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
Phone: (409) 862-3411

Technical Report # 95-011

Abstract

The existing wireless cellular networks are likely to be upgraded for personal communication services (PCS). Studies indicate that the anticipated signalling load due to the location management functions for PCS will be significant using current techniques [4, 14]. This report proposes alternate techniques to reduce network load due to location management functions for PCS users.

Location management is one of the most important issues in mobile wireless networks. It consists of location update and search protocols. The trade-off in location management is between the cost of searches and the cost of updates. The goal of an "efficient" location management strategy is to reduce the search and update costs. In this report, we show that using forwarding pointers in addition to home location servers helps in providing efficient location management. It is also shown that using forwarding pointers reduces the network load during updates, but may increase the search cost due to long chain lengths. We present two heuristics, namely, *movement-based* and *search-based*, to limit the number of forwarding pointers traversed during a search. We analyze the performance of these heuristics and compare them with the IS-41 standard. The performance of the heuristics depends mainly on (i) call-mobility ratio of the user, and (ii) cost of forwarding. Our approach is shown to achieve significant performance improvement over the IS-41 scheme. It was also observed that for most of the call-mobility patterns, the *search-based* heuristic performed better than the *movement-based* heuristic.

On the downside, forwarding pointers (i) are prone to failures, (ii) might require high maintenance costs, and also (iii) might incur a large memory overhead. Omission/corruption of a forwarding pointer could cause the mobile host to be intractable. To overcome this problem, we propose techniques for fault tolerance and automatic recovery of the network. We propose cost-effective techniques for forwarding pointer maintenance. We show that the memory overhead required to implement the forwarding pointer scheme is insignificant.

A *search-update* occurs after a successful search, when the location information corresponding to the searched mobile host is updated at some hosts. This report proposes a strategy for search-updates. Performance analysis of the strategy shows that performing *search-updates* significantly reduces the search costs and the network load.

*Research reported is supported in part by AFOSR. Preliminary version of this report appears as a technical report [1].
†Direct all correspondence to P. Krishna.

1 Introduction

It is expected that existing wireless cellular networks will be upgraded for personal communication services (PCS). 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 the users are mobile and they could be anywhere in the area covered by the network.

An analysis in [4] shows that if location updates were 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 attempts to study the effect of forwarding pointers and search-updates on location management in Personal Communication Networks (PCN). Location management for PCS utilize the fact that there is a home location server (HLS) for every mobile host. There are existing standards for carrying out location management using home location servers (HLS), e.g., Electronic Industry Association/Telecommunication Industry Association (EIA/TIA) Interim Standard 41 (IS-41)[1], and Global System for Mobile Communications (GSM) in Europe [14]. However, these schemes are not efficient, due to increase in network load and location management costs. To overcome these drawbacks, this report presents location management strategies using forwarding pointers in addition to HLS. We present two heuristics to limit the length the number forwarding pointers traversed during a search. 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*. Analysis shows that that performing *search-updates* significantly reduces the search costs and the call-delivery rate at the HLS.

On the downside, forwarding pointers are prone to failures anywhere along the chain of pointers. Omission/corruption of a forwarding pointer could cause the host to be intractable. This report proposes techniques for fault tolerance and automatic recovery of the network. This report also present cost-effective techniques 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. Section 2 presents a network architecture for a distributed system with mobile hosts. Section 3 presents the network events which cause a change in the location information of a mobile host. Section 4 first presents the basic location management scheme that is being currently used in IS-41, and later presents the proposed location management scheme using forwarding

[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.

pointers. The performance analysis of the basic and the proposed location management schemes are presented in Section 5. Section 6 presents the search-update strategy and the corresponding performance analysis. Section 7 presents schemes to make the proposed location management scheme robust. Memory overhead analysis is presented in Section 8. Section 9 discusses algorithms to determine the call-mobility ratio of the users. Review of related literature is presented in Section 10, and conclusions are presented in Section 11.

2 Network Architecture

In this section we present a network architecture for a distributed system with mobile hosts [12, 13, 14]. The static network comprises of the fixed hosts and the communication links between them. Some of the fixed hosts, called *mobile support stations* (*MSS*) [2] are augmented with a wireless interface, and, they provide a gateway for communication between the wireless network and the static network. Due to the limited range of wireless transreceivers, a mobile host can communicate with a *mobile support station* only within a limited geographical region around it. This region is referred to as a mobile support station's *cell*. A mobile host communicates with one *mobile support station* (*MSS*) at any given time. *MSS* is responsible for forwarding data between the mobile host (mh) and static network.

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.

When a mobile host is engaged in a call or data transfer, it will frequently move out of the coverage area of the mobile support station it is communicating with, and unless the call is passed on to another cell, it will be lost. Thus, the task of forwarding data between the static network and the mobile host must be transferred to the new cell's *mobile support station*. This process, known as *handoff*, is transparent to the mobile user. Handoff helps to maintain an end-to-end connectivity in the dynamically reconfigured network topology.

3 Network Events

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*.

3.1 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 1) [13]. When a *MSS* receives a *registration* message from a mobile host, it forwards the message

[2] Mobile support stations are sometimes called *base stations*.

to its location server (step 2 in Figure 1)). 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.

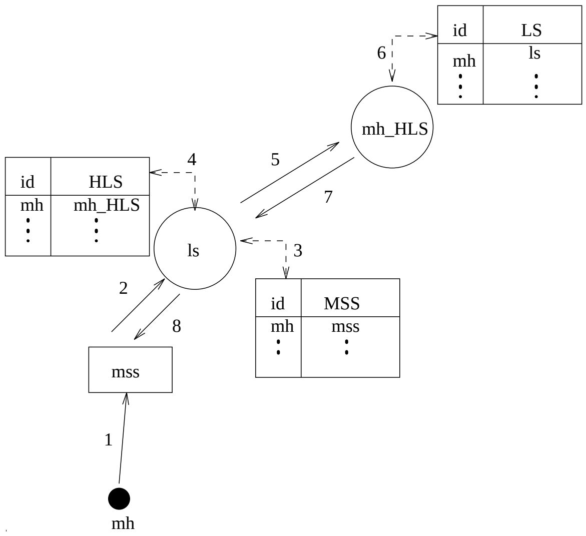

Figure 1: Mobile Host Switches ON

3.2 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.

