A Dynamic Approach to Location Management in Mobile Computing Systems

Ravi Prakash Mukesh Singhal
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210.
E-mail: {prakash,singhal}@cis.ohio-state.edu

Abstract

Managing location information of mobile nodes is an important issue in mobile computing systems. There is a trade-off between location update effort (when a node moves) and node finding effort. In this paper we present a dynamic location management strategy that has the following features: (i) all location servers need not maintain location information about every mobile node, (ii) a coterie based approach is adopted for location update and find, (iii) every node move does not result in location updates, (iv) location updates are done at a subset of location servers, (v) a subset of location servers are queried when a mobile node is to be located, (vi) the set of location servers, corresponding to a mobile node, for location update and find operations is dynamic, (vii) the dynamic nature of these sets helps alleviate situations of heavy burden on some location servers, when a large number of mobile nodes are concentrated in a small geographical area. Thus, location management is done efficiently, and responsibility is shared fairly among location servers.

Keywords: mobile computing, location management, hashing.

1 Introduction

The ability of mobile hosts (MHs) to autonomously move from one part of the network to another part in a mobile computing system, sets it apart from static networks. Unlike static networks, the network configuration and topology keep changing in mobile computing systems. The mobility of some nodes in the network raises interesting issues in the management of location information of these nodes. Creating a fixed location directory of all the nodes *a priori* is not a solution. The location directory has to be dynamically updated to account for the mobility of the MHs.

The design of a location directory whose contents change dynamically raises important issues. Some of them are as follows: (i) When should the location directory be updated? If the updates are done each time an MH's location changes, the directory will always have the

latest location information, reducing the time and effort in locating an MH. However, such a policy imposes a heavy burden on the communication network and the location servers, *i.e.*, nodes that maintain the directory. (ii) Should the location directory be maintained at a centralized site, or should it be distributed? A central location server has problems with regard to robustness and scalability. Hence, a distributed directory server is preferable. This leads us to the next questions. (iii) How should the location information be distributed among the location servers? and (iv) Should location information about an MH be replicated across multiple location servers? It is not possible to *a priori* determine the variations in spatial distribution of MHs in the network, and the frequency with which node location will be updated or queried. Hence, a location management strategy should address issues (iii) and (iv) so as to ensure fair distribution of responsibility among all the location servers, and be scalable.

In this paper we propose a location management strategy in which the location directory is distributed among nodes in the static network, such that all the location servers share the responsibility fairly. Also, fluctuations in query rates of mobile hosts is accounted for, so that no location server is unduly burdened.

Section 2 describes the model of the mobile computing system that is assumed in the rest of the paper. In Section 3, previous work on location management is described. Some of the inadequacies of the previous schemes, which motivate our work, are enumerated in Section 4. In Section 5, we present the basic idea behind the proposed location management strategy. The data structures, and the algorithm for this strategy are presented and described in Section 6. Section 7 describes how the location servers are selected for location updates and queries. In Section 8, we describe how the location management strategy handles fluctuations in location query rates of mobile hosts. The performance of the strategy is analyzed in Section 9, and the conclusion is presented in Section 10.

2 System Model

We assume a cellular communication system that divides the geographical region served by it into smaller regions, called cells. Each cell has a base station, also referred to as the *mobile service station (MSS)*. Figure 1 shows a logical view of a mobile computing system. The mobile service stations are connected to each other by a fixed wire network. A mobile service station can be in wireless communication with the mobile hosts in its cell. The location of a mobile host can change with time. It may move from its present cell to a neighboring cell while participating in a communication session, or it may stop communicating with all nodes for a period of time and then pop-up in another part of the network.

A mobile host can communicate with other units, mobile or static, only through the mobile service station of the cell in which it is present. If a node (static or mobile) wishes to communicate with a mobile host, first it has to determine the location of the MH (the cell in which the MH is currently residing). This location information is stored at location servers. Depending on the frequency of location updates, this location information may be current, or out-of-date. Once the location of the MH has been determined, the information is routed through the fixed wire network to the MSS of the cell in which the MH is present. Then the MSS relays the information to the destination MH over a wireless channel. We

