected from the network. The mobile's position is recovered when the faulty link becomes active again as explained previously in this section.

6.3 Analysis of the Protocol

In this section, we present an analysis of the protocol we described in the previous section in the context of HLR failures. We derive a closed form expression for the average cost of HLR failure in terms of calls lost per mobile and wasted communication (updates lost). The loss of outgoing calls from a mobile to another mobile is included in calculating the loss of incoming calls to the called mobile. Calls from a mobile to a fixed host do not depend on the state of the HLR as these calls are delivered by the MSC directly to the fixed host through the wired network.

We make the following assumptions. Calls to the mobile arrive at the HLR as a Poisson process with parameter λ_a. The inter-arrival time between successive LU's is assumed to be exponentially distributed with parameter λ_l. The time of HLR failure is

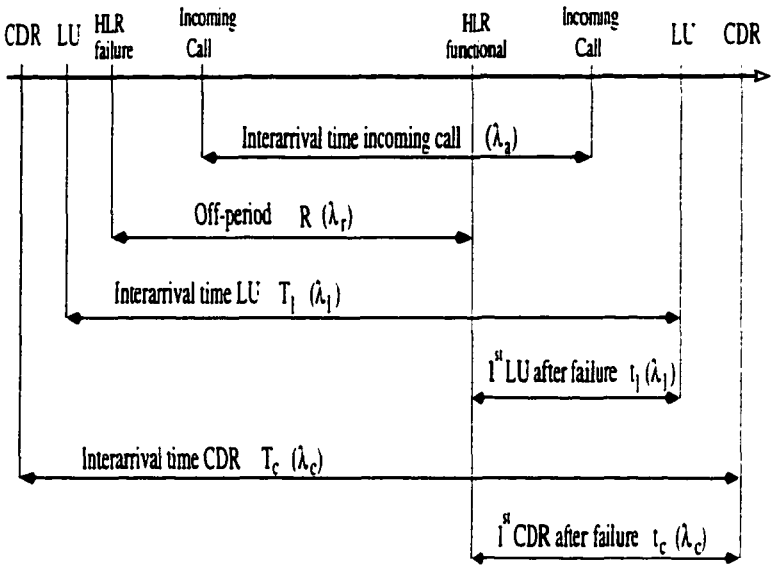

Figure 6.3 Illustration of the various parameters used in the analysis

assumed to be exponentially distributed with parameter λ_r, and is denoted by a random variable R. The mean time to failure (MTTF) of the HLR is T_f. The update retry interval is s. CDR's from the mobile (calls made by the mobile) arrive at the HLR as a Poisson process with parameter λ_c. The parameters are illustrated in Figure 6.3.

6.3.1 Expected Length of Recovery Period

As described in Section 6.2 the recovery period is upper bounded by s. If the HLR receives a LU or CDR from a mobile within a time s then the recovery period is shorter. We now describe the quantities used in our analysis. Let *Rec* be a random variable representing the length of this interval (recovery period). Let the inter-arrival time between two CDRs from the mobile be T_c and let T_l be the inter arrival time between two LUs. T_c and T_l are exponentially distributed by our initial assumption. Let t_c and t_l be the times at which an incoming call or a LU arrives after the HLR becomes functional. By the memoryless property of the exponential distribution, t_c and t_l are exponentially distributed. The length of the recovery period is given by $min(s, t_c, t_l)$. The probability

density function f_{Rec} being

$$f_{Rec}(t) = \begin{cases} (\lambda_c + \lambda_l)e^{-(\lambda_l+\lambda_c)t} & 0 \leq t < s \\ e^{-(\lambda_c+\lambda_l)t}\delta(t-s) & t = s \\ 0 & \text{otherwise} \end{cases}$$

where $\delta()$ is the Dirac Delta function. The average length of the recovery period E_{Rec} is

$$E_{Rec} = \frac{1 - e^{-(\lambda_c+\lambda_l)s}}{\lambda_c + \lambda_l}. \tag{6.1}$$

6.3.2 Expected Loss of Incoming Calls

We now derive expressions for the expected loss of incoming calls to the mobile. These involve calls which arrive during the off period and during the recovery period.

Let X_a be the random variable defining the number of call arrivals to a mobile.The probability that k calls arrive for a given off period r is

$$P(X_a = k|R = r) = \frac{(\lambda_a r)^k e^{-\lambda_a r}}{k!}.$$

Consequently

$$\begin{aligned} P(X_a = k) &= \int_0^\infty \frac{(\lambda_a r)^k e^{-\lambda_a r}\lambda_r e^{-\lambda_r r} dr}{k!} \\ &= \frac{{\lambda_a}^k \lambda_r}{(\lambda_a + \lambda_r)^{k+1}}. \end{aligned}$$

The expected number of calls lost during the off period is

$$\begin{aligned} E_{Aoff} &= \sum_{k=1}^{\infty} \frac{k\lambda_a^k \lambda_r}{(\lambda_a + \lambda_r)^{k+1}} \\ &= \frac{\lambda_a}{\lambda_r}, \qquad \lambda_r > 0\,. \end{aligned} \tag{6.2}$$

Similarly the expected number of incoming calls lost during the recovery period is $\lambda_a E_{Rec}$, where E_{Rec} is the mean length of the recovery period as shown in (6.1). The expected loss of incoming calls is

$$E_A = \frac{\lambda_a}{\lambda_r} + \lambda_a E_{Rec}, \qquad \lambda_r > 0\,. \tag{6.3}$$

6.3.3 Expected Loss of Location Updates

If a MSC happens to send a LU during the off period, it times-out and repeatedly tries again with a period s, until it receives an ack from the HLR. This section analyses the average number of updates lost during the off period.