3.3 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 report. 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 [13]. Please refer to Figure 2 for this discussion. The mobile host mh upon detecting a stronger signal from $mss2$ initiates the handoff procedure by sending a *handoff-init* message to $mss2$ (step 1 in Figure 2). 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.

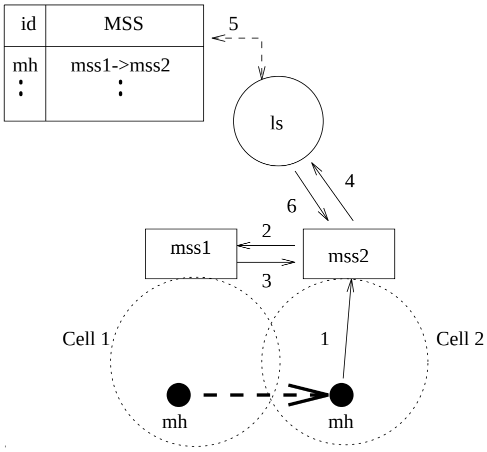

Figure 2: Handoff

3.4 Registration Area Crossing

In this report 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 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 will discuss the location management strategies for registration area crossings.

4.1 Home Location Servers (HLS)

Location management with home location servers is being used in current personal communication systems (PCS) standards proposals such as IS-41 [14]. In this section, we will first present the "basic" 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.

4.1.1 Updates in the Basic Scheme (IS-41)

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 3 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 3). The new_ls looks up its database to determine the home location server (HLS) of mh (step 3). The new_ls then sends a message to HLS notifying it about the new location of mh (step 4). The HLS updates the location information of mh (i.e., changes the current location server to new_ls), and sends the host information (user profile, etc.) to new_ls (steps 5-6). The location server new_ls stores the user profiles, updates its host database and sends an acknowledgement to new_mss (steps 7-8).

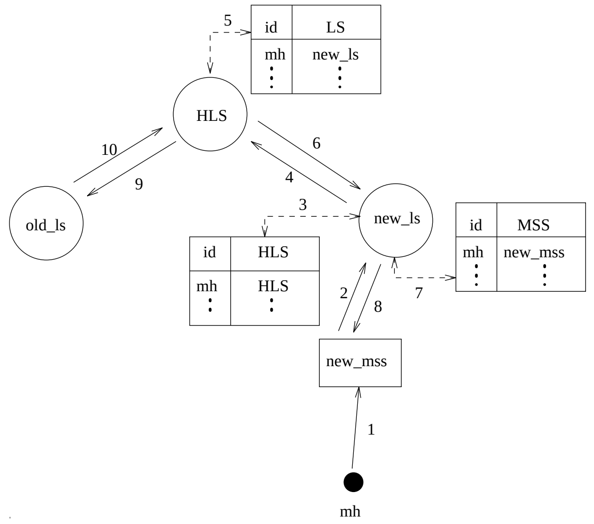

Figure 3: Update in the Basic Scheme (IS-41)

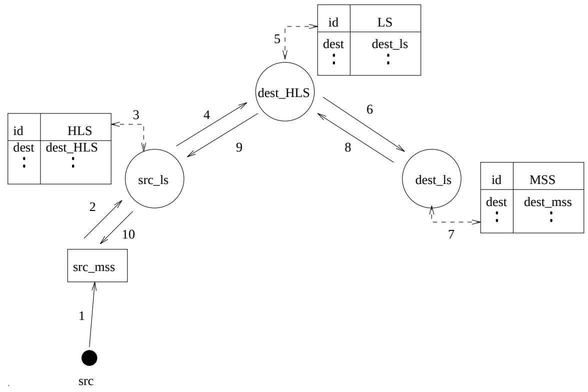

Figure 4: Search in the Basic Scheme (IS-41)

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 Scheme (IS-41)

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 4). The src_mss, forwards the query to the location server (say src_ls) in its registration area (step 2 in Figure 4). The location server src_ls checks whether $dest$ is in its registration area[3]. If $dest$ is in its registration area, then it returns the location information. Otherwise, src_ls looks up its database to determine the HLS of $dest$ (say $dest\_HLS$) (step 3). The location server src_ls then forwards the location query message to $dest\_HLS$ (step 4). The $dest\_HLS$ maintains the identification of the location server (say $dest\_ls$) of the registration area in which the last update of the location of host $dest$ took place (step 5). The $dest\_HLS$ queries the $dest\_ls$ for the cell location of $dest$ (step 6). In reply, $dest\_ls$ sends the identifier of the mobile support station $dest\_mss$. Upon getting the reply from $dest\_ls$ (steps 7 and 8), $dest\_HLS$ returns the current location of $dest$ to src_ls, which, in turn forwards it to src_mss (steps 9 and 10). The location information is nothing but the address of the MSS (named $dest\_mss$) of the cell in which the mobile host $dest$ currently resides. Thereafter, the call is set up between src and $dest$ via src_mss and $dest\_mss$.

[3]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.

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 not necessary. In such scenarios, updates lead to inefficient location management.

4.2 HLS and Forwarding Pointers

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, *move-updates* and *search-updates*. A *move-update* occurs when a mobile host changes location. A *search-update* (also known as caching [15]) occurs after a successful *search*, when some hosts update the location information corresponding to the searched mobile host.

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 [6].

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 5(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 5(b).

