Efficient Location Management in Mobile Wireless Networks *

P. Krishna † N. H. Vaidya D. K. Pradhan

Department of Computer Science

Texas A&M University

College Station, TX 77843-3112

E-mail: {pkrishna,vaidya,pradhan}@cs.tamu.edu

Fax: (409) 862-2758

Phone: (409) 862-3411

Technical Report # 96-030

July 12, 1996

Abstract

Location management schemes presented in the personal communication network standards (e.g., EIA/TIA Interim Standard 41 (IS-41)) are not efficient. We propose forwarding techniques that augment the IS-41 scheme to provide efficient location management. We develop analytical models to compare the performance of the proposed approach with the IS-41 scheme. Analysis shows that forwarding significantly reduces the total network load due to location management. However for some network parameter values, the search cost of the forwarding scheme is very high. In order to reduce the search cost, we present a strategy to perform *search-updates*. A search-update occurs after a successful search, when the location information corresponding to the searched mobile host is updated at some hosts. Analysis shows that for some network parameter values, performing search-updates significantly reduces the search costs and total network load.

*Research reported is supported in part by AFOSR. Preliminary version of this report appears as technical reports [17, 18].

†Direct all correspondence to P. Krishna.

1 Introduction

It is expected that existing wireless cellular networks will be upgraded for personal communication services (PCS) [7, 9, 21]. An important issue in mobile wireless networks is the design and analysis of location management schemes. In order to communicate with any particular user, it is first necessary to locate the user in the network. This is due to the fact that the users are mobile and they could be anywhere in the area covered by the network.

An analysis in [20] shows that if location updates are to occur on each cell crossing the resulting signalling load will have a major impact on the load of the network. The additional signalling traffic on the SS7 signalling system (capacity of 56 Kbps) is expected to be 4-11 times greater for cellular networks than for ISDN and 3-4 times greater for future personal communication networks (PCN) than for cellular networks. The signalling load due to updates alone increases network load by 70%. Thus location updates will become a major bottleneck at the switches such as SS7, and hence mechanisms to control the cost of location update are needed. It is thus evident that new efficient location management strategies need to be investigated for personal communication services (PCS).

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

This report studies the effect of forwarding pointers and search-updates on location management in Personal Communication Networks (PCN). Location management for PCS utilize the fact that there is a home location server (HLS) for every mobile host. There are existing standards for carrying out location management using home location servers (HLS), e.g., (EIA/TIA) Interim Standard 41 (IS-41), and Global System for Mobile Communications (GSM) in Europe [21]. However, these schemes are not efficient, due to increase in network load and location management costs. To overcome these drawbacks, this work presents location management strategies using forwarding pointers in addition to HLS. We present two heuristics to limit the number of forwarding pointers traversed during a search, namely, *movement-based* and *search-based*. We show that significant improvement can be obtained using forwarding pointers in addition to HLS. The report also presents a strategy to perform *search-updates*. A search-update occurs after a successful search, when the location information corresponding to the searched mobile host is updated at some hosts.

Analysis shows that performing search-updates significantly reduces the search costs. However, search-updates also increase the total network load. This extra network load is due to forwarding pointer maintenance.

The performance of the proposed heuristics depend on (i) the relative cost of setting and traversing the forwarding pointers (parameter α), and, (ii) call and mobility patterns of the user (parameter r). Through analysis, we show that among the schemes considered, there is no single location management scheme that performs well for all values of r and α. However, among the schemes considered, we determine the best location management scheme for each environment, as shown in Figure 1. $\mathcal{M}$ is a parameter that determines the performance of the movement-based heuristic scheme. It will be explained in detail in Section 4.2.2.

<table>
<tr><th>α
r</th><th>Low</th><th>High</th></tr>
<tr><td>Low</td><td>Search-based with Search-Updates</td><td>Movement-based (M<5)</td></tr>
<tr><td>Moderate</td><td rowspan="2">Movement-based with Search-Updates (M<5)</td><td>Movement-based (M<3)</td></tr>
<tr><td>High</td><td>Search-based</td></tr>
</table>

Figure 1: Best Location Management Scheme

On the downside, forwarding pointers are prone to failures anywhere along the chain of pointers. Omission/corruption of a forwarding pointer could cause the host to be intractable. We propose cost-effective techniques for fault tolerance and automatic recovery of the network, and also for forwarding pointer maintenance. We then show that the memory overhead due to forwarding pointers is insignificant when compared to savings in terms of network load.

This report is organized as follows. Review of related literature in location management is presented in Section 2. Section 3 presents a network architecture for a distributed system with mobile hosts. Section 4 presents the proposed location management scheme using forwarding pointers, and the corresponding performance analysis. Section 5 presents the search-update strategy and the corresponding performance analysis. Section 6 presents schemes to make the proposed location management scheme robust. Memory overhead analysis is presented in Section 7. Section 8 discusses algorithms to determine the call-mobility ratio of the users. The work presented in this report is compared with other centralized approaches in Section 9. Summary is presented in Section 10.

2 Related Work in Location Management

Numerous location strategies have been proposed in the recent years. One of the earlier works which dealt with object tracking was done in 1986 by Fowler [8]. Fowler deals with techniques to efficiently use forwarding addresses for finding decentralized objects. Our research borrows the idea of manipulating forwarding pointers upon a successful search.

Awerbuch et.al. proposed a theoretical model for online tracking of mobile hosts [1]. Their architecture consists of a hierarchy of "m-regional matching directories". In their scheme, forwarding pointers and regional matching directories are used to enable localized updates and searches.

Spreitzer et.al. proposed a network architecture which consists of *user agents* and a *location query service* (*LQS*) [22]. There is a dedicated user agent per user and these user agents are responsible to forward any communication to or from the user. This scheme is mainly aimed for local networks – the kind used within building premises. As the number of hosts in a network increase, it might not be efficient to have a dedicated user agent per user.

Wu et. al. dealt with the idea of caching location data at the Internet Access Point(IAP) [25]. Here, the IAP will maintain location data of some of the hosts. This becomes useful when optimal routing decisions are to be taken. If the IAP does not have an entry for a host, the message is forwarded to the *Mobile Router* (*MR*) which maintains information of all the hosts. It is a very simple idea that will be effective for local networks. However, when the network sizes increase, the *MR* will become a serious bottleneck, and one has to resort to more efficient location management techniques.

Ioannidis et.al. proposed IP-based protocols for providing continuous networks access for mobile computers using caching and forwarding technique [11]. However, these protocols are primarily proposed for a campus environment with mobile computers.

Badrinath et.al. examine strategies that reduce search costs and control the volume of location updates by employing user profiles [2]. Their architecture consists of a hierarchy of location servers which are connected to themselves and to the base stations (or mobile support stations) by a static network. User profiles are used to create partitions. It is only when the user crosses partitions that the update takes place. However, in most cases, user profiles are not always available a priori.

A modified tree structure for location management was proposed in [6]. The root and the some of the higher levels of the tree was replaced by a *set-ary* butterfly network. This helped in balancing the search requests at the nodes. Also proposed were schemes to make the protocol self-stabilizing.

The idea of location management with *home location servers* (HLS) has also been proposed in [12, 13]. Caching [12] and forwarding [13] have been independently proposed for Personal Communication Services (PCS). Similar to our work in this report, these strategies augment the basic location strategy which uses HLS. Section 9 compares them with the work presented in this report. Other techniques like user profile replication [23] and local anchoring [10] have also been proposed to reduce network load due to location management. In the next section we will present the network architecture typically used in centralized location management schemes.

3 Network Architecture

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

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

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

4 Location Management

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

4.1 Overview of the Basic IS-41 Scheme

Location management with home location servers is being used in current personal communication systems (PCS) standards proposals such as IS-41 [21].

4.1.1 Updates in the Basic IS-41 Scheme

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

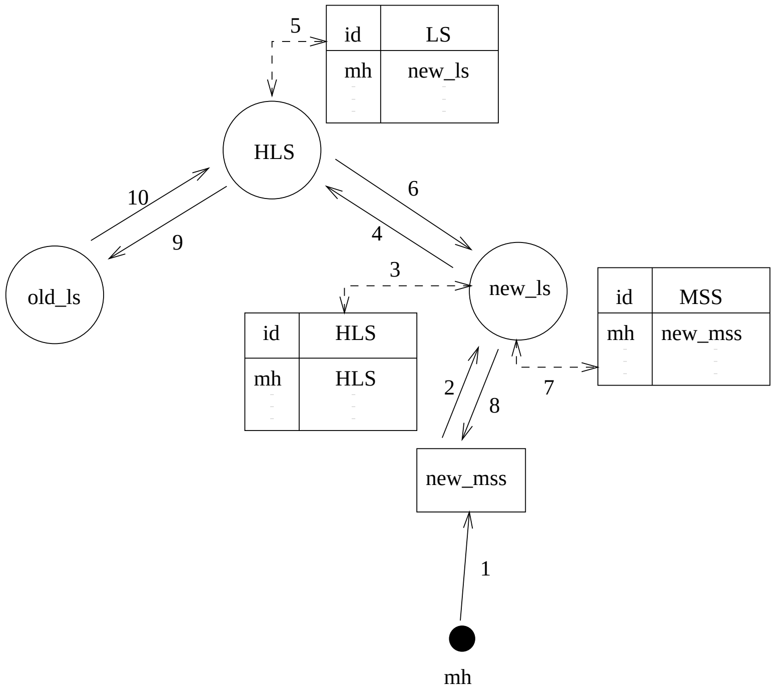

Figure 2: Update in the Basic Scheme (IS-41)

new_mss (steps 7-8). The *HLS* also sends a message to the location server of registration area *old*, named *old_ls*, requesting it to delete any host information stored at *old_ls* (step 9). The location information of the mobile host is then purged from *old_ls*, after that, *old_ls* sends a confirmation message to *HLS* (step 10).