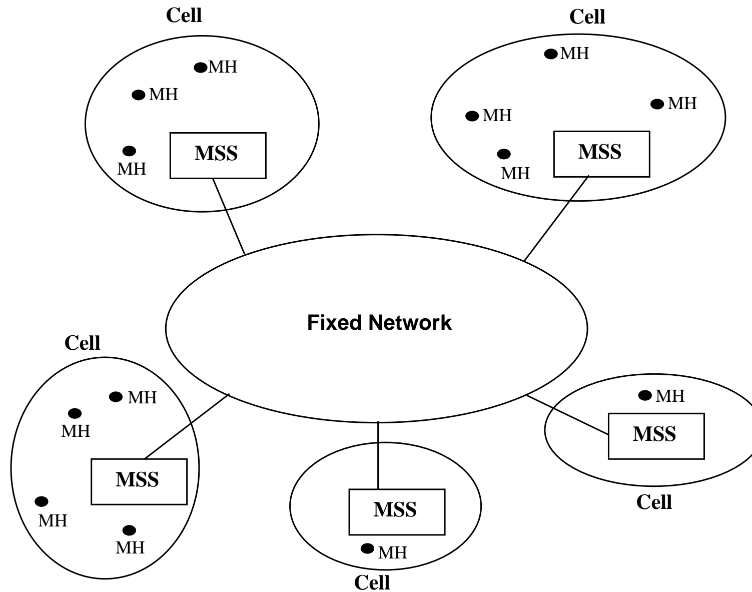

Figure 1: Logical view of a mobile computing system.

assume that MSSs act as location servers. Hence, all the MSSs collectively maintain the location directory.

3 Previous Work

Updating the location directory each time an MH moves from one cell to another can be very expensive. Three alternatives, namely, time-based, number of movements-based, and distance-based strategies for directory updates have been proposed in [3]. The location updates are done less often, and impose lower overheads.

A simple solution to location tracking is to have a centralized location server. However, if the node maintaining the directory crashes, location information about all the nodes becomes inaccessible. Also, a centralized directory is unable to exploit the geographical distribution of MHs in the system, and locality of reference patterns to minimize the cost of directory update and query operations.

The locality of reference patterns is exploited in [7]. The notion of *working set* for mobile hosts is introduced. Nodes in an MH's working set communicate with the MH more frequently than nodes that are not in the working set. A location management scheme has been described in [7] in which an MH can dynamically determine its working set depending on the call-to-mobility ratio between network node and MH pairs. Nodes in the working set are informed about the location update when an MH moves, while other nodes are made to search for the MH when they wish to communicate with the MH.

In [2], some MSSs are designated as *reporting centers* (similar to location servers). Location update is done when an MH moves into the cells corresponding to the reporting centers. When an MH has to be located, it is searched for in the vicinity of the reporting center at which the last update was made. However, an issue that needs to be addressed is how such a reporting center is determined. One simple solution would be to probe *all* reporting centers to determine the one with the latest update. However, this imposes high

communication overheads on the fixed network.

In [1], a hierarchy of distributed regional directories is maintained. The ith level regional directory enables a node, static or mobile, to track any mobile host within a distance of 2^i from it. Corresponding to each level i, *read* and *write* sets of nodes are associated with nodes u, v such that $read_i(u) \cap write_i(v) \neq \phi$, $\forall u, v$ within 2^i distance from each other. The *write* set for a node is the set of nodes where the location information of the node is stored. The *read* set for a node is the set of nodes that will be probed to find the location of a target node.

4 Motivation

In all the schemes described above, even though the location directory is distributed across several MSSs, the responsibility of location tracking is not guaranteed to be shared equally. This is due to the following reasons:

1. The geographical distribution of MHs may change with time. Quite often a significant fraction of MHs are concentrated in a very small area, while there is a very low density of MHs in the rest of the network. For example, most of the MHs may be situated on the highways and other major streets of a city during the morning and evening rush-hour traffic, and most of the MHs may be concentrated in the business districts of the city during rest of the day. In such situations, the directory servers [1] and reporting centers [2] in the high density regions will be overburdened, while the directory servers and reporting centers in other regions will be comparatively lightly loaded.

2. Location of some MHs will be queried more often than others. So, even when the MHs are evenly distributed across the network, the location servers (directory servers in [1] and reporting centers in [2]) for these MHs will be queried more often, increasing their load. If the identity of such MHs were known *a priori*, appropriate actions could be taken to distribute the load. However, such is not the case. An MH that is *hot* (location queried frequently) may go *cold* (location queried infrequently), and vice versa.