The following obervations can be made [6] :

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. □

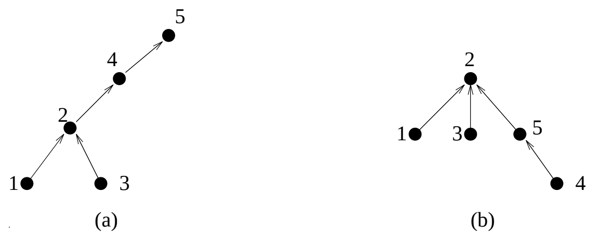

Figure 5: An Example

3) It follows from (1) and (2) that the graph formed by the forwarding pointers is always loop-free. □

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 5(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. These kind of loops are transient, and do not exist under stable conditions.

4.2.2 Move-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*.

Move-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 M, where 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 whenever a search takes place for the host.

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).

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 6 for this discussion. The first four steps of the search procedure are identical to steps

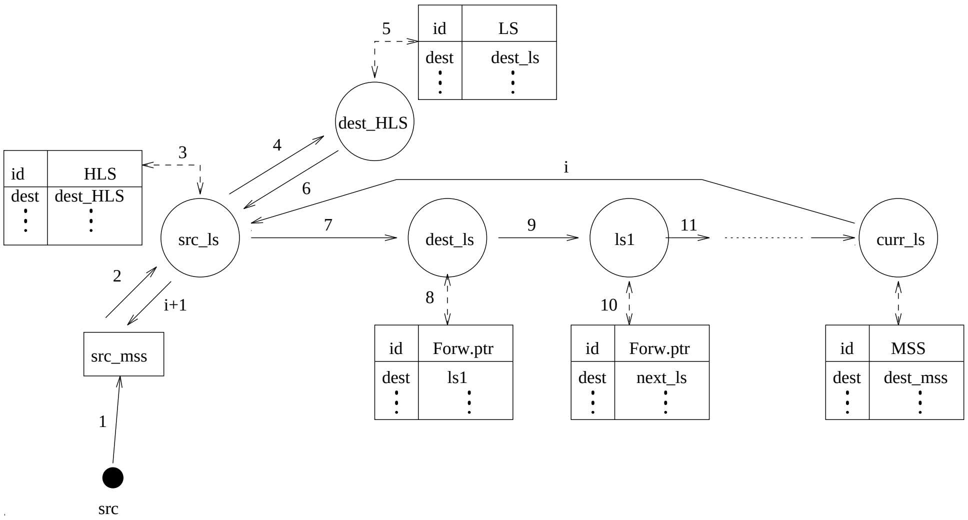

Figure 6: Search using HLS and Forwarding Pointers

1-4 in Figure 4, 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 6). Thereafter, the call is set up between src and $dest$ via the src_mss and $dest\_mss$.

4.2.4 Forwarding Pointer Maintenance

In the search-based heuristic scheme, a forwarding pointer for a host is used only once. Once a chain is traversed, the same chain is not going to be used again. This is because the HLS of a host gets updated after every search for the host. Therefore, a forwarding pointer could be purged as soon as it is used once. However, the same cannot be done if the movement-based heuristic is used, because, HLS does not get updated after every search. Thus, 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[4]. 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.

5 Performance Analysis

In this section we analyze the proposed location management scheme using forwarding pointers and compare it with the basic scheme using only HLS. 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 refer them simply as call arrivals), and for moves which are registration area crossings. We define "cost" of a sending a message as the amount of network usage due to sending the message. Let,

- r = average number of calls per registration area crossing. We call this the call-mobility ratio. This is little different than definition of call-mobility ratio in some previous literature [8].

[4] 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.

• C_u = cost of a move-update per registration area crossing using only HLS. We will derive an expression for C_u later.
• C_s = cost of a location search using only HLS. We will derive an expression for C_s below.
• $C_t = C_u + rC_s$ = total cost of move-update and searches per registration area crossing, when using only HLS.
• C'_u, C'_s, C'_t are the move-update, search and total cost, respectively, using HLS and forwarding pointers. $C'_t = C'_u + rC'_s$.
• cost$(x \rightarrow y)$ = cost of sending a message from x to y.
• S = 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.

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.

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 3 we can obtain,

$$\begin{aligned}
C_u &= \text{cost}(new_mss \rightarrow new_ls) + \text{cost}(new_ls \rightarrow HLS) + \text{cost}(HLS \rightarrow new_ls) \\
&\quad + \text{cost}(new_ls \rightarrow new_mss) + \text{cost}(HLS \rightarrow old_ls) + \text{cost}(old_ls \rightarrow HLS) \\
&= S + B + B + S + B + B \\
&= 2S + 4B
\end{aligned}$$

Step 1 involves message transfer over the wireless link, and hence is not considered in the cost analysis. Steps 3 and 5 in Figure 3 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 4, we can obtain,

$$\begin{aligned}
C_s &= \text{cost}(src_mss \rightarrow src_ls) + \text{cost}(src_ls \rightarrow dest_HLS) + \text{cost}(dest_HLS \rightarrow dest_ls) \\
&\quad + \text{cost}(dest_ls \rightarrow dest_HLS) + \text{cost}(HLS \rightarrow src_ls) + \text{cost}(src_ls \rightarrow src_mss) \\
&= S + B + B + B + B + S \\
&= 2S + 4B \qquad (1)
\end{aligned}$$

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

Now we will evaluate the schemes with forwarding pointers. Let,
• k = For a caller host, this is the number of forwarding pointers traversed before locating the destination host.
• M_u = number of registration areas crossed by a host when the proposed scheme performs a move-update for that host. The number of forwarding pointers due to moves by the host when a move-update

takes place for that host is $K_u = (M_u - 1)$.

- 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.