Let X_l be the random variable denoting the number of retries lost. If the first LU arrives at the HLR at time t after the HLR fails (during the off period), then the probability that there are k retries during the off period is given by

$$\begin{aligned} P(X_l &= k \mid \text{1st update at time t}) \\ &= \int_{t+ks}^{t+(k+1)s} \lambda_r e^{-\lambda_r r} dr \\ &= e^{-\lambda_r t}[e^{-\lambda_r ks}(1 - e^{-\lambda_r s})] \,. \end{aligned}$$

Factoring out the time t we have,

$$\begin{aligned} P(X_l = k) &= \int_0^\infty \lambda_l e^{-\lambda_l t} e^{-\lambda_r t}[e^{-\lambda_r ks}(1 - e^{-\lambda_r s})]dt \\ &= \frac{\lambda_l}{\lambda_r + \lambda_l}[e^{-\lambda_r ks}(1 - e^{-\lambda_r s})] \,. \end{aligned}$$

The expected loss of communication during the off period noting that on an average[2] $\frac{\lambda_l}{\lambda_r}$ LUs arrive during the off period and if for a LU k retries are made we loose $k+1$ updates is:

$$E_L = \frac{\lambda_l^2}{\lambda_r(\lambda_r + \lambda_l)(1 - e^{-\lambda_r s})}, \qquad \lambda_r > 0 \,. \tag{6.4}$$

6.3.4 Expected Cost of a HLR failure

We associate costs of C_A, C_L for the loss of an incoming call and LU, respectively. C_A is a cost associated with the QoS of the system, while C_L is a cost associated with the system management. We have placed emphasis on the QoS in this paper, and hence we associate a higher cost to the loss of incoming calls rather than wasted communication. We however realize that the cost of wasted communication is important in designing a

[2]Using a similar argument as in (6.2)

practical system, and hence include it in an appropriate way in our discussion of the protocol. If T_f is the MTTF then C_{tot} the average cost associated with the protocol is given by

$$C_{tot} = \frac{C_A E_A + C_L E_L}{T_f} . \tag{6.5}$$

The cost function does not include the cost of the ALL-FREEZE and UN-FREEZE messages. This is a fixed cost and does not play any role in determining the minimum of the cost function. It is also important to note that the protocol does not send any more messages for each LU_r it receives in the recovery period than what it would send for each LU it receives in a fully functional state. Hence we do not include the cost of protocol messages during recovery in calculating the total cost of the protocol.

6.4 Discussion

In our discussion we analyze the cost of the protocol in terms of incoming calls and LUs lost with respect to a single mobile, with various values of λ_a, λ_c, λ_l and s, and fixed values of $\lambda_r = 1, C_a = 1, C_l = 0.1$. The value of T_f does not play a very important factor in deciding the optimal value of s as it is just a scaling factor. We fix $\lambda_r = 1$ as λ_r just serves as a time scale.

Case 1: High λ_a, High λ_l. In Figure 6.4, when λ_c is low, the value of s is critical, and the minimum cost is achieved for $s \approx 0.1$. This is about 10% of λ_r. When λ_c is comparable to λ_a, the protocol incurs a high cost when s is small. The cost is lower for larger values of s, but does not vary much with increasing s.

This behavior can be explained as follows. For larger values of λ_c, λ_c determines the length of the recovery period. At lower values of λ_c, however the value of s plays an important part in deciding how long the recovery phase is, and hence a smaller s is better. However a very small retry interval increases the communication cost.

Case 2: Low λ_a, High λ_l. As seen in Figure 6.5, when λ_c is high, a large value of s is preferred. This is because the recovery period is dominated by λ_c, so the recovery

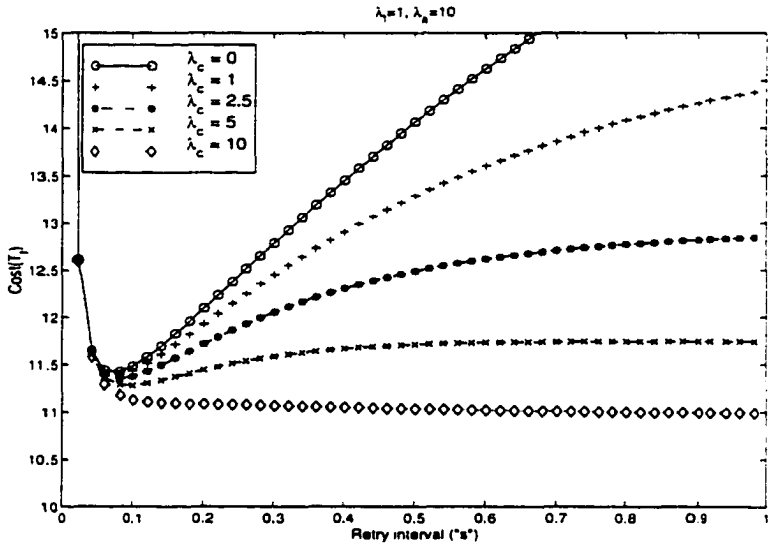

Figure 6.4 Cost vs (λ_c, s) for high λ_l and λ_a

period is almost always lesser than s. Hence in this case it is better not to loose $LU_r's$ during the HLR failure as will be the case when we choose a small s. On the other hand when λ_c is very low, the optimal value of s is small. This is because the cost associated with a lost call is much higher than a wasted update, and as λ_c is low the chance that the length of the recovery process being s is high, and hence higher is the probability of calls being lost during recovery.