Hence, there is a need for a distributed location directory management scheme that can adapt to changes in geographical distribution of MH population in the network, and to changes in MH location query rate.

5 Basic Idea

The problem at hand is as follows: given an MH, determine the location server(s) that will store the location of the MH.

Storing the location information of an MH at *only* one MSS (serving as the MH's location server) is not desirable due to the following reasons:

1. MHs exhibit a spatial locality of reference: even though all nodes in the system can potentially communicate with the network, bulk of the references originate from only a subset of them (referred to as the *working set* in [7]). The nodes in the working set may be clustered in different parts of the network. So, to reduce query costs, it is advisable to have location servers for the MH in the vicinity of such clusters.

2. Multiple location servers for an MH make the distributed directory tolerant to the failure of some of these servers.

So, let there be a function $f : MH \rightarrow S_{MSS}$, which given the identity of an MH, determines the set of MSSs that are the location servers for that MH. However, using only the MH's identity to determine its location servers fails to exploit the locality of reference characteristics of mobile networks. Regardless of where the MH is located in the network, its location information will always be stored at the same nodes. Usually, an MH is more likely to be in communication with nodes in its vicinity, than with nodes at a greater distance. As a result the working set of an MH can change with its location. Therefore, associating a static set of location servers with each MH is not advisable.

The above problem can be solved as follows: let there be functions (i) $g' : MH \rightarrow MSS$, which maps an MH to the MSS of its current cell, and (ii) $g'' : MSS \rightarrow S_{MSS}$, which maps an MSS to a set of MSSs. Then, we can define a function $g : MH \rightarrow S_{MSS}$ to be equivalent to $g''(g'(MH))$. Given an MH, function g determines the location servers of the MH based on the cell in which the MH is present. So, the location servers of an MH change as the MH's location changes.

However, determining the location servers of an MH based solely on the cell in which that MH is present will lead to uneven distribution of responsibility. For example, when there is a high concentration of MHs in a cell, all these MHs will be mapped to the same location servers which will be overburdened. Hence, it is desirable that the location servers storing the location information of an MH be a function of the identities of the MH as well as the cell in which that MH is present. Such a function can be represented as follows: $h : MSS \times MH \rightarrow S_{MSS}$. The location servers corresponding to an MH will change as the MH moves in the network. Also, MHs in the same cell need not have the same set of location servers.

Nodes that wish to locate an MH should be able to access at least some of the location servers of the MH quickly, and in an inexpensive fashion. Broadcasting a query to the entire network is an expensive solution. Function h, described above, can also be employed to determine the set of location servers that should be queried when an MH is to be located. The set $h(MSS, MH)$ can represent the set of MSSs that a node, in the cell represented by MSS, should query when it wishes to locate a mobile host MH. Thus, function h determines the *write set* for location updates when an MH moves, and the *read set* for querying the location of the MH.

A naive implementation of the function h would be to have a look-up table with an entry for each (MH, MSS) pair. Each entry would store the corresponding S_{MSS}. The size of such a table will be large as there are a large number of MHs in a mobile computing system. Even if the location directory were to be distributed across all MSSs, each MSS would have

to store its share of the look-up table, having as many entries as the number of MHs in the system.

In order to avoid the storage overheads incurred by such a table, there is a need for function h to be computationally inexpensive, mapping (MH, MSS) pairs to S_{MSS}. It will be desirable if the mapping is distributed over the range of MSSs, for fairness.

MHs that are *hot* will have their locations queried more frequently than other MHs. Nodes querying the locations of *hot* MHs may be spread all over the network. Therefore, a greater number of location servers, spread throughout the network, should maintain location information about *hot* MHs. Fewer location servers need to maintain location information about *cold* MHs. For this purpose, alias(es), referred to as virtual MH identity, are assigned to each MH. A hot MH is assigned multiple virtual identities, while a cold MH is assigned a single virtual identity, i.e., the location management scheme considers a hot MH to be equivalent to multiple cold MHs.