Since the number of forwarding pointers for a host when a move-update takes place is K_u, $K_u F$ is the total cost of creating forwarding pointers before a move-update takes place for that host. Therefore the move-update cost per registration area crossing for the proposed scheme is,

$$C'_u = \frac{1}{M_u}(K_u F + C_u)$$

From Figure 6 we can obtain the search cost for the proposed scheme as follows :

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

Step 1 in Figure 6 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,

$$\begin{aligned} C'_s &= S + B + B + kF + kF + S \\ &= 2S + 2B + 2kF \end{aligned} \qquad (2)$$

From equations (1) and (2), we get,

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

The maximum search cost will occur when the number of forwarding pointers traversed upon a search is maximum possible. This is the number of forwarding pointers for the host just before a move-update, i.e., K_u. Therefore,

$$\max_k(C'_s) = S + \frac{C_s}{2} + 2K_u F$$

The worst case average value of k will be $K_u/2$ (See Appendix 1). We assume the worst case average value $K_u/2$ to be the average value of k. Therefore, the actual performance gains should be more than what is derived here. The average search cost is as follows :

$$C'_s = S + \frac{C_s}{2} + 2\frac{K_u}{2}F$$

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

5.1 Analysis

The relative total cost is given as follows :

$$\begin{aligned}\frac{C'_t}{C_t} &= \frac{rC'_s + C'_u}{rC_s + C_u} \\ &= \frac{r(S + \frac{1}{2}(C_s + 2K_uF)) + \frac{1}{M_u}(C_u + K_uF)}{rC_s + C_u}\end{aligned} \tag{3}$$

We assume that the cost 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 cost of setting a forwarding pointer and traversing a forwarding pointer is assumed to be less than the cost of move-updates and searches. It is assumed to be a fraction of the cost of move-update and searches, i.e., $F = \alpha C_u$, where, $\alpha \leq 1$. Substituting $F = \alpha C_u$, $S \ll C_u$, $C_s = C_u$ in equation (3) yields,

$$\frac{C'_t}{C_t} = \frac{2(\frac{r}{2} + \frac{1}{M_u})(1 + \alpha K_u) + \alpha r K_u}{2(r+1)} \tag{4}$$

The relative search cost is given as follows :

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

As explained in the previous section, there are two heuristics to decide when to perform a move-update at the HLS. The two heuristics are analyzed next.

5.1.1 Movement-based Heuristic

With this heuristic, the location information at the HLS gets updated every M registration area crossings, where, M is a constant parameter. The number of forwarding pointers will be at most $M-1$. Thus, replace $M_u = M$, and $K_u = M - 1$ in the above equations.

$$\frac{C'_t}{C_t} = \frac{1}{2M}\left(\frac{(1 + Mr)(1 + 2\alpha(M-1)) + 1}{r+1}\right) \tag{6}$$

$$\frac{C'_s}{C_s} = \frac{1}{2}(1 + 2\alpha(M-1)) \tag{7}$$

5.1.2 Search-based Heuristic

A HLS update takes place upon every search. First consider the case when $r \leq 1$. Since, there are r calls per registration area crossing, $M_u = 1/r$, and $K_u = M_u - 1$, $K_u \geq 0$. Replacing these values in equations (4) and (5) we get for $r \leq 1$,

$$\frac{C'_t}{C_t} = \frac{3r + 4\alpha - 4\alpha r}{2(1+r)} \tag{8}$$

$$\frac{C'_s}{C_s} = \frac{1}{2r}(r + 2\alpha(1-r)) \tag{9}$$

When $r > 1$, on average, there is more than one call between two registration area crossings. This means that, if search-based heuristic is used, there will be more than one move-update at the HLS between two registration area crossings. This is not necessary, because the host has not changed its location. Therefore, we modify the search-based heuristic for $r > 1$ to update the HLS only once upon every registration area crossing. There will be no forwarding pointers. Thus, the move-update cost C'_u will be equal to C_u. However, the search cost C'_s will be half of C_s because there is half the number of HLS interactions[5].

5.2 Performance Evaluation of the Heuristics

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).

Using forwarding pointers in addition to HLS updates make sense for users with high mobility. Instead of updating the HLS on every move, forwarding pointers can be used. Thus, the effectiveness of heuristics are to be checked for call-mobility ratios, $r \leq 1$. In this report, we evaluate the performance of the heuristics for call-mobility ratios, $r \leq 1$. A good heuristic should have both the *relative total cost* and *relative search cost* less than 1.

5.2.1 Performance of Movement-based Heuristic

Figures 8-10 plot the variation of *relative total cost* with r for different values of M and α. It is observed in the figures that, the *total relative cost* increases with M and α. For low α values, movement-based heuristic is better. For high α values, the movement-based heuristic performs poorly compared to the basic HLS scheme (*relative total cost* > 1 implies that the basic scheme is better). For example, when $\alpha = 0.9$, $M = 3$, and $r = 0.7$, the *relative total cost* is 1.5. This means that the total cost of the proposed forwarding scheme using movement-based heuristic is 50% more than the basic scheme. From Figure 11, it can also be observed that the relative search cost increases with M and α. Considering total cost as the performance index, movement-based heuristic performed better than the basic HLS scheme for the call-mobility ratios that satisfy the following :

$$r(2\alpha(M-1)-1) \leq \frac{2}{M}(M-1)(1-\alpha) \tag{10}$$

5.2.2 Performance of Search-based Heuristic

The relative total cost for search-based heuristic is sensitive to α. For low α, the relative total cost increases with r. This is because, as r increases, the relative move-update cost increases, and the search cost decreases. However, the decrease in search cost is less compared to the increase in the move-update cost. And since, the total cost includes the search cost and the move-update cost, there is an increase in the total cost as r increases.

[5] As stated earlier, we assume that the cost of HLS interactions is the dominant cost in C_s and C_u.