4.1.2 Searches in the Basic IS-41 Scheme

Let us suppose that a mobile host *src* wants to communicate with another mobile host *dest*. Let *src_mss* be the *MSS* of the cell in which host *src* currently resides. When *src* wants to communicate with a mobile host *dest*, it has to first determine the location of *dest*. Host *src* sends a location query message to the mobile support station for its cell, *src_mss* (step 1 in Figure 3). The *src_mss*, forwards the query to the location server (say *src_ls*) in its registration area (step 2 in Figure 3). The location server *src_ls* checks whether *dest* is in its registration area[2]. If *dest* is in its registration area, then it returns the location information.

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

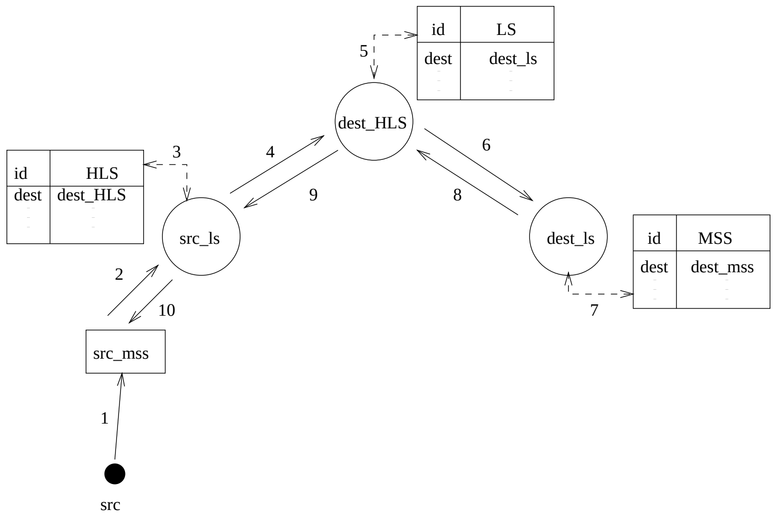

Figure 3: Search in the Basic Scheme (IS-41)

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

4.1.3 Drawbacks

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

- Inefficient location management: Updates on each registration area crossing is useful to reduce the search costs (i.e., *call set-up* time), only if the user is being called frequently.

However, if the user is not being called frequently, updates on each crossing are not necessary. In such scenarios, updates on each crossing lead to inefficient location management.

4.2 Proposed Scheme

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

We classify location *updates* further into two types, namely, *updates* and *search-updates*. A *update* occurs when a mobile host changes location. A *search-update* (also known as caching [12]) occurs after a successful *search*, when some hosts update the location information corresponding to the searched mobile host. We discuss more about search-updates in Section 5.

4.2.1 Forwarding Pointers

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

The forwarding pointers for a mobile host can be interpreted as forming a directed graph with the location servers as vertices and with an edge from location server ls_1 to location server ls_2 if and only if ls_1 has a forwarding pointer that points to ls_2. The resulting graph will be a tree with edges directed towards the root, the current location of the mobile host [8].

Example 4.1: For an easier understanding we will illustrate with an example. Let the notation $move$(a,b) represent a move of the mobile host from registration area a to registration area b. Let a mobile host make the following moves: $move$(1,2), $move$(2,3), $move$(3,2), *move*(2,4), and *move*(4,5). The resulting graph due to the forwarding pointers is shown in Figure 4(a). Now, the mobile host makes the following move, *move*(5,2). The mutation of the graph due to *move*(5,2) is shown in Figure 4(b).

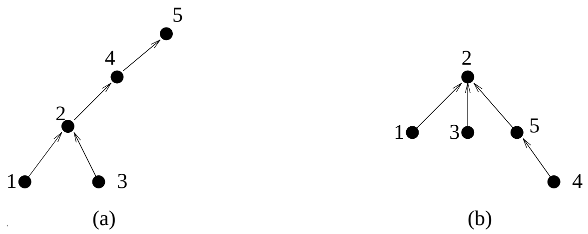

Figure 4: An Example of Forwarding Tree

The following obervations can be made [8]:

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

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

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

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

4.2.2 Updates Using Forwarding Pointers

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

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

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

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

The procedure to update the HLS in the proposed scheme is same as the update procedure in Section 4.1.1. After such an update, the HLS will know the present location of the mobile host, and no forwarding pointers need to be traversed (until the mobile host moves from its present registration area).

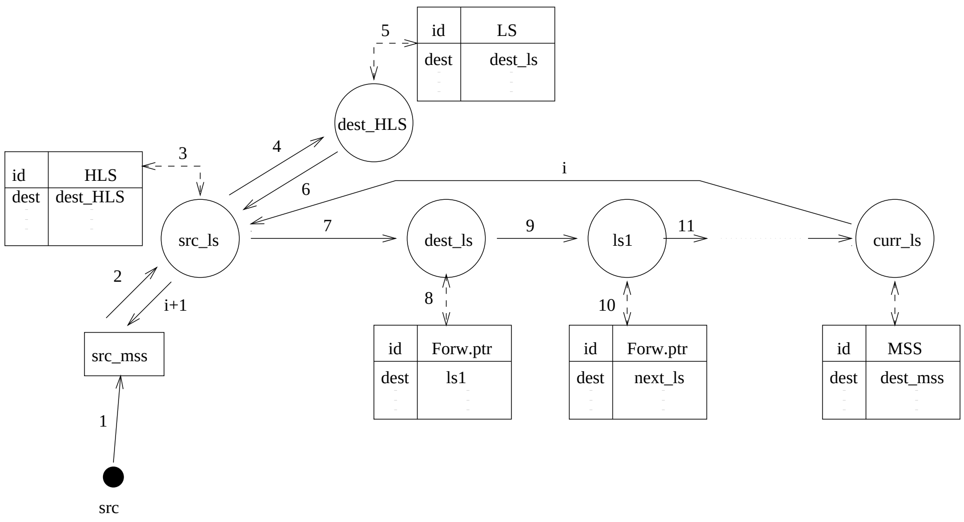

Figure 5: Search using HLS and Forwarding Pointers

4.2.3 Location Searches Using Forwarding Pointers

Let us suppose that a mobile host *src* wants to communicate with another mobile host *dest*. Please refer Figure 5 for this discussion. The first four steps of the search procedure are identical to steps 1-4 in Figure 3, until the location query is forwarded to the home location server of $dest$, $dest\_HLS$. $Dest\_HLS$ returns the address of the location server ($dest\_ls$) of the registration area in which the mobile host was residing when the last update at $dest\_HLS$ took place (steps 5 and 6). Upon receiving the address of $dest\_ls$, src_ls sends a message query that traverses the chain of forwarding pointers originating at $dest\_ls$ (step 7). Each location server on the chain looks up its forwarding pointer database to determine the next location server on the chain, and forwards the location query to it (e.g., steps 8 and 9 for $dest\_ls$). This continues till the end of the chain is reached (say $curr\_ls$ is the location server). The host $dest$ will be located in a cell in the registration area of $curr\_ls$. The location server $curr\_ls$ returns the current cell location to src_ls, which in turn forwards it to src_mss (steps i and $i+1$ in Figure 5). Thereafter, the call is set up between src and $dest$ via the src_mss and $dest\_mss$.

4.2.4 Forwarding Pointer Maintenance

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

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

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

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

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

4.3 Performance Analysis

In this section we analyze the proposed location management scheme using forwarding pointers and compare it with the basic scheme (IS-41). We will then compare the performance of the two heuristics presented in this report.

Calls between two hosts within the same registration area will not involve the home location servers. Moves within the registration area will also not involve any home location server interaction. Therefore, we analyze the performance of the location management schemes for calls which arrive from outside the registration area of a host (hereafter, we

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

refer them simply as call arrivals), and for moves which are registration area crossings. We define "cost" of sending a message as the amount of network usage due to sending the message. In other words, it is the amount of time the network is busy trasmitting the message.