Determination of location servers for an MH involves three steps:

1. Mapping the MH identity (MH_id) to a virtual MH identity (VMH_id): Let each MH_id correspond to a non-negative integer. Non-negative integers are also used to represent VMH_id. Let there be an integer constant x which is a parameter of the algorithm, and is determined *a priori*. If an MH is *cold*, its $VMH_id = MH_id + x$. If an MH is *hot*, then it is assigned multiple virtual identities. Solely for the purpose of explanation in this paper, we assume that each *hot* MH is assigned at most two virtual identities: $VMH_id_1 = MH_id + x$, and $VMH_id_2 =$ an integer between 0 and $x - 1$ (inclusive) that has not already been assigned as a virtual identity to some other *hot* MH.

   When a *hot* MH turns *cold*, it relinquishes its second virtual identity which is returned to the pool of available virtual identities, and can be assigned to *hot* MHs in the future. It is to be noted that the first virtual identity of each node is unique as the MH_id of each MH is assumed to be unique.

   Thus, the location management scheme can efficiently handle up to x hot MHs: each being assigned its second virtual identity from the range $[0, x - 1]$ of integers. If there are more than x hot MHs, then the surplus hot MHs can be assigned only one virtual identity each, and are treated no different from cold MHs.

2. Given an MSS_id, denoting the cell in which the mobile host is present, and a VMH_id for that mobile host, we employ the principle of open addressing hash function, specifically the double hashing [4], as follows:
   $h(MSS_id, VMH_id) = (h_1(MSS_id) + VMH_id \times h_2(MSS_id))\ mod\ m$
   where m is the number of MSSs in the network, and h_1 and h_2 are auxiliary hash functions. Functions h_1 and h_2 are uniformly distributed over the range $[0, m - 1]$, and $h_2(MSS_id)$ is relatively prime to m. Given an (MSS_id, VMH_id) pair, $h(MSS_id, VMH_id)$ will be uniformly spread over the range $[0, m - 1]$, *i.e.*, mapped uniformly over all the MSSs in the network [4].

3. Corresponding to each $i = h(MSS_id, VMH_id)$, there is a set S_i of MSSs with the following properties:

(a) $\cup_{i=0}^{m-1} S_i = S$, where S is the set of all MSSs.

(b) $S_i \not\subset S_j$, for $0 \leq i, j \leq m-1, i \neq j$.

(c) $S_i \cap S_j \neq \phi$, for $0 \leq i, j \leq m-1$.

(d) $\mid S_1 \mid = \ldots \mid S_{m-1} \mid = K$.

(e) Any MSS_j, $0 \leq j \leq m-1$, is contained in K S_i's, $0 \leq i \leq m-1$.

Properties (d) and (e) represent the *equal effort* and *equal responsibility* properties, respectively. Together they represent the *symmetry* property.

So, if an MH with virtual identity VMH_id, in the cell corresponding to MSS_id, updates its location information, the update is done at all $MSS_i \in S_j : j = h(MSS_id, VMH_id)$. The set S_j of MSSs is the *write* set for the (MSS_id, VMH_id) pair. The value of j is evenly distributed over the range $[0, m-1]$. Hence, the responsibility for location tracking of all the *virtual MHs* (there are as many virtual MHs as the number of virtual identities assigned) is evenly distributed among the MSSs. The set S_i of MSSs is also referred to as a *quorum*.

Locating an MH: When a mobile service station with identity MSS_id, or an MH inside the cell corresponding to this MSS wishes to locate an MH whose identity is MH_id, following actions are taken by the MSS:

1. Determine all virtual identities, VMH_id, for the MH with identity MH_id.

2. $query_set \leftarrow \phi$.

3. (a) for all VMH_id, compute $j \leftarrow h(MSS_id, VMH_id)$; $query_set \leftarrow query_set \cup S_j$.
   OR

   (b) select one VMH_id for the MH; compute $j \leftarrow h(MSS_id, VMH_id)$; $query_set \leftarrow S_j$.

4. Send QUERY to all $MSS_i \in query_set$ to locate MH with identity MH_id. The $query_set$ is similar to the *read* set, described in previous algorithms, for the (MSS_id, VMH_id) pair.

5. If a queried MSS_i contains location information about MH_id, it sends this information in its response.

The significance of two different options in constructing the query_set (using all VMH_id's vs. only one VMH_id) will be explained later, in Section 8.

As $S_i \cap S_j \neq \phi$ for $0 \leq i, j \leq m-1$, the *read* and the *write* sets for every pair of tuples (MSS_1, MH_id) and (MSS_2, MH_id), respectively, are bound to intersect. Therefore, if the latest updated location of an MH is stored at a set of MSSs, S_i, then regardless of which cell originates a location query for the MH, at least one MSS belonging to S_i will be probed (property (c)). So, all queries are guaranteed to return the latest location information of the mobile hosts.