However, for high α, the relative cost decreases with r. This is because for high values of α, as r increases, the decrease in the search cost is much more than the increase in the move-update cost. The performance of search-based heuristic is poor for low values of r. This is due to very high search costs.

Figure 12 illustrates the variation of relative search costs with r for various values of α. It can be observed that the search cost is very high for low values of r. This is because for small r, the move-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.

Considering total cost as the performance index, search-based heuristic performed better than the basic HLS scheme for the following call-mobility ratios follows from equation (8) :

$$r \geq \frac{2(1-2\alpha)}{(1-4\alpha)} \tag{11}$$

5.2.3 Search-based vs. Movement-based

Now that we have determined the call-mobility ratios for which the heuristics perform better than HLS scheme, we now determine when search-based heuristic is better than movement-based heuristic. This can be found from equations (6) and (8). If there is a movement-based heuristic scheme which updates upon M moves, the search-based heuristic scheme will perform better (lower total cost) for the following call-mobility values :

- $0 < \alpha \leq \frac{1}{(1+M)}$ and $0 < r \leq \frac{1}{M}$.
- $\frac{1}{(1+M)} < \alpha \leq 1$ and $r \geq \frac{1}{M}$.

5.2.4 Observation

Forwarding pointers are not beneficial in environments where r is very low and α is very high. The following table gives us a fair idea about when forwarding pointers are beneficial. If they are beneficial, the table shows which heuristic performs the best. Movement-based move-updates might cause

r \ α	LOW	MEDIUM	HIGH
LOW	Call -based	Movement -based	Not Beneficial
HIGH	Movement -based	Call -based	Call -based

Figure 7: Performance Chart

unnecessary updates even if the host does not move far since it moves in and out of a registration area frequently (thereby causing a lot of registration area crossings). On the other hand, when the call-mobility ratio is very high, search-based move-updates will cause a lot of unnecessary updates. However, if α is high, search-based move-updates are preferable even for high values of call-mobility ratios.

The performance of the search-based heuristic can be improved for low values of r by bounding the length of the chain of forwarding pointers. The search cost of movement-based heuristic increases linearly with M. This is again due to the fact that a search, on an average, has to traverse a longer chain of forwarding pointers. In the next section we present a search-update strategy to reduce the search cost.

6 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 6) pointing to the start of the chain. 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 (i.e., *call set-up* time). The use of search-updates have previously been discussed for network architectures without home location servers [2, 3]. There we presented three search-update strategies, namely, (i) *no search-update*, (ii) *jump update*, and (iii) *path compression update*. The *no search-update* strategy has been analyzed in the previous section. In this section, we will analyze the effects of *jump updates* on the network loads in PCNs. *Path compression update* is omitted due to lack of space.

6.1 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.

6.2 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).

6.3 Forwarding Pointer Maintenance for Search-Updates

Forwarding pointer maintenance procedure described in Section 4.2.4 was basically a "storage space retrieval" procedure. 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 6.2, it should be ensured that the forwarding pointers at the location server are not stale at any point of time.

In search-based scheme without search-updates, a forwarding pointer is used only once. Once a chain is traversed, the same chain is not going be used again. This is because the HLS gets updated after every call anyway. Therefore, a forwarding pointer could be purged as soon as it is used once. Note that the same cannot be done if search-updates are used. This is because, the location entry for *dest* at *src_ls* might point to a location server which could have purged its forwarding pointer already due to some other host using the chain of forwarding pointers to locate *dest*.

The cases when a location information at a location server needs to be purged or updated when search-updates are used are :

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

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.

Let the HLS update take place after M_u registration area crossings. Apart from the periodic purging (due to timeouts), we have to purge the forwarding pointers which become stale after a move-update at the HLS. This will bound the maximum length of forwarding pointers traversed during a search to K_u, where, $K_u = M_u - 1$. 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 upper bound on the average number of pointers for a host in the network at any time is $K_u + rM_u$. The explanation is as follows : The maximum length of the chain before a move-update at the HLS is K_u, i.e, there is a maximum of K_u 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 upper bounded by rM_u. Therefore, the upper bound on the average number of location servers that have forwarding pointers created due to search-updates is rM_u. Therefore, the average number of purge messages sent during an HLS update is upper bounded by $K_u + rM_u$.

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 (Section 3.2). The HLS will also know the location of the host whenever it reconnects/switches ON (Section 3.1). 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

6.2) 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 during our analysis of the search-update scheme. Since we are mainly concerned with the network load due to the host movement, we do not consider the network load due to disconnections in our analysis.

6.4 Performance Analysis

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 move-updates at the HLS take place only upon searches (with search-based heuristic) or upon moves (with movement-based heuristic). We also assume that the host is never in the disconnected mode. We analyze the performance improvement of search-based and movement-based heuristics due to search-updates. Let :

• s = 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 corresponding move-update, search, search-update and total cost using HLS and forwarding pointers. $C''_t = C''_u + r(C''_s + C''_{su})$.

• k' = For a caller host, this is the number of forwarding pointers traversed before locating the destination host when search-updates are used.

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

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

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 $(K_u + rM_u)$. 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 $(K_u + rM_u)F$. Therefore,

$$C''_u = C'_u + \frac{F}{M_u}(K_u + rM_u)$$

$$C''_s \quad = \quad s\{\text{cost}(src\_mss \rightarrow src\_ls) + \text{cost}(\text{traversing the forwarding pointers to } curr\_ls)$$

$$
\begin{aligned}
& +\text{cost}(curr\_ls \rightarrow src\_ls) + \text{cost}(src\_ls \rightarrow src\_mss)\} + (1-s)\{C'_s\} \\
= \ & s(S + k'F + k'F + S) + (1-s)(2S + 2B + 2k'F) \\
= \ & 2S + 2k'F + 2(1-s)B
\end{aligned}
$$

We assume that the average value of k' is equal to $K_u/2$. This is a pessimistic assumption. In an analysis presented in the appendix, we show that typically the average value of k' is less than $K_u/2$. Replacing $k' = K_u/2$, we get the average search cost using search-updates as,

$$
\begin{aligned}
C''_s &= C'_s - 2sB \\
&= C'_s - s(S + C_s/2)
\end{aligned}
$$

As stated earlier, the cost of search-update, C''_{su}, is basically the cost of sending a message to src_ls to create a location entry. Therefore, C''_{su} is equal to F.