Case 3: High λ_a, Low λ_l. In this case (Figure 6.6), the recovery period is determined by λ_c and s as λ_l is low. In other words the recovery period can be considered to be $min(s, t_c)$. As the value of λ_a is high, irrespective of what the λ_c is, we use the lowest possible value for s, taking into account the number of LU_r's generated, to minimize the recovery period.

Case 4: Low λ_a, Low λ_l. The behavior of the system (Figure 6.7) is essentially the same as in the previous case. The overall cost is lower due to a low λ_a. The optimal value of s is low, but is larger than the value of s shown in the previous case.

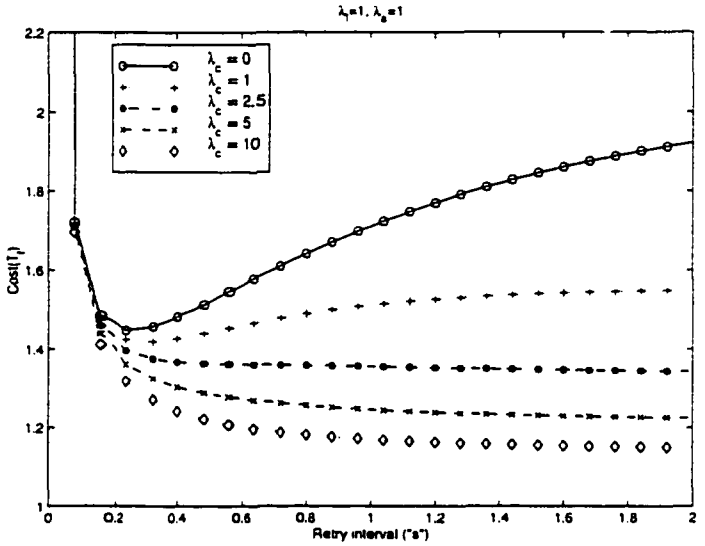

Figure 6.5 Cost vs (λ_c, s) for high λ_l and low λ_a

We are able to upper bound the recovery period by s in the worst case and to a much smaller value than s when the values of λ_c and λ_a are comparable or λ_l is high. In the case our initial assumption of negligible network latency is not true, the upper bound of the recovery period is s + *maximum latency* of the network. In future mobile networks, one can assume that mobiles registered in different HLR's reside in the same LA. The failure of a database should not negatively affect the QoS provided to the other mobiles. A recovery protocol therefore should minimize the usage of wireless bandwidth. The protocol proposed in this paper is simple to implement and does not use any wireless bandwidth while bounding the recovery period which makes it a feasible choice in designing future mobile networks.

6.5 Optimistic Recovery Protocol (ORP)

In this section we describe an optimistic recovery protocol to recover from HLR failures. In this scheme, we do not use the UPDATE_FREEZE message described in Section 6.2. Instead we send an UPDATE for each LU_retry message it receives. This

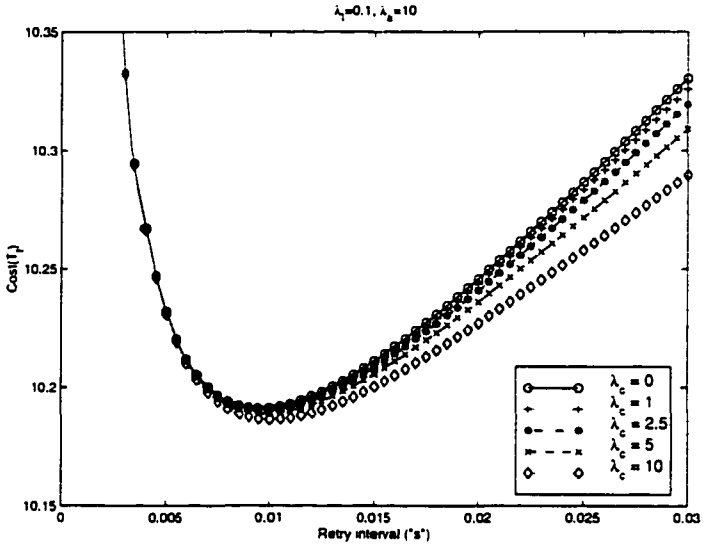

Figure 6.6 Cost vs (λ_c, s) for low λ_l and high λ_a

scheme is useful if in the majority of the cases the HLR failed period is small. In other words, a mobile moves at most once during the HLR failure. The main drawback in this scheme is that the HLR could temporarily have the wrong location information of a mobile if it had moved more than once during the HLR failure and several MSCs retry for the mobile. Now if the first LU_retry received by the HLR is from a MSC which does not represent the latest location of the mobile we could have temporary misdirection of calls. This lasts until the LU_retry representing the latest position of the mobile is received by the HLR. We assume that such calls are dropped. The main consideration therefore in choosing the optimistic approach over the FRP described in Section 6.2 is the cost of such misdirected calls. The rates λ_a, λ_l, λ_c, λ_r determine the number of misdirected calls. To study the performance of the optimistic approach we simulate a mobile environment with various settings of the network parameters.