6 Data Structures and the Algorithm

The data structures required by the location management algorithm are:

assigned[0..x-1]: array of booleans. This is a global variable. The element assigned[i] is set to TRUE if integer i has been assigned as a virtual identity, VMH_id, to a mobile host that is hot. Otherwise, it is FALSE. Each element of the array is initialized to FALSE.

location[MH_id]: each MSS maintains a cache in which it stores its knowledge of the locations of some MHs queried by the MSS in the recent past. If there is a *location* entry for MH_id in the cache, the MSS first tries to find the target MH in the corresponding cell. Otherwise, it queries the distributed location directory server, as described below. It is to be noted that the cache entries may be out-dated as the MH may have moved to another cell

VMH_id(MH_id) : the set of virtual identities associated with an MH whose identity is MH_id. This set is maintained, on behalf of the MH, by the MSS of the cell in which the MH is resident. When the MH moves from one cell to another, the set is migrated from the MSS of the old cell to the MSS of the new cell.

6.1 The Algorithm

Next we present pseudo-code for components of the location management scheme.

```
assign_virtual_ids(MH_id)
{int i; boolean found;
 VMH_id(MH_id) ← {MH_id + x};
 i ← 0; found ← false;
 while(i<x and not(found))
        {if (assigned[i] = FALSE)
            {assigned[i]←TRUE;
             VMH_id(MH_id)←VMH_id(MH_id)∪{i};
             found ← TRUE;
            }
          i ← i+1;
        }
}
transition_hot_to_cold(MH_id, MSS)
{ int i, j, k;
  for all i ∈ VMH_id(MH_id) do
       if (i ≠ MH_id + x) do
          { j ← h(MSS, i);
            for all k ∈ S_j do
                 send(MSS_k, delete, MH_id, MSS);
```

```
        assigned[i] ← FALSE;
      }
}
transition_cold_to_hot(MH_id, MSS)
{ int i, j, k;
  assign_virtual_ids(MH_id);
  for all i ∈ VMH_id(MH_id) do
      if (i ≠ MH_id + x) do
        { j ← h(MSS, i);
          for all k ∈ S_j do
              send(MSS_k, add, MH_id, MSS);
        }
}
location_update(MH_id, old_MSS, new_MSS)
{ int i, j, vmh;
  for all vmh ∈ VMH_id(MH_id) do
      { i ← h(old_MSS, vmh);
        for all j ∈ S_i do
            send(MSS_j, delete, MH_id, old_MSS);
        i ← h(new_MSS, vmh);
        for all j ∈ S_i do
            send(MSS_j, add, MH_id, new_MSS);
      }
}
locate_MH(MH_id, MSS)
{ int i, j, k;
  if ((i=location(MH_id)) ∈ local cache)
     { send(MSS_i, QUERY, MH_id);
       wait(response from MSS_i);
       if (response = YES)
          return(response.location);
       else
          delete(location(MH_id) from local cache;
     }
  i ← any virtual identity of MH_id;
  j ← h(MSS, i);
  for all k ∈ S_j do
      send(MSS_k, QUERY, MH_id);
  wait(positive response from any MSS_k);
  location(MH_id) ← response.location;
  if no positive response
     send(broadcast, QUERY, MH_id);
}
```

6.2 Description

Virtual Identity Determination: The procedure *assign_virtual_ids* takes the identity of a hot mobile host as the input, and assigns at most two virtual identities to that mobile host. These virtual identities are stored in the set $VMH_id(MH_id)$. If x hot mobile hosts have already been assigned two virtual identities each, then any other mobile host that now becomes hot will be assigned only one virtual identity, which is equal to $MH_id\ + x$: its *native* virtual identity.

State Transitions: hot and cold: When a mobile host makes a transition from a hot state to a cold state, the procedure *transition_hot_to_cold* is invoked. As a result, values of the hash function are computed with respect to the MSS in which the mobile host is present, and each of the virtual identities of the mobile host, except for the *native* virtual identity of the mobile host: $MH_id\ + x$. The result of each such computation (an integer j) indicates a set S_j of MSSs that store the location information about the mobile host. All these MSSs are requested to delete their information about the location of the mobile host. As a result, only one set of MSSs, corresponding to the pair (MSS, $MH_id{+}x$), maintain location information about the mobile host. Thus, the number of MSSs that maintain location information about a cold mobile host is less than the number of MSSs that maintain the corresponding information about a hot mobile host.