The following terminology is used in the analysis:

- r = average number of calls per registration area crossing. We call this the call-mobility ratio.
- C_u = cost of an update per registration area crossing using IS-41. We will derive an expression for C_u later.
- C_s = cost of a location search using IS-41. We will derive an expression for C_s below.
- C'_u, C'_s are the update and search cost, respectively, using forwarding pointers.
- $\text{cost}(x \rightarrow y)$ = cost of sending a message from x to y.
- A = average cost of sending a location query/reply message between MSS and the LS within a registration area.
- B = average cost of sending a location query/reply message between a location server and a home location server. This is thus the average cost of a HLS interaction.
- k = For a caller host, this is the number of forwarding pointers traversed before locating the destination host.
- F = cost of *setting* a forwarding pointer between two location servers. This cost includes the cost of sending a message between the two location servers. The cost of *traversing* a forwarding pointer also includes the cost of sending a message between two location servers. The cost of traversing a forwarding pointer is also assumed to be F.
- α = Relative forwarding cost = $\frac{F}{C_u}$.

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

4.3.1 Performance Metrics

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

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

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

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

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

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

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

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

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

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

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

4.3.2 IS-41 Scheme

Let a mobile host mh move from a registration area to a new registration area, and, let the corresponding location servers be old_ls and new_ls respectively. Let the new mobile support station be new_mss. Let the home location server of mh be HLS. From the update scheme presented in Section 4.1.1, and Figure 2 we can obtain,

$$\begin{aligned} C_u &= \text{cost}(new\_mss \rightarrow new\_ls) + \text{cost}(new\_ls \rightarrow HLS) + \text{cost}(HLS \rightarrow new\_ls) \\ &\quad + \text{cost}(new\_ls \rightarrow new\_mss) + \text{cost}(HLS \rightarrow old\_ls) + \text{cost}(old\_ls \rightarrow HLS) \\ &= A + B + B + A + B + B \\ &= 2A + 4B \end{aligned}$$

Step 1 involves message transfer over the wireless link, and hence is not considered in the cost analysis. Steps 3 and 5 in Figure 2 are database lookup and update operations, and do not incur any network load. Thus, they too are not considered in the cost analysis.

Let a mobile host src call another mobile host $dest$. Let the location servers of the registration areas in which src and $dest$ are currently residing be src_ls and $dest\_ls$ respectively. Let the home location server of $dest$ be $dest\_HLS$. From Section 4.1.2 and Figure 3, we can obtain,

$$\begin{aligned} C_s &= \text{cost}(src\_mss \rightarrow src\_ls) + \text{cost}(src\_ls \rightarrow dest\_HLS) \\ &\quad + \text{cost}(dest\_HLS \rightarrow dest\_ls) + \text{cost}(dest\_ls \rightarrow dest\_HLS) \\ &\quad + \text{cost}(HLS \rightarrow src\_ls) + \text{cost}(src\_ls \rightarrow src\_mss) \\ &= A + B + B + B + B + A \\ &= 2A + 4B \qquad (1) \end{aligned}$$

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

4.3.3 Movement-based Heuristic

As stated earlier, the number of registration areas crossed by a host when the movement-based heuristic scheme performs an update for that host is $\mathcal{M}$. Thus, the number of forwarding pointers due to moves by the host when an update takes place for that host is

$(\mathcal{M}-1)$. Since the number of forwarding pointers for a host when an update takes place is $(\mathcal{M}-1)$, $(\mathcal{M}-1)F$ is the total cost of creating forwarding pointers before an update takes place for that host. Therefore the update cost per registration area crossing for the movement-based heuristic scheme is,

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

From Figure 5 we can obtain the search cost for the movement-based scheme as follows:

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

Step 1 in Figure 5 involves message transfer over the wireless link, and hence is not considered in the cost analysis.

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

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

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

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

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

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

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

The relative total cost is given as follows:

$$\frac{C'_t}{C_t} = \frac{\frac{1}{r}C'_u + rC'_s}{\frac{1}{r}C_u + rC_s}$$

$$= \frac{\frac{1}{r\mathcal{M}}(C_u + (\mathcal{M}-1)F) + r(A + \frac{1}{2}(C_s + 2(\mathcal{M}-1)F))}{\frac{1}{r}C_u + rC_s} \tag{3}$$

We assume that the cost B of HLS interactions is the dominant cost in C_s and C_u. This may not be true for all network architectures. This assumption was mainly motivated by the studies which indicate HLS to be the bottleneck in PCNs. The forwarding cost F is assumed to be a fraction of the cost of update and searches, i.e., $F = \alpha C_u$, where, $\alpha \leq 1$. Substituting $F = \alpha C_u$, $A \ll C_u$, and $C_s = C_u$ in (3) yields,

$$\frac{C'_t}{C_t} = \frac{1}{2\mathcal{M}} \left(\frac{(1 + \mathcal{M}r^2)(1 + 2\alpha(\mathcal{M}-1)) + 1}{1 + r^2} \right) \tag{4}$$

The relative search cost is given as follows:

$$\frac{C'_s}{C_s} = \frac{1}{2}(1 + 2\alpha(\mathcal{M}-1)) \tag{5}$$

Using the above equations we will now determine the values of $\mathcal{M}$ that satisfies the conditions stated in Section 4.3.1. Condition 2 requires the search cost of the proposed scheme to be less than or equal to twice than the search cost of the IS-41 scheme. For this condition to be satisified, following should hold:

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

Figure 6 illustrates the variation of maximum allowable $\mathcal{M}$ with α. Note, that for low α values, $\mathcal{M} > 5$ is allowable. Figs. 7-9 illustrate the variation of relative total cost with r for different values of $\mathcal{M}$, for $\alpha = 0.2, 0.5, 0.8$ respectively. It can be observed that at high values of r, Condition 1 is violated for high values of $\mathcal{M}$. In other words, long chain lengths are not suitable at high values of r. Thus, a small $\mathcal{M}$ has to be chosen in order to obtain good performance for a wide range of r and α. We keep $\mathcal{M} < 5$ in our analysis. We analyze the performance of movement-based heuristic scheme for $\mathcal{M} = 2$, 3 and 4. Note that in Figs. 7-9, the relative total cost of the movement-based scheme increases as r and α increases. Longer chain lengths (i.e., larger $\mathcal{M}$) are preferred at low values of r. However, as r increases, schemes using smaller chain lengths (smaller $\mathcal{M}$) perform better. It can be seen in Figure 8 and Figure 9 that for high α values, the performance of movement-based heuristic with large $\mathcal{M}$ performs worse than the IS-41 scheme at high values of r.