The simulation was performed on SUN Ultra machines running Solaris 2.5.1 and Solaris 2.6. Poisson rates was considered for λ_a, λ_l and λ_c. The failed for the HLR is assumed to be exponentially distributed. Using the simulations we compare the average

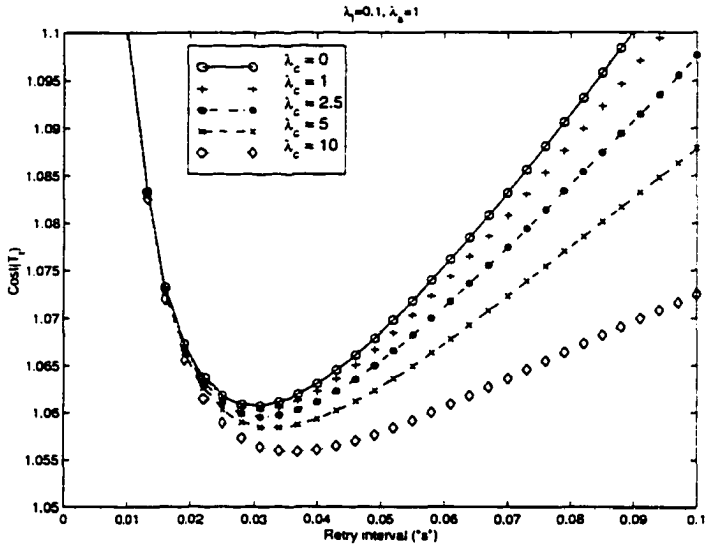

Figure 6.7 Cost vs (λ_c, s) for low λ_l and λ_a

Table 6.1 ORP vs. FRP, $\lambda_a = 1$, $\lambda_l = 0.1$

Optimal s	λ_c	FRP Recovery Interval	ORP Recovery Interval	Average Missed Calls	FRP Cost	ORP Cost
0.042	0.001	0.0442	0.0410	0.0000	10.28	10.33
0.046	2	0.0454	0.0365	0.0000	10.2731	10.3060
0.048	8	0.0416	0.0358	0.0001	10.2477	10.3565

recovery times for the ORP and the FRP, using the optimal retry interval. We associate a cost of 10 for lost and misdirected calls. LUs generated are assigned a cost of 1 as it involves usage of the wireless medium. LU_retrys have a cost of 0.1 as the transmission is in the wireline network. We use a value of $\lambda_r = 1$ as this is just a scaling factor.

When the call arrival rate and the mobility rate of the mobile is low (Table 6.1), ORP functions better than the FRP, as it provides lower recovery intervals with comparable costs. A similar performance is observed (Table 6.2) when the mobility is low and the call arrival rate is high.

Table 6.2 ORP vs. FRP, $\lambda_a = 10$, $\lambda_l = 0.1$

Optimal s	λ_c	FRP Recovery Interval	ORP Recovery Interval	Average Missed Calls	FRP Cost	ORP Cost
0.015	0.001	0.016	0.0134	0.0001	100.6358	100.6554
0.014	2	0.0147	0.0102	0.0001	100.0943	100.9191
0.018	8	0.0186	0.0147	0.0001	100.2734	100.9662

Table 6.3 ORP vs. FRP, $\lambda_a = 1$, $\lambda_l = 1$

Optimal s	λ_c	FRP Recovery Interval	ORP Recovery Interval	Average Missed Calls	FRP Cost	ORP Cost
0.4	0.001	0.3117	0.2573	0.0147	11.2389	11.4697
0.4	2	0.2368	0.2272	0.015	11.2485	11.4881
0.4	8	0.1242	0.1158	0.0148	11.2098	11.5152

In Table 6.3, we compare the two protocols when the mobility rates is high. Here the FRP performs better than the ORP as the improvement in the recovery interval is not significant but the cost is higher when we use the ORP. A similar performance is observed when both the arrival rates and mobility rates are high (Table 6.4).

Table 6.4 ORP vs. FRP, $\lambda_a = 10$, $\lambda_l = 1$

Optimal s	λ_c	FRP Recovery Interval	ORP Recovery Interval	Average Missed Calls	FRP Cost	ORP Cost
0.14	0.001	0.1338	0.0954	0.0498	101.273	102.5928
0.19	2	0.1523	0.1267	0.0673	101.1017	102.6690
0.17	8	0.0947	0.0845	0.0611	101.1718	102.7509

6.6 Recovery from VLR Failures

In this section we use the Fast Recovery Protocol to recover from VLR failures. A VLR failure affects incoming calls to mobiles in the VLR's LA. A VLR needs to be updated when a mobile moves into/out of a LA. We analyze each of these cases if they happen during a VLR failure. The methodology adopted by the VLR to maintain its stable storage is assumed to be similar to the one assumed for the HLR as described in Chapter 1; i.e the VLR is backed up periodically and logs of transactions made are kept between the periodic backups.

- A Mobile m_i moves into the LA of a failed VLR_j:

  The mobile sends a LU to MSC_j. MSC_j propagates the LU to the HLR. The HLR transfers other necessary information back to MSC_j to be updated in VLR_j. Once the MSC receives the information from the HLR and the mobile, it tries to update the VLR. These messages are time-stamped with the same time-stamp sent by the mobile. As the VLR is failed at this time the MSC retries the update with a period s chosen as described in Section 4.3.

- A Mobile m_i moves out of a LA with a failed VLR, VLR_j:

  In such a situation the HLR sends a DEL message to MSC_j which in turn deletes m_i's information in VLR_j. The MSC retries this message with a period s.

Once the VLR becomes functional, it recreates its state at the time of failure with the help of the checkpoint and logs. It then starts its recovery procedure, which lasts for a time which is bounded by the mobile's retry interval s. The actual recovery time of the VLR for a mobile is given by $min(t_c, t_l, s)$, where t_c, and t_l have the same meaning as in Section 6.4. For each mobile if the VLR does not receive any update within a time s (the mobile's retry interval) after becoming functional, it uses the state at the time of its failure as its present state. Under the assumption that the VLR does not deliver any calls unless it is sure of the position of the mobile, all the calls to the mobile during the

recovery period for the mobile are lost. The expected recovery period can be calculated as in Section 4.3.

6.7 Recovery in a Distributed LID Architecture

The distinguishing feature of the FRP is that we use existing network elements like the MSC to aid in the recovery of the databases. This is independent of whether the database is centralized or distributed. In this section we adapt the protocol to recover from failures in a distributed LID architecture.