The converse is true when a mobile host makes a transition from a cold to a hot state. The procedure *transition_cold_to_hot* is invoked. Multiples sets of MSSs (at most two sets in the algorithm stated above) are requested to store location information about the hot mobile host. The reason for having a greater number of MSSs store location information about a hot mobile host is as follows: the location of a hot mobile host is likely to be queried by a larger number of nodes (both mobile and static), and also more often, than the location of a cold mobile host. Also, these querying nodes are likely to be located in different parts of the network. By having more MSSs store the location information, the chances of a querying node finding the required information at a nearby MSS are higher. So, the response time of location queries is lower. Hence, the cost of locating a hot mobile host is expected to be less than the cost of locating a cold mobile host.

Location Update: When a mobile host moves from one cell to another, its location has to be updated at the appropriate MSSs that act as the distributed location servers. The choice of the update strategy (time-based, number of movements-based, or distance-based) is orthogonal to the *location_update* procedure. The parameter *old_MSS* denotes the MSS of the cell in which the mobile host was resident when the last location update was done. The current cell's MSS is called the *new_MSS*. Out-dated location information is deleted from all the MSSs that were the location servers with respect to the (*old_MSS*, MH_id) pair. New location information is stored at the MSSs that are determined to be the location servers, by the hash function, with respect to the (*new_MSS*, MH_id) pair. These MSSs correspond to the *write set* for location information. The state of the mobile host (hot or cold) is implicitly accounted for, in the location update procedure, as the hash function is computed for each element in the virtual identity set (VMH_id) of the mobile host.

Location Query: A static node, say MSS, or a mobile host in the cell corresponding to MSS, wishing to communicate with a target mobile host first needs to know the location of

the target. Let the target mobile host's identity be MH_id. To locate the target, the function *locate_MH* is invoked. First, MSS searches its cache for MH_id's entry. If such an entry is found, the corresponding mobile service station, MSS_i, is probed to determine if MH_id is still in the same cell. If so, MSS_i returns its own location in the response. Otherwise, one of the virtual identities of MH_id is arbitrarily selected. This virtual identity is used by the hash function to determine the set of MSSs that should be queried about MH_id's location (the *read set* for location information). If a queried mobile service station, MSS_i, has the location of MH_id in its directory, it is sent in the response. If no queried mobile service station has the location of MH_id, the query is broadcast over the network. Once MSS receives the location of the cell in which MH_id is present, the message is sent over the fixed wire network to the corresponding mobile service station. If MH_id has moved out of the cell since the last location update, a sequence of forwarding pointers (depending on the path taken by MH_id since it moved out of MSS_i's cell) is followed to the cell in which MH_id is currently present.

Selection of Arbitrary Virtual Identity: As mentioned above, in the procedure *locate_MH*, one of the virtual identities of the target MH is arbitrarily selected. Such a selection does not require each MSS to store the virtual identities of all MHs. That would impose prohibitive storage overheads. In the simplest implementation, the virtual identity selected is the *native* virtual identity of the MH, i.e., $MH_id\ + x$. Alternatively, the global array *assigned[0..x-1]* could be an array of 2-tuples of the form (boolean, MH_id). When a hot mobile host with identity MH_id is assigned a virtual identity i ($0 \leq i < x$), assigned[i] is set to (TRUE, MH_id). Therefore, to find all the possible virtual identities of an MH, and then arbitrarily select one, the MSS only has to scan through the *assigned* array of x tuples. This array can be stored at a central site, or replicated at several sites for fault-tolerance.

Simultaneous State Transition and Migration: Out-dated location information may be left at some MSSs if location update and state transition have to be done at the same time. On state transitions, the *add* and *delete* messages should be sent to nodes determined with respect to the (*new_MSS*, MH_id) and (*old_MSS*, MH_id) pairs, respectively. However, the only mobile service station parameter in the transition procedures is the new MSS. So, during transitions from a hot to a cold state, that happen concurrently with location changes, *delete* messages may be sent to some MSSs that did not store the mobile host's location, while some MSSs that did store old location information about the mobile host may not receive such a message. To avoid such situations, first the location update should be done, followed by the execution of the state transition procedure. By doing this, it is ensured that out-dated location information is deleted from all the MSSs before any new location information is stored.