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

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

$$
\frac{C''_t}{C'_t} = 1 - \left(\frac{r(s-2\alpha)}{2C'_t} - \frac{\alpha}{C'_t}(\frac{K_u + rM_u}{M_u})\right) \tag{12}
$$

$$
\frac{C''_s}{C'_s} = 1 - \frac{r(s-2\alpha)}{2C'_s} \tag{13}
$$

For search-updates to be beneficial, we require $\frac{C''_t}{C'_t} \leq 1$. The value of s for search-updates to be beneficial is given as follows,

$$
s > (\frac{2\alpha}{rM_u}(K_u + rM_u) + 2\alpha) \tag{14}
$$

For movement-based heuristic, we replace $M_u = M$, and $K_u = M - 1$. We obtain,

$$
s > \frac{2\alpha}{rM}(M - 1 + 2rM)
$$

Similarly for search-based heuristic, we replace $M_u = \frac{1}{r}$, and $K_u = \frac{1-r}{r}$, $r \leq 1$. When $r > 1$, we replace $M_u = 1$, and $K_u = 0$. Substituting the values of M_u and K_u in equation (14) yields,

$$
\begin{aligned}
&\text{when} \quad r \leq 1, \quad s > \frac{2\alpha}{r}(1 + r) \\
&\text{when} \quad r > 1, \quad s > 4\alpha
\end{aligned}
$$

The value of s and the average value of k' for a particular *dest* depends on the purge time interval T_{purge} for *dest*. They are also location dependent, since the rates of searches (calls) destined for *dest* may vary widely across location servers. In practice, however, we expect that the calls for a particular user will display locality.

In the analysis, we have assumed that the average value of k' is also equal to $K_u/2$. In Appendix 2 we present an analysis where we determine the value of s and the average value of k' for *jump updates*. Under certain simplifying assumptions, we show that the average value of k' is typically much lesser than $K_u/2$. Thus, the performance gains determined here in equations 12 and 13 for search-updates are actually the lower bounds since we assume a higher average value of k' (i.e., $K_u/2$).

Studies indicate that the call-delivery (or searches) rate at HLS is very high [14]. The search request (call-delivery) rate at the HLS can be reduced using search-updates, because most of the time, location entry for the host can be obtained at the location server itself. For example, for movement-based heuristic, when $K = 2$, $r = 5$, the value of s is determined by the analysis presented in Appendix 2 to be 0.25 (See Figure 16), i.e., 25% of the search requests are satisfied at the location server itself. Therefore, for this example, the call-delivery rate at HLS is reduced by 25%.

However, it is noticed from equation (14) that the value of s depends critically on the value of α. Due to the high purge costs involved, search-updates will be only beneficial in networks which have low α.

7 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 ensures that these errors are handled automatically.

Fault tolerance requires that regardless of the initial state, the system will eventually converge to the correct state. 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 "bidirectional" 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 "bidirectional" 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) 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 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.

8 Memory Overhead

A limitation of the forwarding pointer approach is 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 forwarding pointers into two types :

• 1) 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.

• 2) Bi-directional Pointers : This is required if the network needs to be fault-tolerant, and 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 [14]. 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. Thus, 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 $\frac{\rho v L}{\pi}$ [17]. Replacing the values, we obtain $R = 5.85/s$. Let the maximum number of pointers that exists 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 cost of memory is of the order of \$40 for 1 Mbyte of RAM. Since we require a fast access for checking the forwarding pointer database, these pointers have to be stored in a RAM in the location server. 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.

9 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 [15]. In our earlier work [3], we proposed algorithms to estimate r based on the history of call and mobility patterns. The basic assumption behind these algorithms were that past history of the user's call mobility patterns will reflect the behavior in future.

10 Related Work

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 [6]. Fowler deals with techniques to efficiently use forwarding addresses for finding decentralized objects. Our report borrows the idea of manipulating forwarding pointers upon a successful search.

Awerbuch et. al. proposed a theoretical model for online tracking of mobile hosts [9]. 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*) [10]. 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 was 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.

Ioannidis et. al. proposed IP-based protocols for providing continuous networks access for mobile computers using caching and forwarding technique [7]. However, these protocols were primarily proposed for a campus environment with mobile computers.

In our earlier work, we discussed location management schemes using forwarding pointers and search-updates for network architectures consisting of a hierarchy of location servers. This was done without using designated home location servers [2, 3]. The proposed location management strategies were a combination of the search strategy, a update strategy, and a search-update strategy. Simulations were carried out to evaluate the performance of the various location management strategies. It was noticed that performing search-updates significantly reduced network load.

The idea of location management with home location servers has been earlier proposed in the literature [8, 14, 15]. Caching techniques have been proposed for Personal Communication Services (PCS) [15]. Here, an auxillary strategy called *per-user caching* is proposed to locate users who move from place to place while using PCS. This strategy augments the basic location strategy which uses *home location servers* (HLS). Badrinath et. al. examined strategies that reduce search costs and control the volume of location updates by employing user profiles [8]. 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. Each user is assumed to be registered under one of the location servers called the *home location server* (HLS). The user profiles were used to create partitions. It is only when the user crosses partitions that the update takes place. However, user profiles are not always available a priori.