The distributed architecture shown in Figure 6.8 is proposed in Chapter 3. There are n databases with identical storage and access capabilities. These databases are connected to each other and to the MSCs through a wired network. In this architecture, a mobile's location information is updated in k of the n LIDs. k is called the replication factor for the mobile. The replication factor is chosen to trade of update and query costs for the mobile.

Using the update and query strategies described in Chapter 3, we now describe a distributed recovery protocol.

Recovery Protocol for Distributed LID Architecture: As described earlier, after a failure a LID reconstructs the location information of the registered mobiles from its logs. Unlike the HLR (IS-41 standard) which receives just one message modifying its data, a LID in the distributed architecture described in [51], receives two messages which modify its data, namely the DELETE message and the UPDATE message. The DELETE message is used by the MSC to delete a mobile's information from a LID, and the UPDATE message is used to update a mobile's location information in a LID. Therefore, a MSC must not only retry the UPDATE message but also the DELETE message. If at the time of update or deletion a LID is failed, the MSC retries the message with the appropriate time-stamp. As before, the retry interval is decided by the activity of the retrying mobile. The LID on becoming functional again, enters a

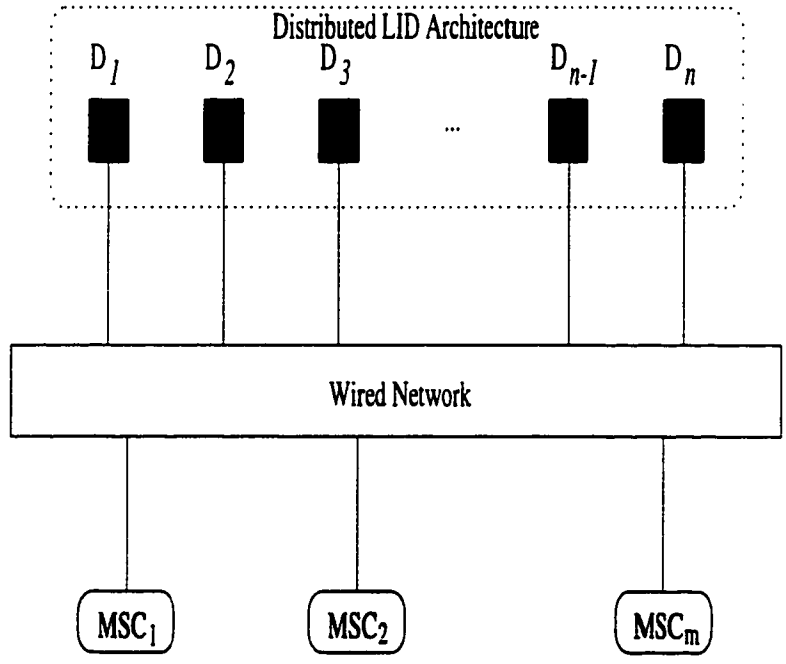

Figure 6.8 A distributed location database architecture

recovery phase, wherein it processes the various DELETE and UPDATE retry messages it receives and incrementally rebuilds its data for each mobile. Using a similar analysis as shown in Section 6.3 we see that the recovery of each mobile is bounded by its retry interval.

It is to be noted however, that the recovery process described above architecture specific. The procedure described above is suited for the update and query strategies described in Chapter 3. To use it in a different architecture, we have to adapt the protocol taking to consideration the update and query strategies.

6.8 Summary

We have presented a new protocol to recover from database and link failures in mobile networks. The protocol is simple and is easy to implement in current networks, in the sense that it does not require the use of special protocol specific network nodes. The

protocol does not require use of wireless bandwidth for recovery messages. [32] requires the mobiles to inform the HLR of its location periodically (irrespective of the state of the HLR) to reduce the recovery period after a HLR failure. The recovery protocol presented does not send any message when the HLR has not failed. We have shown by analysis that the length of the recovery phase for a mobile is upper bounded by its retry interval s. We have presented a methodology to analyze the system, and have shown the values of s which minimize the cost in different system scenarios. An optimistic recovery protocol which functions well in low mobility cases was proposed. We demonstrated the feasibility FRP for VLR recovery and recovery in distributed architectures.

7 CONCLUSIONS AND FUTURE WORK

PCS networks is growing at a fantastic rate. More and more users are coming into the fold of PCS networks every day. The location management problem is a fundamental and unique problem associated with wireless networks. As the number of users increase in the network better architectures for the network are the need of the hour. The current standard uses a centralized database architecture to store location information. This poses restrictions on the scalability of the network as well as the reliability of the network. As the number of users increase in the network, the load on the database also begins to play an important role in QoS provided by the network. In this proposal we have proposed novel algorithms for distributed dynamic load balanced location management and fault recovery.

7.1 Conclusions

7.1.1 The Issue of Load Balance

The first part of this dissertation studied the feasibility of load balanced location management algorithms for PCS networks. In Chapter 3 fundamental analytical bounds were derived for any load balanced location management algorithm in which the access cost for each database is the same. We measured the performance of the algorithm with respect to three standard performance metrics namely worst-case query delay, average query delay and call blocking probability. The three metrics effectively quantify the performance of the network and give us a complete picture about the QoS that can be expected from the network. We then proceeded to give a construction of optimal

update and query strategies which achieve these bounds. The algorithm uses a generic distributed database architecture. In this chapter we also describe update algorithms in which the mapping between the mobiles and the databases where their location information is updated in is either static or predominantly static. We analyze the performance of these two strategies.