7 Read and Write Set Construction

The performance of the location management and tracking scheme depends on the construction of the *read* and *write* sets (quorums), referred to as S_i, S_j, etc. in Sections 5 and 6. Let the total number of MSSs in the network be m. In order to minimize the communication overhead of the algorithm, it is necessary that the size of each quorum S_i be kept as small as possible. Also, to distribute the responsibility of storing the distributed location directory

in a fair manner among the MSSs, the quorums should be symmetric. One approach to constructing such quorums is proposed in [6], and is described below:

Iterative approach: Initially, the quorums are trivially constructed, each containing just over $m/2$ MSSs whose identities form a contiguous sequence. An iterative reduction of the quorum size is done. In each iteration, the quorum (sequence of MSSs) is partitioned into three partitions of roughly the same size, and the middle partition is discarded. The other two partitions are further reduced in the next iteration. A set of MSSs is not partitioned any further if its size plus one is not divisible by three, or its size is less than seven. This leads to the generation of symmetric quorums of size $0.97m^{0.63}$.

Another approach to constructing quorums, similar to that proposed in [5] in the context of the mutual exclusion problem, is as follows:

Grid based scheme: Let $m = l^2$ for an integer l. Then an $l \times l$ grid is constructed. The grid points are numbered from 0 to $m-1$. The quorum S_i consists of the grid points on the row, and the grid points on the column passing through point i. Thus, the cardinality of each quorum S_i is equal to $2\sqrt{m}-1$. If m is not a square of an integer, a degenerate grid can be constructed with the outermost row/column being reduced in size. In such a case, while constructing S_i, the partial row/column is complemented using grid points from another row/column. Therefore, the size of each *read* and *write* set is $O(\sqrt{m})$, where m is the number of MSS in the network.

8 Handling Hot Mobile Hosts

As described in Section 6, mobile hosts whose location is queried often have at least one, and often two *write* sets associated with them. So, the average distance between a node querying the location of the hot MH and an MSS containing the MH's location is less than the situation if the location information was stored at MSSs belonging to only one set.

A querying node probes MSSs belonging to only one of its two possible *read* sets for the hot mobile host. There are two possible cases:

1. Let an MSS belong to both the *read* sets corresponding to a (querying node, hot MH) pair. In that case, to the MSS the hot MH appears to be equivalent to two cold virtual MHs (on account of the two virtual identities of the hot MH). By the symmetry property of *read* and *write* set construction, and the fact that the hashing function is uniformly distributed over all the MSSs, it is implied that this MSS stores location information of fewer MHs than an MSS that stores location information only for cold nodes.

2. Let an MSS belong to only one of the two *read* sets. In that case, as query messages are sent to only one arbitrarily selected *read* set, this MSS will, on an average, receive only half the location queries for the hot MH.

Thus, an MSS storing location information about a hot MH either stores location information about fewer MHs, and/or receives only half the queries corresponding to the hot MH. Hence, open addressing double hashing along with virtual MH identities ensures that:

- location information about mobile hosts is fairly distributed among all the MSSs constituting the distributed location directory, irrespective of the geographical distribution of the mobile hosts.

- the number of location queries received by each participating MSS is fairly distributed regardless of whether it stores location information about only hot MHs, or only cold MHs, or a combination of hot and cold MHs.

The performance can be further optimized if the hot nodes were not restricted to having only two virtual identities. Instead, the number of virtual identities of an MH could vary from 1 to n, for an integer n. The more frequently an MH is queried, the greater the number of virtual identities assigned to it.

The determination of whether an MH is hot or cold is done by the MSS of the cell in which the MH is present. If at time t, the number of queries per unit time received by the MSS on behalf of the MH, from nodes outside the cell, averaged over the interval $[t-T, t]$, exceeds a pre-determined threshold, the MH is hot. Otherwise, it is cold. T is a pre-selected constant. A small value of T makes the state transitions very sensitive to fluctuations in query rates. A large value of T, on the other hand, reduces the sensitivity, but also avoids state changes due to short lived aberrations in query patterns.