11 Conclusions

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 analyzed the performance of these heuristics, and compared them with the scheme used in the IS-41 standard. It was observed that using forwarding pointers was beneficial for most of the call-mobility ratios. However, when the relative cost of forwarding pointer operations is high, forwarding pointers are not beneficial for low call-mobility ratios. We also observed that the search-based heuristic performed better than the movement-based heuristic for most of the call-mobility ratios and α values. Using search-based heuristic reduces the *call set-up* time (search cost) significantly for high values of call-mobility ratios and α. The performance of location strategies using forwarding pointers and search-updates were analyzed. Using search-updates, significant reduction in call-delivery rate at the HLS was obtained.

The main assumptions made in this report 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 was mainly motivated by the studies which indicate HLS to be the bottleneck in PCNs. 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 could be selected for the user based on the user's call-mobility ratio (r) and the network design parameter (α). This would result in lower network load enabling the network to support more mobile hosts than it would using the basic HLS strategy proposed in IS-41 standard.

Future Work : In this report we did not consider any particular network architecture to analyse the schemes. It would be interesting to determine the effects of forwarding pointers on the network load in different network architectures (topologies, location of HLS, etc.). Secondly, in this report we did not consider *variance* in the *call set-up* times. Future work should involve development of schemes that reduce the *variance*.

In our earlier work we found that if the call-mobility patterns of the user were arbitrary, *path compression* performed better than *jump update* [2, 3]. However, the analysis was based on a network architecture sans a home location server, and comprising of a hierarchy of location servers. It will be interesting to analyze the effects of *path compression* update in location management in PCNs.

References

[1] 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.

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

[3] P. Krishna, N. H. Vaidya and D. K. Pradhan, "Static and Dynamic Location Management in Distributed Mobile Environments," Technical Report # 94-030, Dept. of Computer Science, Texas A&M University, June, 1994.

[4] 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.

[5] L. J. Ng, et. al., "Distributed architectures and databases for intelligent personal communication networks," *Proc. of IEEE ICWC,* 1992.

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

[7] J. Ioannidis, et. al., "IP-based Protocols for Mobile Internetworking," *Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols,* pp. 235-245, October 1991.

[8] 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.

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

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

[11] Amotz Bar-Noy and H. Kessler, "Tracking Mobile Users in Wireless Communication Networks," *Proc. Infocom*, 1993.

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

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

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

[15] R. Jain, et. al., "A Caching Strategy to Reduce Network Impacts of PCS," *IEEE Journal on Selected Areas in Communications,* Vol. 12, No. 8, Oct., 1994.

[16] C. Lo and R. Wolff, "Estimated Network Database Transaction Volume to Support Wireless Personal Data Communications Applications," *Proc. of Intl. Conf. Communications,* May, 1993.