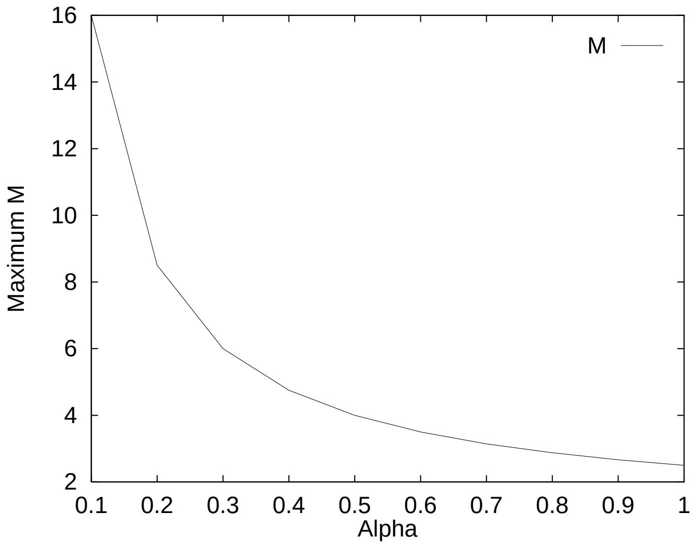

Figure 6: Maximum Allowable Chain Length

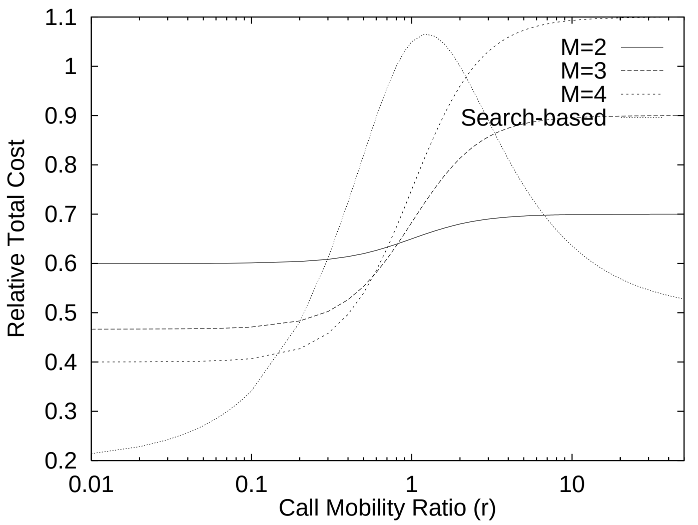

Figure 7: Relative Total Costs: $\alpha = 0.2$

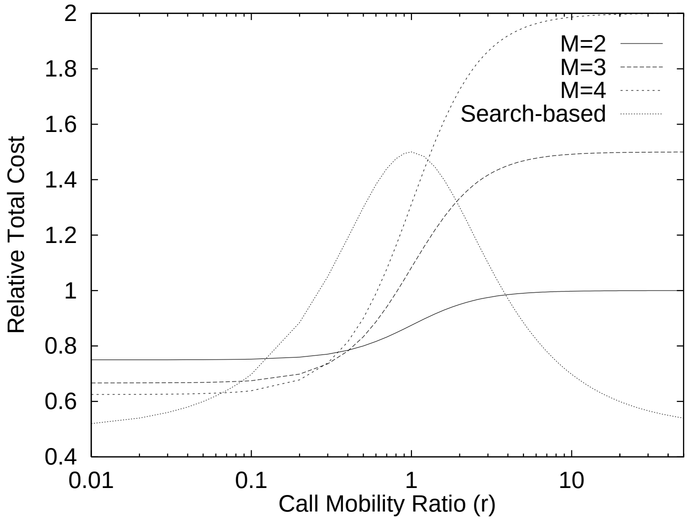

Figure 8: Relative Total Costs: $\alpha = 0.5$

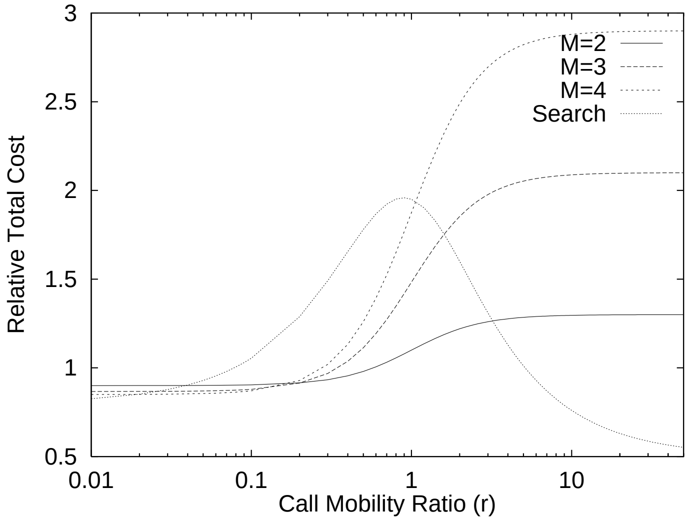

Figure 9: Relative Total Costs: $\alpha = 0.8$

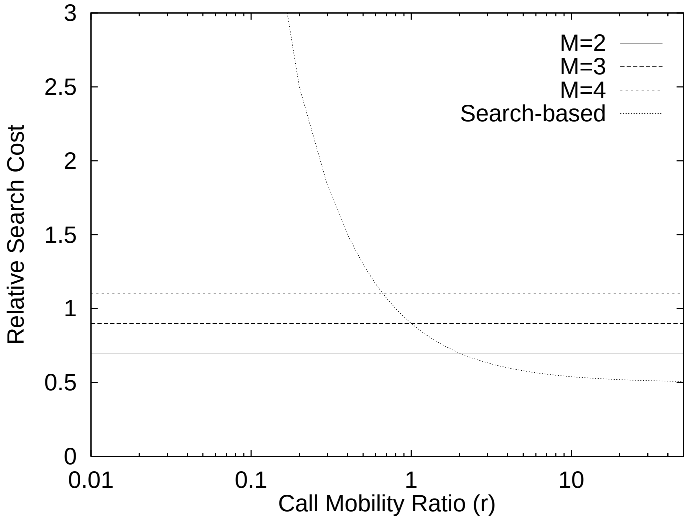

Figure 10: Relative Search Costs: $\alpha = 0.2$

Figs. 10-12 illustrate the variation of the relative search cost of the movement-based heuristic scheme with r for $\alpha = 0.2, 0.5, 0.8$ respectively. It can be observed that the relative search cost of the movement-based heuristic scheme is independent of r. However, the relative search cost increases with α. At high values of α, Condition 2 is violated for large values $\mathcal{M}$.

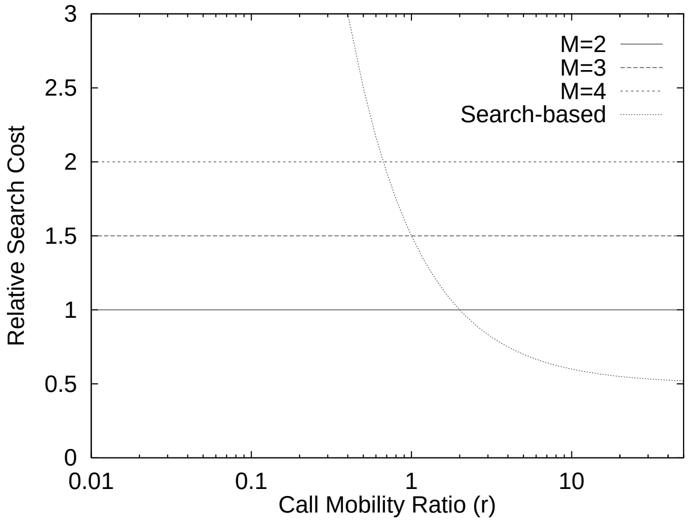

Figure 11: Relative Search Costs: $\alpha = 0.5$

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

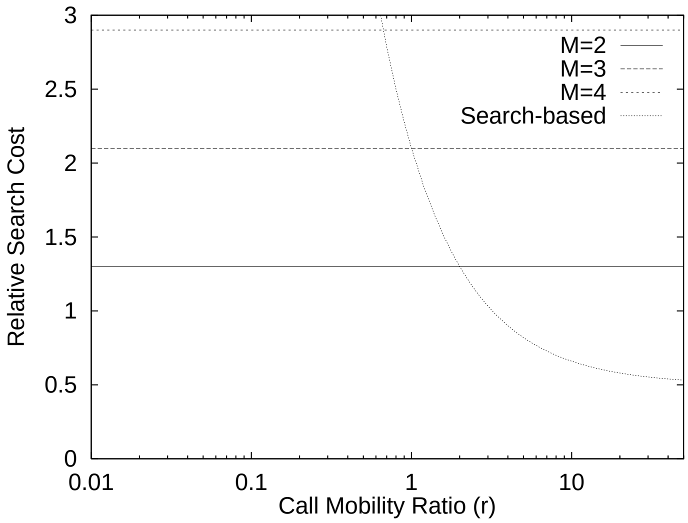

Figure 12: Relative Search Costs: $\alpha = 0.8$

for high values of α and r.

4.3.4 Search-based Heuristic

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

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

The update cost for search-based heuristic scheme is,

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

Thus, the total cost is,

$$C_t' = \frac{1}{r} C_u' + r C_s' = \frac{1}{r}(F + \frac{r}{\mathcal{S}} C_u) + r(A + \frac{C_s}{2} + 2(\frac{\mathcal{S}+1}{2r})F) \quad (6)$$

After substitutions similar to that in Section 4.3.3, the relative total cost and relative search cost for the search-based heuristic are as follows:

$$\frac{C_t'}{C_t} = \frac{2\mathcal{S}\alpha(1 + r(\mathcal{S}+1)) + r(2 + \mathcal{S}r)}{2\mathcal{S}(1 + r^2)} \quad (7)$$

$$\frac{C'_s}{C_s} = \frac{1}{2r}(r + 2(\mathcal{S} + 1)\alpha) \tag{8}$$

Using the above equations, we will now determine a 'good' value of $\mathcal{S}$ that satisfies the conditions stated in Section 4.3.1. For Condition 2 to hold, the following should hold:

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

Figure 13 illustrates the variation of maximum allowable $\mathcal{S}$ with r for $\alpha = 0.2$. Note, that for low values of r, the maximum allowable $\mathcal{S}$ is less than 1. This suggests that HLS updates should take place more than once between two consecutive searches. The maximum allowable $\mathcal{S}$ increases with r. For our analysis, we fix $\mathcal{S} = 1$. This is because, we observed that the performance of search-based heuristic scheme with $\mathcal{S} = 1$ is quite close to that with 'optimal' $\mathcal{S}$, in terms of relative total cost, where the optimal $\mathcal{S}$ is the value of $\mathcal{S}$ that provides the minimum relative total cost.