9 Computation, Communication, and Storage Overheads

The location management and tracking algorithm imposes low computation and communication overheads. In order to assign virtual identities to a hot mobile host, the *assigned* array of x elements has to be traversed at most once. Hence, the computation complexity of this operation is $O(x)$.

The auxiliary hash functions h_1 and h_2 usually employ a small, constant number of integer multiplication and division operations. Hence, determination of sets of MSSs (quorums) to which location *add* or *delete* messages should be sent is computationally inexpensive. There are $O(\sqrt{m})$ or $O(m^{0.63})$ elements in each quorum depending on the quorum construction strategy employed, where m is the total number of MSSs in the system. Typically, m, the number of MSSs, is a very small fraction of the total number of nodes in the network.

In the state transition procedures, *add* or *delete* messages are sent to at most one quorum of MSSs. In the location update procedure, *add* and *delete* messages are sent to at most two quorums of MSSs, each. When the location of a mobile host is to be determined, *query* messages are sent to only one quorum of MSSs. Hence, the message complexity of each directory operation is $O(\sqrt{m})$ if the grid based scheme is used, and $O(m^{0.63})$ if the iterative approach is employed. Each message has a small size, consisting of at most two MSS identities, one MH identity, and a flag indicating the nature of the message. Moreover, these messages are transmitted in the fixed wire network, which has a much higher bandwidth than the wireless MH-MSS links.

Assuming that x is a constant value and the number of mobile hosts in the network is M, each MSS has to store location information about $O(M/\sqrt{m})$ mobile hosts for the grid

based approach. This is because the location information of each of the M mobile hosts is replicated $O(\sqrt{m})$ times in the directory, and the entire directory is distributed uniformly over m MSSs acting as location servers. Each of these $O(M/\sqrt{m})$ pieces of information contains the MSS_id for the cell in which the mobile host is present, which requires $\lg(m)$ bits. Besides this directory information, each MSS has to store an extra $O(m\sqrt{m})$ entries, each of size $\lg(m)$ bits, indicating memberships of all the *read* and *write* sets: there are at most m *read* and *write* sets, each with a cardinality of $O(\sqrt{m})$.

10 Conclusion

The popularity of mobile computing is on the rise. As mobile computing systems mature, the number of mobile hosts in a network is bound to increase at a rapid pace. In order to keep track of these hosts, and to be able to interact with them, it is important to store and update their location information efficiently.

In this paper, we presented a distributed dynamic location management and tracking scheme. Location information about mobile hosts is replicated at $O(\sqrt{m})$ MSSs, where m is the total number of MSSs in the system. So, not all MSSs need to store the location of every mobile host. Mobile hosts that are queried more often than others have their location information stored at a greater number of MSSs. The set of MSSs that store a mobile host's location change dynamically as the host moves from one part of the network to another. Also, MSSs that store location information of frequently queried mobile hosts store information about fewer hosts than the MSSs that only store location information of infrequently queried mobile hosts. As a result, the location directory is fairly distributed throughout the network, and no single MSS is overburdened with the responsibility of responding to location queries.

The distributed location management scheme imposes low computation, communication, and storage overheads. Moreover, mobile hosts and the wireless links do not incur any of these overheads, which is a desirable feature as they are usually resource poor. The overheads are visible to the MSSs and the fixed wireline network, which are comparatively resource rich.

References

[1] B. Awerbuch and D. Peleg. Online Tracking of Mobile Users. *Journal of the Association for Computing Machinery*, 42(5):1021–1058, September 1995.

[2] A. Bar-Noy and I. Kessler. Tracking Mobile Users in Wireless Communication Networks. In *Proceedings of IEEE INFOCOM*, pages 1232–1239, 1993.

[3] A. Bar-Noy, I. Kessler, and M. Sidi. Mobile Users: To Update or not to Update? In *Proceedings of IEEE INFOCOM*, pages 570–576, 1994.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press–McGraw-Hill Book Company, 1990.

[5] M. Maekawa. A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. *ACM Transactions on Computer Systems*, pages 145–159, May 1985.

[6] W. K. Ng and C. V. Ravishankar. Coterie Templates: A New Quorum Construction Method. In *Proceedings of the* 15th *International Conference on Distributed Computing Systems*, pages 92–99, May 1995.

[7] S. Rajagopalan and B. R. Badrinath. An Adaptive Location Management Strategy for Mobile IP. In *Proceedings of First ACM Mobicom 95*, November 1995.