In Chapter 4, we analyze the problem of optimal replication factor in PCS networks. We optimally choose the number of databases a mobile's location information is updated in based on the level of activity of the mobile. We show that the optimal replication factor is of $O(\sqrt{n})$, where n is the number of databases in the network. The protocol presented does not use a centralized storage medium to store information about the activity states of mobiles. We use existing network elements to dynamically maintain the activity of the mobiles. We also show a methodology using which the mobile population can be partitioned based on their activity states.

7.1.2 The Issue of Fault Tolerance

The second part of the dissertation deals with the effect failures in PCS networks and methodologies to recover soon from their undesirable ramifications. In Chapter 5, we derive analytical bounds satisfied by any load balanced location management algorithm in the presence of database failures or link failures which manifest as database failures. We then present robust update and query strategies based on the strategies described in Chapter 3 and Chapter 4. We also compare the performance of these robust algorithms with respect to the bounds derived and show that they come close to the bounds when the number of failures in the system is low. We also show that the robust protocols anyway perform much better than a totally random update and query strategy.

In Chapter 6, we developed a fast recovery protocol, to recover from location database/link failures in PCS networks. The basic idea of the protocol was to use the MSC We first gave a detailed methodology to recover from failures in a network based on the IS-41 PCS architecture. We analyzed the different network scenarios possible and showed the working of the protocol in each of the cases. We also presented an optimistic protocol to

recover from failures which do not last for a long time. We then extended the protocol to accommodate VLR failures. We also demonstrated the protocols versatility by extending it to distributed architectures. We analyzed the cost of the protocol and showed through simulation that it performs better than the IS-41 standard recovery protocol.

7.2 Directions for Future Research

Mobile networks is probably the fastest growing area of networking. In this section we illustrate some of the active areas of research which is being undertaken by various research agencies and institutions.

The load balancing research reported in this dissertation assumes that the databases in the network have identical access and processing capabilities. This clearly may not be true for an arbitrary network. To design a protocol which balances the load optimally over such a network, together with optimizing delays is a very interesting avenue for future research. In such a network, it may not be possible to balance both the updates and queries individually over the databases. We may have to balance the overall number of updates and queries received by each database in the network.

Another current area of research is the so called *Ad-Hoc Mobile Networks*. In such a network, the wired network backbone assumed in this dissertation is absent. Instead, the mobiles communicate with each other through a wireless link. The unique aspect of such a network is the randomness of the network topology which is constantly changing. Finding the optimal path to route messages in such a network is an interesting research issue. The main aim in this research is to contact the minimum number of mobiles in sending a message to the destination mobile.

Using wireless networking for transferring data is another area of research. TCP/IP is the widely accepted protocol for computers to communicate in wired networks. TCP for wired networks when used directly on wireless networks does not give satisfactory performance. This is because of the nature of wired networks. In wired networks interference is not a problem, congestion is. Therefore whenever TCP senses a delay in the

arrival of packets, it adopts a strategy which of exponential back off; i.e. the protocol waits for longer and longer time for the packet to arrive before attempting retransmission. In wireless networks however links are susceptible to interference, and in such a scenario packets are lost. An exponential back off is not an appropriate strategy. In such a case, aggressive retransmission is a better approach.

Future mobile networks will need a high bandwidth and feasibility for global roaming. Designing architectures for such systems is an interesting and active area for research. When dealing with global roaming in satellite systems we will have to deal with the movement not only among mobiles but also the base stations themselves.

Adapting ATM to wireless has been an active research area. ATMs have been used to ensure QoS in wired networks. Therefore several research efforts are being undertaken to adapt this technology into the wireless medium to account for heavy traffic in the future.

With the launch of the Iridium satellite system by Motorola, Globalstar system by Qualcomm, using satellites for global communication is proven to provide global roaming. Mobility issues in such a system also provides scope for active research. Fault tolerance issues in wireless networks is also an emergent area.

The future communication network will be truly diverse with different protocols all existing together. Wireless networks will be an inherent part of this network as it provides the possibility of anywhere communication. The future for wireless networking is therefore bright and strong.

APPENDIX

Proposition A1: *Consider an algorithm which randomly updates k out of n databases, and queries the databases in a random order. The average query delay of this algorithm is given by* $E(n,k) = \frac{n+1}{k+1}$.

Proof: We prove the claim through induction. The claim is true for n = 1. Assume that the claim is true for some n and $1 \leq k \leq n$. We now prove that it is true for the case when the number of databases is $n+1$.

Using the fact that the first database queried has the queried mobile's information with probability $\frac{k}{n+1}$ we have

$$\begin{aligned} E(n+1,k) &= \frac{k}{n+1} + (1 - \frac{k}{n+1})(1 + E(n,k)) \\ &= 1 + (1 - \frac{k}{n+1})E(n,k) \end{aligned} \tag{7.1}$$

for $1 \leq k \leq n$. Using the value of $E(n,k)$ from the claim in this recursion we have the result for $n+1$ and $1 \leq k \leq n$. When $k = n+1$, we have $E(n+1,n+1) = \frac{n+2}{n+2} = 1$, which also satisfies the claim. ■

Since $\frac{n+1}{k+1} \geq \frac{n+k}{2k}$ for $1 \leq k \leq n$, the random algorithm in Proposition A1 has a larger average query delay than the algorithm proposed in Section 3.3.

BIBLIOGRAPHY

[1] I. F. Akyildiz, J. S. M. Ho, "On Location Management for Personal Communication Networks", *IEEE Communications Magazine*, Vol.34, no.9, pp. 138-145, September 1996.

[2] S. Tabbane, "Location Management for Third-Generation Mobile Systems", *IEEE Communications Magazine*, Vol.35 no.8, pp. 72-84, August 1997.