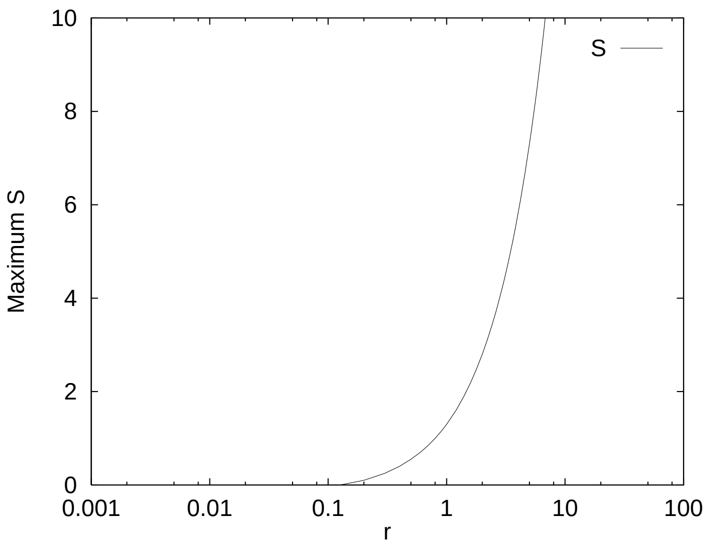

Figure 13: Maximum Allowable Searches per HLS Update: $\alpha = 0.2$

The variation of the relative total cost with r for the search-based heuristic scheme is illustrated in Figs. 7-9. It can be observed that the relative total cost is very low for low values and r and high values of r. At low values of r, HLS updates occur very rarely (i.e., only during searches). Since the update cost is the dominating factor in the total cost at low values of r, the relative total cost of the search-based update heuristic is very low. On the other hand, HLS updates occur very often at high values of r. This causes the search cost to be very low. Since the search cost is the dominating factor in the total cost at high values of r, the relative total cost of the search-based heuristic is very low. However at moderate values of r, the performance of search-based heuristic scheme is worse than the IS-41 scheme.

Figs. 10-12 illustrate the variation of relative search costs with r for $\alpha = 0.2, 0.5, 0.8$ respectively. It can be observed that the search cost is very high for low values of r. This is because for small r, the HLS updates occur infrequently, and the moves frequently. Thus, each search has to traverse a long chain of forwarding pointers. Moreover, there is a rapid decrease in search costs as r increases. This is because the length of chain of forwarding pointers reduces as calls arrive more often.

4.3.5 Observations

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

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

5 A Search-Update Strategy

The search cost could be reduced by eliminating the cost of querying the HLS for the destination host. This can be achieved by having a location entry at the src_ls (in Figure 5) pointing to the start of the chain of forwarding pointers. This is called a "search-update". A search-update occurs after a successful *search*, when the location server of the requesting host updates the location information corresponding to the mobile host. Search-updates might reduce the search cost (call set-up time). In this work, we will use a particular search-update strategy called *jump update*.

5.1 Jump update

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

5.2 Altered Search Procedure for Search-Updates

The search procedure needs to be altered to make use of the location information (if available) at the location server. If $dest$ is not in the registration area of src_ls, check if src_ls has a location entry for $dest$. The location entry for $dest$ at src_ls might be either the current location of $dest$, or the start of a chain of forwarding pointers to the current location of $dest$. Thus, when the location entry can be found in src_ls, a HLS query could be avoided. The chain of forwarding pointers is traversed to get to the current location of $dest$. Else, if the location entry for $dest$ is not found in src_ls, follow the previously explained search procedure (Section 4.2.3).

5.3 Forwarding Pointer Maintenance for Search-Updates

In the forwarding pointer maintenance procedure described in Section 4.2.4, we were not concerned about the stale forwarding pointers at the location servers, because all the searches (Section 4.2.3) went to the HLS first. We ensure that the HLS does not have stale location information by updating the HLS at least once every T_{purge} units of time. The location servers also purge forwarding pointers that are older than T_{purge} units of time, thus, ensuring that stale forwarding pointers are not encountered during a search. However, if the search procedure of a host is altered as in Section 5.2 it should be ensured that the forwarding pointers at the location server are not stale at any point of time.

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

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

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

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

location servers which have a forwarding pointer for a host due to moves before an HLS update takes place for the host. If search-updates are used, then each call for the host from a location server will lead to a forwarding pointer for the host at the location server. The average number of calls that can occur before an HLS update is $r\mathcal{M}$. Therefore, the average number of location servers that have forwarding pointers created due to search-updates is upper bounded by $r\mathcal{M}$. This is because there can be more than one call from the area served by a location server. Therefore, for the movement-based heuristic, the average number of purge messages sent during an HLS update is upper bounded by $((\mathcal{M} - 1) + r\mathcal{M})$.

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

Disconnection: Disconnections can also cause inconsistency in location information. If a host disconnects, its location information should be purged from all location servers. The HLS of the host will know the last location of the host before it disconnects (Appendix A). The HLS will also know the location of the host whenever it reconnects/switches ON (Appendix A). However, if the host switches ON at a location different from the location where it had earlier disconnected, the location servers which had forwarding pointers to the old location, have stale and incorrect information. Thus, if the altered search procedure (Section 5.2) is used, we need to purge the stale and incorrect location information at these location servers. This will require purge messages to be sent to the location servers which have the forwarding pointers for the disconnected host. Subsequent search for the host initiated from these location servers will be forwarded to the HLS of the host. The HLS will not have incorrect location information of the host. Any search query for a disconnected host initiated from these location servers while the purge is in progress will lead to a location server ls_i which has no information of the host. The location server ls_i sends back an error message to the location server which initiated the search. Upon receiving an error message, the location server will purge its forwarding pointer, and forward the search query to the

HLS of the host.

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

5.4 Performance Analysis of Search-update Scheme

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

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

5.4.1 Movement-based Heuristic and Search-updates

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

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

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

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

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

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

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

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

$$\frac{C''_t}{C_t} = \frac{C'_t}{C_t} - \frac{r}{1+r^2}\left(\frac{rs}{2} + r\alpha((\mathcal{M}-1) - 2k') - \frac{\alpha((r+1)\mathcal{M} - 1)}{r\mathcal{M}} - r\alpha)\right) \tag{9}$$

$$\frac{C''_s}{C_s} = \frac{C'_s}{C_s} - \left(\frac{s}{2} + \alpha(\mathcal{M} - 1 - 2k')\right) \tag{10}$$

For movement-based heuristic scheme with search-updates to be beneficial, we require $\frac{C''_t}{C_t} \leq \frac{C'_t}{C_t}$ in (9). The value of s for movement-based heuristic scheme with search-updates to be beneficial is given as,

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

We determine the value of s and k' for different values of $\mathcal{M}$ using the markov state analysis presented in Appendix B. Using the s and k' value obtained from Appendix, we determine the relative total cost (in (9)) and the relative search cost (in (10)). Figs. 14-16 illustrate the variation of relative total cost with r for $\alpha = 0.2, 0.5, 0.8$ respectively. Similarly, Figs. 17-19 illustrate the variation of relative search cost with r for $\alpha = 0.2, 0.5, 0.8$ respectively. It can be observed that the relative search cost has considerably reduced as compared to the schemes without search-updates. Moreover, the relative search cost is less than $\mathcal{R}$ (equal to 2) for a wide range of α and r. However, for $r < 1.0$, the relative total cost is greater than the relative total cost for movement-based heuristic without search-updates. This is due to the additional purging costs.