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

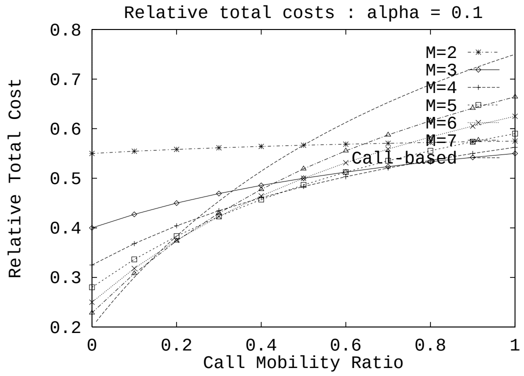

Figure 8: Relative Total Costs : $\alpha = 0.1$

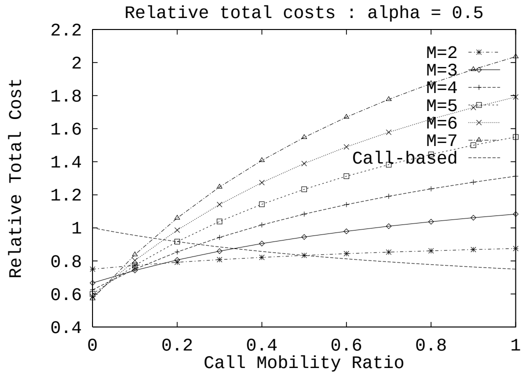

Figure 9: Relative Total Costs : $\alpha = 0.5$

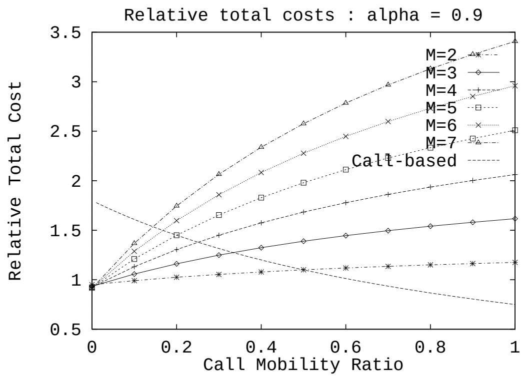

Figure 10: Relative Total Costs : $\alpha = 0.9$

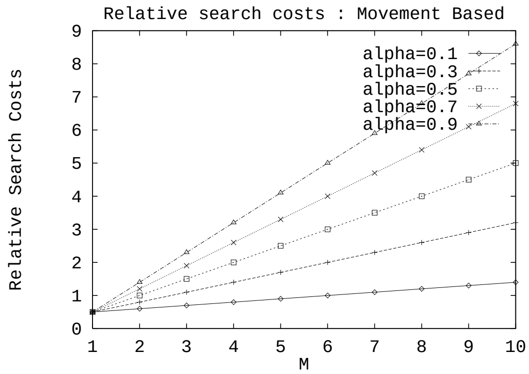

Figure 11: Relative Search Costs : Movement-based

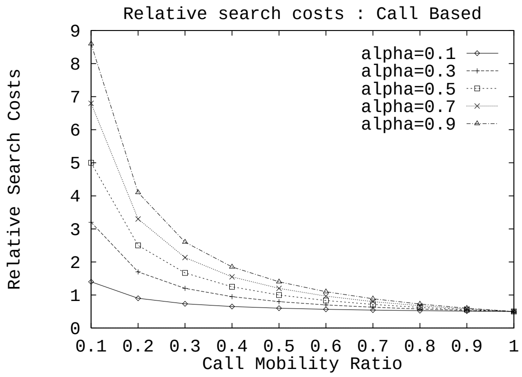

Figure 12: Relative Search Costs : Search-based

Appendix 1 : Calculation of k

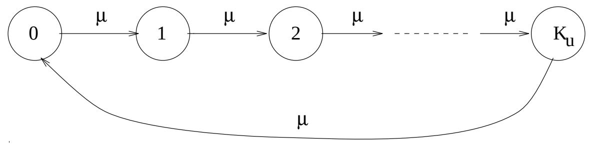

Figure 13: State Transitions

To determine the average value k we model the number of forwarding pointers traversed during a search by a markov process. We assume that the call arrivals are a poisson process with an arrival rate λ (Though we will not be using this information here, it will be used in our analysis later). We also assume that time interval between two registration area crossings has an exponential distribution with a mean $1/\mu$. We assume that during the life of the chain, the host does not return to a registration area. Thus, the number of forwarding pointers traversed during a search always increases when a host moves (crosses registration areas). However, in practice, the host could move to any of the neighboring registration areas, which includes the registration area it was previously in. Thus, this analysis will give a pessimistic value of k.

Figure 13 illustrates the state transitions. Let an HLS update take place when the length of the chain becomes K_u. State i represents the state where the length of the chain is i. Let the probability of the process being in state i be P_i, $0 \leq i \leq K_u$. Since, each state has an equal probability, and there are $(K_u + 1)$ states, $P_i = \frac{1}{(K_u+1)}$. Thus, the expected value k is $\sum_{i=0}^{K_u} iP_i = K_u/2$.

Appendix 2 : Calculation of s, and average value of k' for search-updates

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.
- We assume that the move-updates at the HLS is based on movement-based heuristic, i.e., the HLS update takes place every M registration area crossings.
- We assume that at any time, the maximum length of the chain of forwarding pointers will be $K = M - 1$.
- 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 transistion diagram based on above assumptions is as shown in Figure 14. 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.

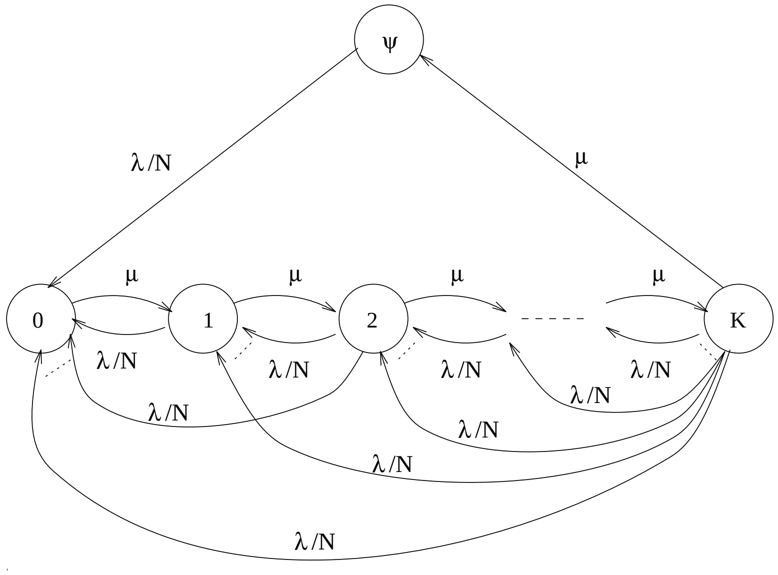

Figure 14: State Transitions when Search-Updates are used

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

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

Simplifying it we get,

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

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

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

Now, the state transition equation for state ψ,

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

The average value of $k' = \sum_{i=0}^{K} iP_i$, and the probability of finding a location entry at the location server, $s = 1 - P_\psi$.

For $K = 2$, the value of k' and s can be obtained as follows :

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

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

For $N = 50$, Figure 15 illustrates the variation of k' when r changes, and Figure 16 illustrates The variation of s with r. The maximum value of k' (which is 0.54) is much less than $K/2$ (which is 1.0). As shown in Figure 16, 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.

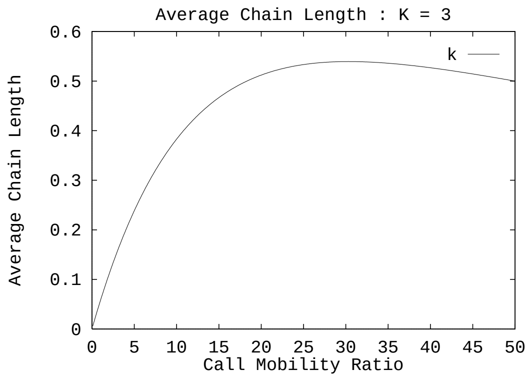

Figure 15: Variation of k with r for $K = 2$

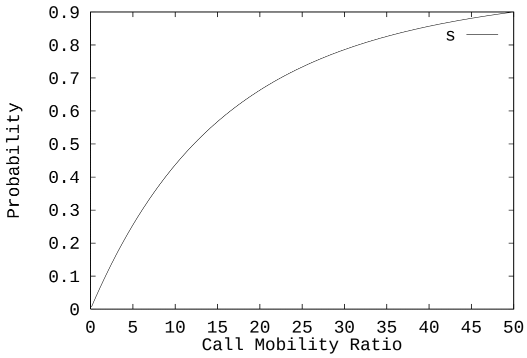

Figure 16: Variation of s with r for $K = 2$