[3] EIA/TIA, "Cellular Radio-Telecommunications Intersystems Operation, PN-2991", November 1995.

[4] D. Goodman, "*Wireless Personal Communications Systems*", Addison-Wesley Publishing Company, Reading, MA, September 1997.

[5] D. A. Patterson, G. A. Gibson, R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", in Proceedings of *ACM SIGMOD International Conference on Management of Data*, ACM Press, New York, pp. 109-116, May 1989.

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

[7] I. Katzela, M. Naghshineh, "Channel Assignment Schemes for Cellular Mobile Telecommunication Systems", *Personal Communications Magazine*, vol.3, no.3, pp. 10-31, June 1996.

[8] R. Caceres, V. N. Padmanabhan, "Fast and Scalable Handoffs for Wireless Internetworks", Proceedings of the *Second Annual IEEE/ACM International Conference on Mobile Computing and Networking*, pp. 56-67, November 1996.

[9] M. Karol, M. Veeraraghavan, and K. Y. Eng, " Implementation and Analysis of Handoff Procedures in a Wireless ATM LAN", Proceedings of *IEEE Globecom'96*, London, U.K., pp. 216-223, November 1996.

[10] J. R. Lorch, A. J. Smith, "Reducing Processor Power Consumption by Improving Processor Time Management in a Single-User Operating System", Proceedings of *Second Annual IEEE/ACM International Conference on Mobile Computing and Networking* , pp. 143-154, November 1996.

[11] S. Rajagopalan, B. R. Badrinath, "An Adaptive Location Management Strategy for Mobile IP", Proceedings of *First Annual IEEE/ACM International Conference on Mobile Computing and Networking* , pp. 170-180, November 1995.

[12] V. Anantharam, M. L. Honig, U. Madhow, V. K. Wei, "Optimization of a Database Hierarchy for Mobility Tracking in a Personal Communications Network", *Performance Evaluation*, vol.20, no.1-3, pp. 287-300, May 1994.

[13] J. Z. Wang, "A Fully Distributed Location Register Strategy for Universal Personal Communication Systems", *IEEE Journal on Selected Areas of Communication*, vol. 11, no.6, pp. 850-860, August 1993.

[14] B. R. Badrinath, T. Imielinski, A. Virmani, "Locating Strategies for Personal Communication Network", Proceedings of the *Workshop on Networking of Personal Communication Applications*, pp. 22-24, December 1992.

[15] J. S. M. Ho, I. F. Akylidiz, "Dynamic Hierarchical Database Architecture for Location Management in PCS Networks", *IEEE/ACM Transactions on Networking*, vol. 5., no. 5., pp. 646-660, October 1997.

[16] M. Fujioka, S. Sakai, H. Yagi, "Hierarchical and Distributed Information Handling for UPT", *IEEE Network Magazine*, pp. 50-60, November 1990.

[17] S. K. Das, S. K. Sen, "Adaptive Location Prediction Strategies Based on a Hierarchical Network Model in Cellular Mobile Network", Proceedings of the *Second International Mobile Computing Conference*, pp. 131-140, March 1996.

[18] S. Dolev, D. Pradhan, J. Welch, "Modified Tree Structure for Location Management in Mobile Environments", Proceedings of *IEEE INFOCOM*, pp. 530-537, 1995.

[19] H. Harjono, R. Jain, S. Mohan, "Analysis and simulation of a cache-based auxiliary location strategy for PCS", Proceedings of the *IEEE Conference on Networks and Personal Communications*, pp. 1-5, March 1994.

[20] R. Jain, Y.-B. Lin, S. Mohan, "A Caching Strategy to Reduce Network Impacts of PCS", *IEEE Journal on Selected Areas of Communication*, vol.12, no.8, pp. 1434-1445, October 1994.

[21] R. Jain, Y.-B. Lin, "An auxiliary user location strategy employing forwarding pointers to reduce network impacts of PCS", Proceedings of the *IEEE International Conference on Communications*, pp. 740-744, June 1995.

[22] R. Jain, Y.-B. Lin, S. Mohan, "A Forwarding Strategy to Reduce Network Impacts of PCS", Proceedings of *IEEE INFOCOM*, pp. 481-489, April 1995.

[23] P. Krishna, N. H. Vaidya, D. K. Pradhan, "Static and Adaptive Location Management in Mobile Wireless Networks", *Journal of Computer Communications*, vol.19, no.4, pp. 321-334, March 1996.

[24] J. S. M. Ho, I. F. Akylidiz, "Local Anchor Scheme for Reducing Location Tracking Costs in PCNs", Proceedings of the *First Annual IEEE/ACM International Conference on Mobile Computing and Networking*, pp. 181-193, November 1995.

[25] Z. Naor, H. Levy, "Minimizing the Wireless Cost of Tracking Mobile Users: An Adaptive Threshold Scheme", Proceedings of *IEEE INFOCOM*, pp. 720-727, March 1998.

[26] S. K. Das, S. K. Sen, "A New Location Update Strategy for Cellular Networks and its Implementation using a Genetic Algorithm", Proceedings of the *Third Annual IEEE/ACM International Conference on Mobile Computing and Networking*, pp. 185-194, September 1997.

[27] N. Shivakumar, J. Widom, "User Profile Replication for Faster Location Lookup in Mobile Environments", Proceedings of the *First Annual IEEE/ACM International Conference on Mobile Computing and Networking*, pp.161-169, November 1995.

[28] R. Holzman, Y. Marcus, D. Peleg, "Load Balancing in Quorum Systems", *SIAM Journal of Discrete Mathematics*, vol. 10, no. 2, pp. 223-245, May 1997.