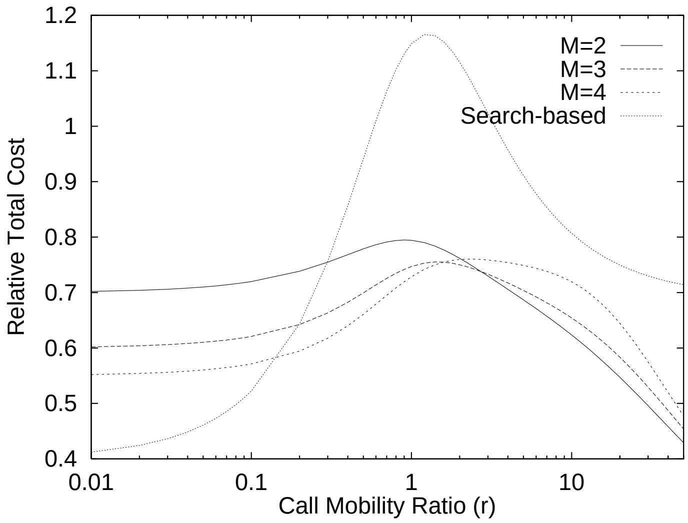

Figure 14: Relative Total Costs (with Search Updates): $\alpha = 0.2$

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

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

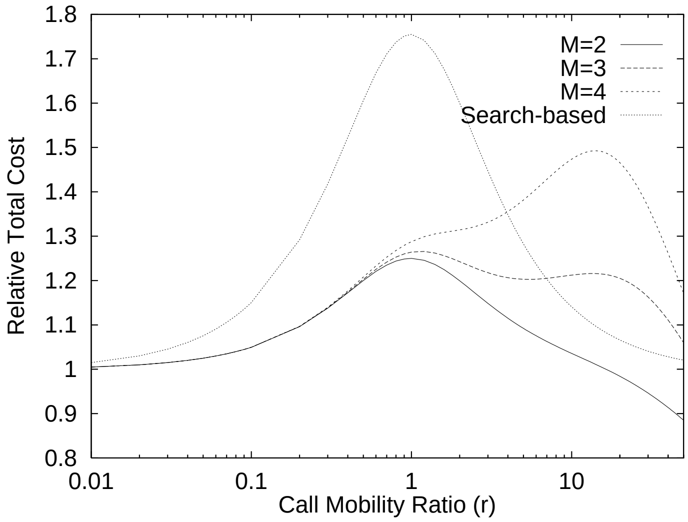

Figure 15: Relative Total Costs (with Search Updates): $\alpha = 0.5$

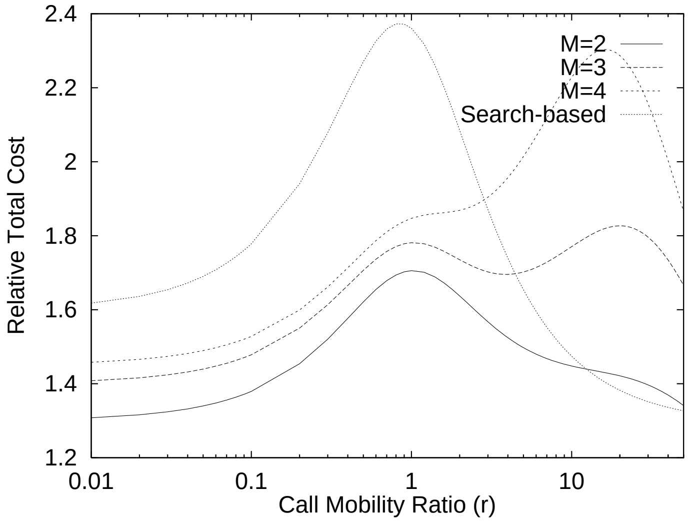

Figure 16: Relative Total Costs (with Search Updates): $\alpha = 0.8$

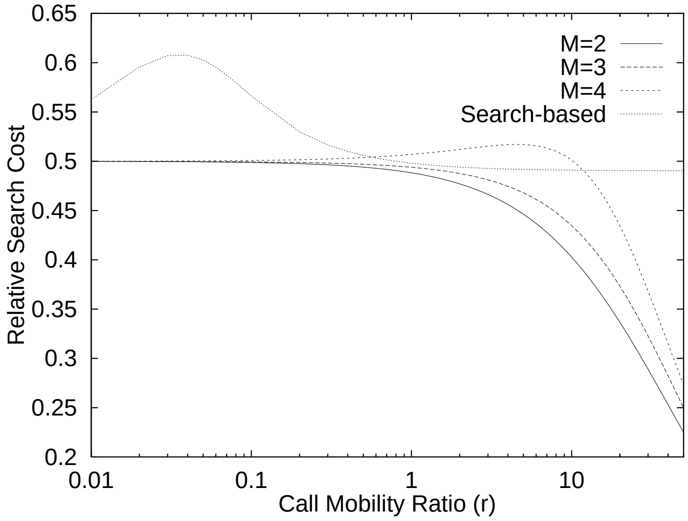

Figure 17: Relative Search Costs (with Search Updates): $\alpha = 0.2$

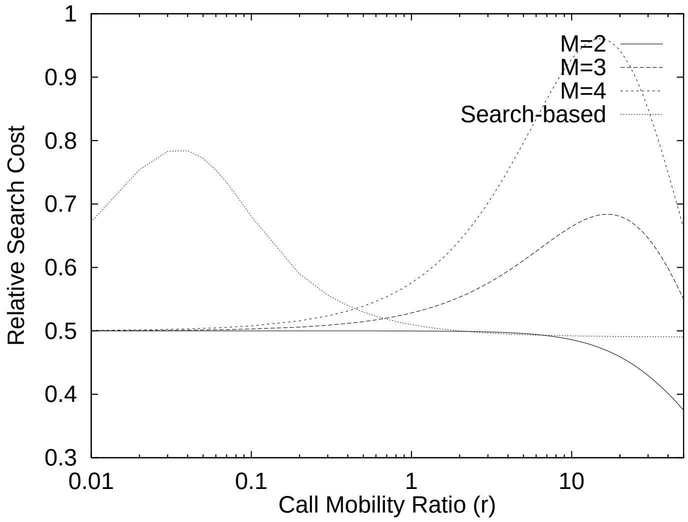

Figure 18: Relative Search Costs (with Search Updates): $\alpha = 0.5$

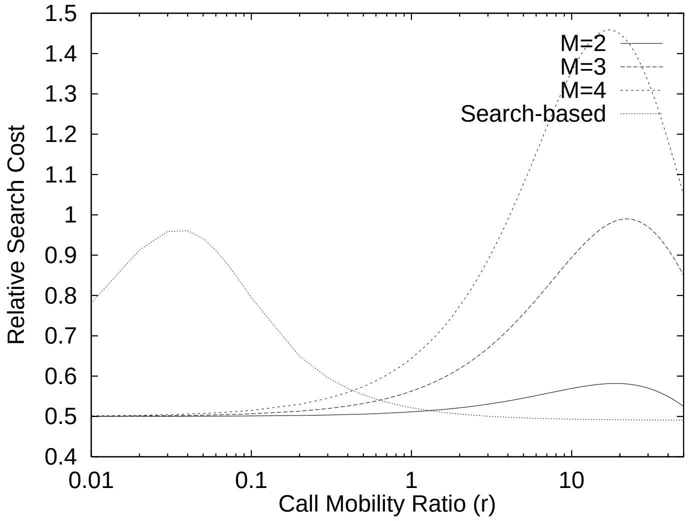

Figure 19: Relative Search Costs (with Search Updates): $\alpha = 0.8$

5.4.2 Search-based Heuristic and Search-updates

As stated earlier in Section 4.3.4, we fix $\mathcal{S} = 1$. Similar to the movement-based heuristic, the update cost will also include the purge costs incurred due to forwarding pointer maintenance. As determined earlier in Section 5.3, the upper bound on average purge cost for the search-based heuristic when search-updates are used is $(1 + \frac{1}{r})F$. Therefore,

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

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

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

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

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

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

$$\frac{C''_t}{C_t} = \frac{C'_t}{C_t} + \frac{r}{1+r^2}\left(\frac{sr}{2} - \frac{\alpha(r+1)}{r} + 2r\alpha(\frac{1}{r} - k') - r\alpha\right) \tag{11}$$

$$\frac{C''_s}{C_s} = \frac{C'_s}{C_s} - \left(\frac{s}{2} + 2\alpha(\frac{1}{r} - k')\right) \tag{12}$$

For search-based heuristic scheme with search-updates to be beneficial, we require $\frac{C''_t}{C_t} \leq \frac{C'_t}{C_t}$ in (11). The value of s for search-based heuristic scheme with search-updates to be beneficial is given as:

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

We determine the value of s and k' using the markov state analysis presented in Appendix C. Using the s and k' value obtained from Appendix, we determine the relative total cost (in (11)) and the relative search cost (in (12)). As seen in Figs. 17-19, the relative search cost of the search-based heuristic scheme has considerably reduced upon using search-updates. This has happened at the expense of a small increase in the relative total cost (Figs. 14-16). It should however be noted that at high α values, search-updates are not suitable for search-based heuristic scheme. This is because of the high purging cost involved.

5.4.3 Observations for Search-updates

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

Figure 20 presents a summary of the results. This figure has been repeated from Section 1 for convenience. As stated earlier, we divide the r space into three regions: *low* ($r < 0.1$), *moderate* ($0.1 \leq r < 5$) and *high* ($r \geq 5$). We divide the α space into two regions: *low* ($\alpha < 0.5$), and *high* ($alpha \geq 0.5$).

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

<table>
<tr><th>α
r</th><th>Low</th><th>High</th></tr>
<tr><td>Low</td><td>Search-based
with
Search-Updates</td><td>Movement-based
(M<5)</td></tr>
<tr><td>Moderate</td><td rowspan="2">Movement-based
with
Search-Updates
(M<5)</td><td>Movement-based
(M<3)</td></tr>
<tr><td>High</td><td>Search-based</td></tr>
</table>

Figure 20: Performance Chart

6 Fault Tolerance

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

Fault tolerance requires that regardless of the initial state, the system will eventually converge to the correct state [5, 6]. The correct state in the scope of location management is that the location servers have the correct location information of the hosts. To recover from forwarding pointer corruption due to a transient failure, the location servers have to periodically send the pointer information to other location servers reachable using the bi-directional pointers. An inconsistency can occur if a location server ls has a pointer for a host h to another location server ls_1, and the location information of h at ls_1 is corrupted by a transient failure. Thus, periodic exchange of information along the bi-directional pointers will ensure consistent forwarding pointer information to be stored in the location servers. Apart from the forwarding pointers, the host database (which stores the cell location of the host in its registration area) at the location server should also be fault tolerant. This will require periodic exchange of information between the MSSs in the registration area and the location server. An inconsistency can occur if according to the location server database, a host is located in more than one cell, or not located in any cell although the host is present in some MSS's cell in the registration area.

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

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

7 Memory Overhead

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

- Forwarding Pointers: A forwarding pointer for a host is created at a location server when the host goes out of the location server's registration area.

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

We use the following notations and data for our analysis. The data is obtained from [21]. Let:

- U = Total number of users in the network = 2.87 million.
- N = number of registration areas in the network = 128.
- ρ = mean density of the users = 390/sq.km.
- L = Registration area boundary length = 30.3 km.
- v = average speed of the user = 5.6 km/hr.
- λ = Rate of call origination and delivery = 1.4/hr/user.
- B_1 = Size of a forwarding pointer in bytes.
- B_2 = Size of a bi-directional pointer in bytes.
- t_p = purge time interval for the pointers.

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

The rate of registration area crossing R is given as $\frac{\rho v L}{\pi}$ [24]. Replacing the values, we obtain $R = 5.85/s$. Let the maximum number of pointers that exist at a location server at any time be P. Let us consider forwarding pointers first. Since pointers are created only due to moves, $P = R * t_p$. The memory required for forwarding pointers is $M_1 = P * B_1 = 5.85 * 3600 * 16 \approx 337 Kbytes$.

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

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

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

8 Estimation of Call-Mobility Ratio r

As stated earlier, the performance of the proposed scheme depends on the call-mobility ratio (r) and the relative cost of forwarding pointers (α). High network load is encountered if forwarding strategies are applied in networks where α is high and users have low r. The value of α is network dependent and should be known to the system designer. However, user's call-mobility ratio is not known a priori to the system designer. Thus, it is important to estimate the call-mobility ratio of the users so that the proper forwarding strategy could be applied. Various techniques have been proposed in literature to estimate r. A detailed investigation of the algorithms for estimating r, namely, *running average*, and *reset-K* were presented in [12]. In our earlier work [16], we propose algorithms to estimate r based on the history of call and mobility patterns. The basic assumption behind these algorithms are that past history of the user's call mobility patterns will reflect the behavior in future.

9 Comparison with Other Centralized Schemes

Our work presented in this research [17] differs from other centralized schemes [12, 13] in the following respect:

- A new heuristic (search-based) is proposed in this work. In [13] only movement-based heuristic is discussed.
- This work studies the impact of combining forwarding and caching on the perfomance. We show that caching in addition to forwarding does improve performance over schemes that employ only forwarding.
- However, combining caching and forwarding brings about unique problems with forwarding pointer maintenance. In this report we propose schemes for forwarding pointer maintenance. We also account for the overhead incurred due to maintenance in our analysis. Forwarding pointer maintenance and its associated cost was not taken into account in [13].
- We propose schemes to make location management strategy robust in the presence of failures.
- To assist in maintenance and fault-tolerance, we employ bi-directional pointers instead of uni-directional pointers [13].

10 Summary

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

The main assumptions made in this work are: (1) HLS interaction cost is the dominant cost, and (2) the length of the chain is a non-decreasing function of the moves. Assumption (1) may not be true for all network architectures. This assumption is mainly motivated by the studies which indicate HLS to be the bottleneck in PCNs [12, 13, 21]. On the other hand, assumption (2) helps in determining the lower bounds for the performance of our schemes.

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

References

[1] B. Awerbuch, D. Peleg, "Concurrent online tracking of mobile users," *Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols,* October 1991.

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

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

[4] Amotz Bar-Noy, H. Kessler, "Tracking Mobile Users in Wireless Communication Networks," *Proc. IEEE, INFOCOM*, 1993.

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

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

[7] G.H.Forman, J. Zahorjan, "The Challenges of Mobile Computing," *IEEE Computer,* Apr.1994.

[8] R. J. Fowler, "The Complexity of using Forwarding Address for Decentralized Object Finding," *Proc. ACM Symposium on Principles of Distributed Computing,* 1986.

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

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

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

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

[13] R. Jain and Y. B. Lin, "A Forwarding Strategy to Reduce Network Impacts of PCS," *Proc. of IEEE INFOCOM,* Mar., 1995.

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

[15] P. Krishna, N. H. Vaidya and D. K. Pradhan, "Location Management in Distributed Mobile Environments," *Proc. of the IEEE Intl. Conf. on Par. and Dist. Information Systems,* pp. 81-89, Sep. 1994.

[16] P. Krishna, N. H. Vaidya and D. K. Pradhan, "Static and Adaptive Location Management in Distributed Mobile Environment," *Computer Communications* (Special Issue on Mobile Computing), Vol. 19, No. 4, March, 1996.

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

[18] P. Krishna, N. H. Vaidya and D. K. Pradhan, "Efficient Location Management in Mobile Wireless Networks," Technical Report # 95-011, Dept. of Computer Science, Texas A&M University, March, 1995.

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

[20] Kathleen S. Meier-Hellstern, et. al., "The Use of SS7 and GSM to support high density personal communications," *Proc. of the Winlab workshop on third generation wireless information networks,* April 1992.

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

[22] M. Spreitzer and M. Theimer, "Providing Location Information in a Ubiquitous Computing Environment," Tech. Rept., Xerox PARC, 1993.

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

[24] R. Thomas, H. Gilbert, and G. Mazziotto, "Influence of the Mobile Station on the Performance of a Radio Mobile Cellular Network," *Proc. 3rd Nordic Seminar,* paper 9.4, Copenhagen, Denmark, Sep., 1988.

[25] S. F. Wu and Charles Perkins, "Caching Location Data in Mobile Networking," *IEEE Workshop on Advances in Parallel and Distributed Systems,* October 1993.

Appendix A: Network Events in Mobile Wireless Network

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

Switch ON

Each *MSS* periodically transmits a beacon identifying itself and the *registration area* it is located in. Each mobile host generates a *registration* message identifying itself when it switches on (step 1 in Figure 21) [3]. When a *MSS* receives a *registration* message from a mobile host, it forwards the message to its location server (step 2 in Figure 21)). The location server updates its database to indicate the current cell location of the mobile host (step 3). The location server determines the *HLS* of the mobile host from the mobile host identifier (step 4). It then informs the *HLS* of the current location of the mobile host (step 5). The *HLS* upon updating the database (step 6) sends back an acknowledgment to the location server (step 7) which in turn sends an acknowledgment to the *MSS* (step 8), thus, completing the registration process.

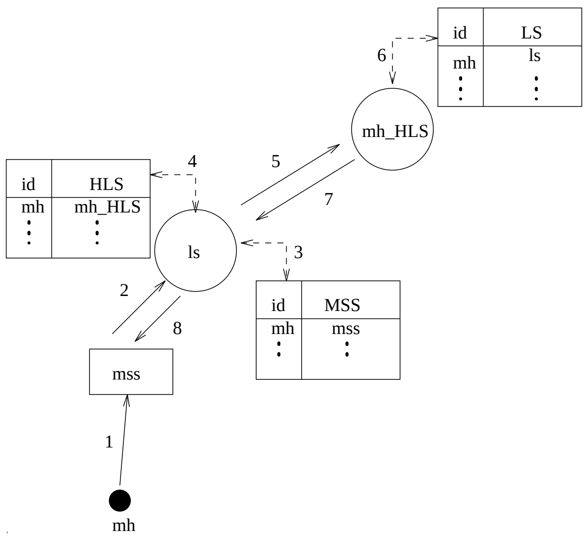

Figure 21: Mobile Host Switches ON

Switch OFF

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

Handoffs

The mobile host can be *idle* (not engaged in a call or data transfer) or *active* (engaged in a call or data transfer) when the host crosses cell boundary. The problem of active handoffs are beyond the scope of this work. Apart from location management, active handoffs involve connection management issues, e.g., connectivity, in-order delivery (in wireless ATM) etc. Here we will describe a procedure for idle handoffs [3]. Please refer to Figure 22 for this discussion. The mobile host mh upon detecting a stronger signal from $mss2$ initiates the

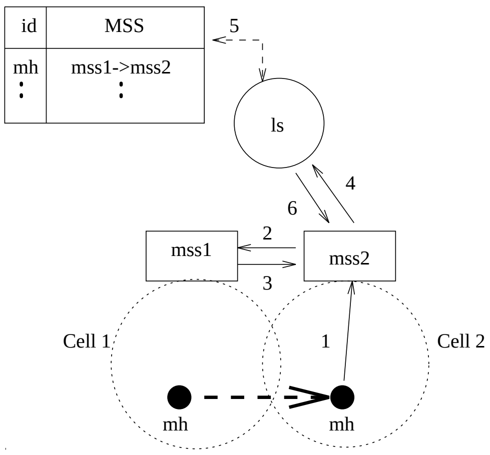

Figure 22: Handoff

handoff procedure by sending a *handoff-init* message to $mss2$ (step 1 in Figure 22). This message includes the identifiers of mh and the old MSS, i.e., $mss1$. The $mss2$ upon receiving the *handoff-init* message requests $mss1$ to transfer the host information (user profile, etc.) (steps 2 and 3). The $mss2$ then sends an update message to the location server (ls) of its registration area (step 4). The ls updates its database and sends back an acknowledgment (steps 5 and 6). The HLS database is not updated during handoffs.

Appendix B: Search-updates Analysis for Movement-based Heuristic

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

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

The state transition diagram based on above assumptions is as shown in Figure 23. We model the length of a chain for a host at a particular location server by a markov process. State i, where, $0 \leq i \leq K$ and $i \neq \psi$, represents the state where the length of the chain at the location server is i. The state ψ is the state when the location server has no forwarding pointer for the host, hence, a search for the host originating from the location server requires a HLS query.