[29] R. Prakash, M. Singhal, "A Dynamic Approach to Location Management in Mobile Computing Systems", Proceedings of the *Eighth International Conference on Software Engineering and Knowledge Engineering (SEKE'96)*, pp. 488-495, June 10-12, 1996.

[30] R. Prakash, Z. Haas, M. Singhal, " Load-Balanced Location Management for Mobile Systems using Dynamic Hashing and Quorums", *Technical Report UTDCS-05-97*, University of Texas at Dallas, October 1997.

[31] D. Peleg, A. Wool, "Crumbling Walls: A Class of Practical and Efficient Quorum Systems", Proceedings of the *14th ACM Symposium on Principles of Distributed Computing*, pp. 120-129, Ottawa, August 1995.

[32] ETSI/TC-SMG, "Digital cellular telecommunications system; Restoration procedures", *Version 5.0.0, GSM Technical Specification (GSM 03.07), ETSI*, November 1996.

[33] Z. Haas, Y.-B. Lin, "On Optimizing the Location Update Costs in the Presence of Database Failures," *Wireless Networks*, vol.4, no.5, pp.419-426, 1998.

[34] Y. Fang, I. Chlamtac, H. B. Fei, "Optimal Location Update Scheme to Combat Database Failure for PCS Networks," in the *Proceedings of the IEEE Global Communications Conference*, Sydney, Australia, November 1998.

[35] Y.-B. Lin, "Failure Restoration of Mobility Databases for Personal Communication Networks", *Wireless Networks*, vol. 1, pp. 365-372, 1995.

[36] B. Awerbauch, D. Peleg, " Concurrent online tracking of mobile users", Proceedings of *SIGCOMM, Symposium of Communication, Architecture and Protocols*, pp. 221-233, October 1993.

[37] S. Mohan, R. Jain, "Two user location strategies for PCS", *IEEE Personal Communications Magazine*, vol.1, no.1, pp. 42-50, 1994.

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

[39] Y.-B. Lin, "Determining the User Locations for Personal Communications Services Networks", *IEEE Transactions on Vehicular Technology*, vol. 43, no. 3, pp. 466-473, August 1994.

[40] D. O. Awduche, A. Ganz, A. Gaylord, "An Optimal Search Strategy for Mobile Stations in Wireless Networks", Proceedings of *IEEE International Conference on Universal Personal Communications*, pp. 946-950, September 1996.

[41] A. Bar-Noy, I. Kessler, " Tracking Mobile Users in Wireless Networks", Proceedings of *IEEE INFOCOM*, pp. 1232-1239, 1993.

[42] A. Bar-Noy, I. Kessler, M. Sidi, "To Update or not to Update", Proceedings of *IEEE INFOCOM*, pp. 570-576, 1993

[43] G. Cho, L. Marshall, " An Efficient Location and Routing Scheme for Mobile Computing Environments", *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 868-879, June 1995.

[44] C. Rose, R. Yates, "Minimizing the Average Cost of Paging Under Delay Constraints", *Wireless Networks*, vol. 1, no. 2, pp. 211-219, 1995.

[45] J. S. M. Ho, I. F. Akylidiz, "Mobile User Location Update and Paging Under Delay Constraints", *Wireless Networks*, pp. 413-425, 1995.

[46] G. Wan, E. Lin, "A Dynamic Paging Scheme for Wireless Communication Systems", Proceedings of the *Third Annual IEEE/ACM International Conference on Mobile Computing and Networking*, pp. 195-203, September 1997.

[47] C. Rose, "Minimization of Paging and Registration Costs Through Registration Deadlines", Proceedings of the *IEEE International Conference on Communications*, pp. 735-739, June 1995.

[48] D. Goodman, P. Krishnan, B. Sugla, "Minimizing Queuing Delays and Number of Messages in Mobile Phone Location", *Mobile Networks and Applications*, vol. 1, pp. 39-48, 1996.

[49] A. Bar-Noy, I. Kessler, M. Naghshineh, "Topology-based Tracking Strategies for Personal Communication Networks", *Mobile Networks and Applications*, vol. 1, pp. 49-56, 1996.

[50] B. Awerbauch, D. Peleg, "Online Tracking of Mobile Users", *Journal of the Association for Computing Machinery*, vol.42, no.5, pp. 1021-1058, September 1995.

[51] G. Krishnamurthi, M. Azizoglu, A. K. Somani, "Optimal Location Managemen Algorithms for Mobile Networks", Proceedings of the *Fourth Annual IEEE/ACM International Conference on Mobile Computing and Networking*, pp. 223-232, Dallas, October 25-30, 1998.

[52] M. R. Garey, D. S. Johnson, "*Computers And Intractability, A Guide to the Theory of NP-Completeness*", W. H. Freeman and Company, San Francisco, 1979.

[53] T. H. Cormen, C. E. Leiserson, R. L. Rivest, "*Introduction to Algorithms*", McGraw-Hill , New York, 1990.

[54] G. Krishnamurthi, S. Chessa, A. K. Somani, "Optimal Replication of Locati on Information in Mobile Networks", To appear in the proceedings of the *IEEE International Conference on Communications*, Vancouver, Canada, June 6-9, 1999.

[55] G. Krishnamurthi, A. K. Somani, "Effect of Failures on Optimal Location Management Algorithms ", To appear in the proceedings of the *IEEE Fault Tolerant Computing Symposium*, Madison, June 15-18, 1999.

[56] G. Krishnamurthi, S. Chessa, A. K. Somani, "Fast Recovery Protocols for Database/Link Failures in Mobile Networks", Proceedings of the *Seventh Annual International Conference on Computer Communications and Networking*, pp. 32-40, LaFayette, October 12-15 1998.