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

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

Simplifying it we get,

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

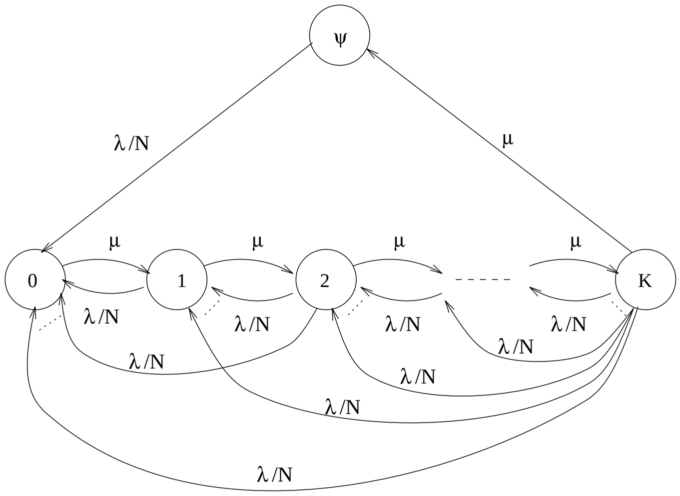

Figure 23: State Transitions when Search-Updates are used with Movement-based Heuristic

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

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

Now, the state transition equation for state ψ,

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

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

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

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

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

For $N = 50$, Figure 24 illustrates the variation of k' when r changes, and Figure 25 illustrates the variation of s with r. As shown in Figure 25, we see that s increases as r increases. This is becauses, at high values of r, calls are more frequent than moves, and since we are using *jump updates*, the probability of the location information being obtained from the location server itself is high.

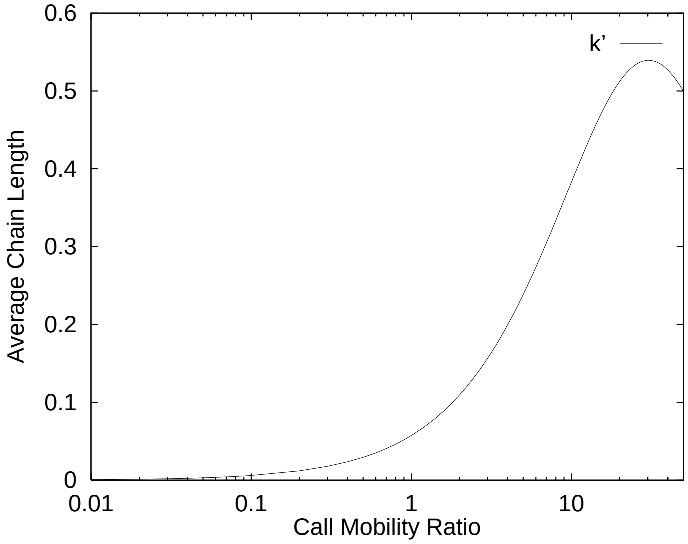

Figure 24: Variation of k' with r for $\mathcal{M} = 3$

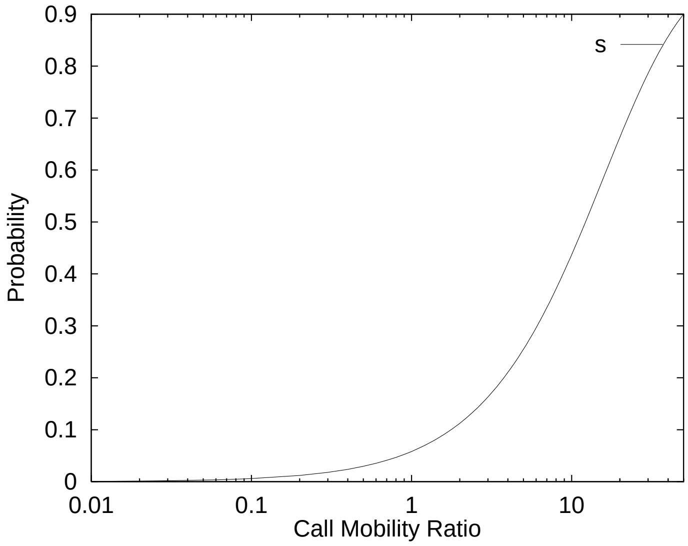

Figure 25: Variation of s with r for $\mathcal{M} = 3$

Appendix C: Search-updates Analysis for Search-based Heuristic

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

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

The state transition diagram based on above assumptions is as shown in Figure 26. The state equations for this state diagram are as follows:

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

For $i > 0$,

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

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

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

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

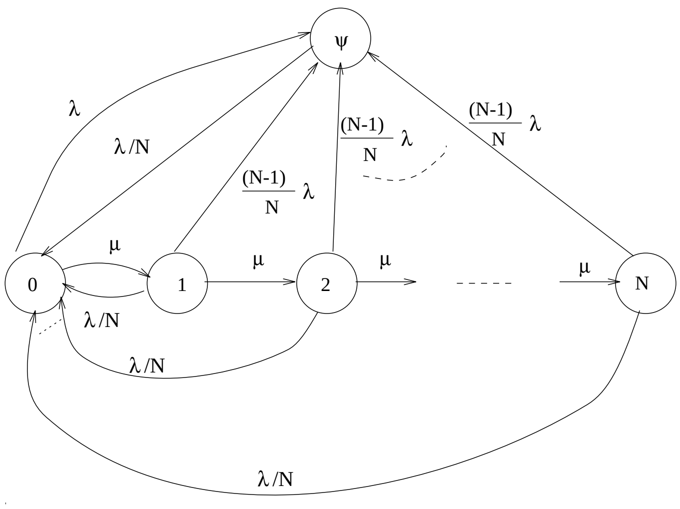

Figure 26: State Transitions when Search-Updates are used with Search-based Heuristic

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

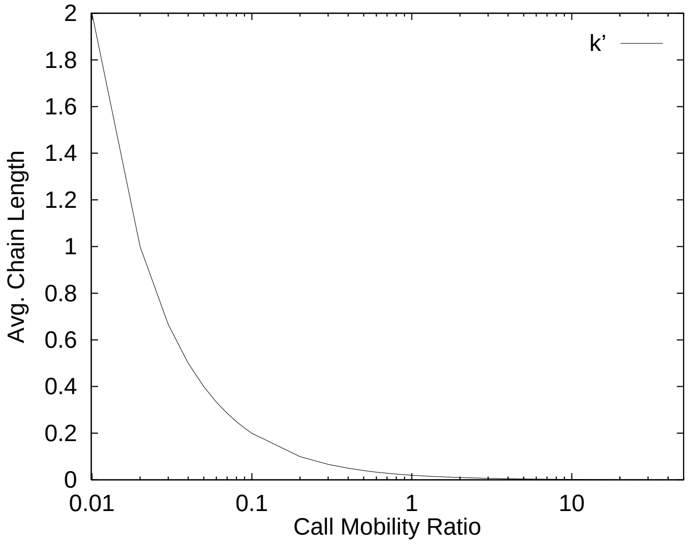

Figure 27: Variation of k' with r (Search-based) for N=50

For $N = 50$, Figure 27 illustrates the variation of k' when r changes, and Figure 28 illustrates the variation of s with r. Unlike the movement-based heuristic, the hit probability for search-based heuristic is very low and also reduces with increasing r. This is because in search-based heuristic an HLS update takes upon every search. The forwarding pointer maintenance policy explained in Section 5.3 requires that the forwarding pointers be purged

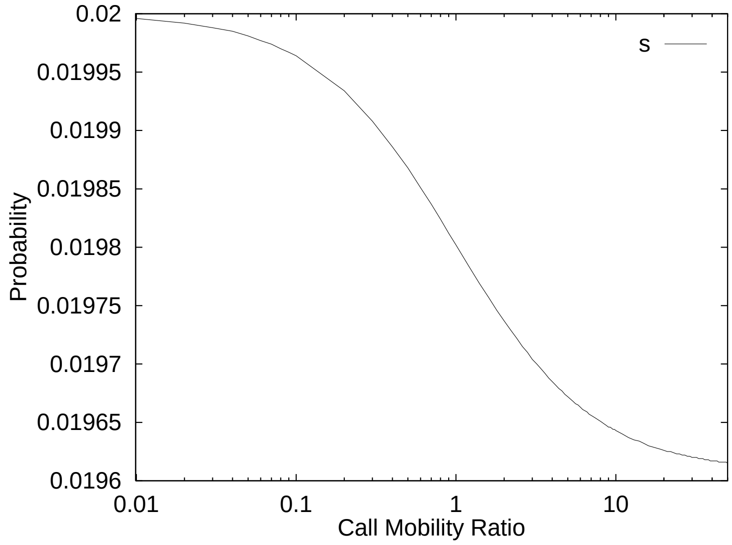

Figure 28: Variation of s with r (Search-based) for N=50

during every HLS update. Thus, as r increases, the lifetime of forwarding pointer at a location server reduces, thus the hit probability reduces too.