Modified Tree Structure for Location Management in Mobile Environments*

Shlomi Dolev† Dhiraj K. Pradhan‡ Jennifer L. Welch§

Abstract

In this paper we suggest a new data structure for location management in mobile networks. The data structure is based on the tree location database structure. We suggest replacing the root and some of the higher levels of the tree with another structure that balances the average load of search requests. For this modification we use a *set-ary butterfly* network, which is a generalization of the well-known *k-ary butterfly*. We also suggest modifying the lowest level of the tree in order to reflect neighboring geographical regions more accurately and to support simple location data management. The modification of the lowest level also supports simple handoffs.

The update of the proposed location database ensures correct location data following any number of transient faults that corrupt the location database information and thus is *self-stabilizing*.

Keywords: Location management, distributed algorithms, fault-tolerance, self-stabilization, handoff management.

*An extended abstract of this work was presented in the Fourteenth Annual Joint Conference of IEEE Computer and Communications Societies, INFOCOM'95.

†Department of Mathematics and Computer Science, Ben-Gurion University, Beer-Sheva, 84105, Israel, Supported in part by TAMU Engineering Excellence funds and NSF Presidential Young Investigator Award CCR-91-58478.

‡Department of Computer Science, Texas A&M University, College Station, TX 77843.

§Department of Computer Science, Texas A&M University, College Station, TX 77843. Supported in part by TAMU Engineering Excellence funds and NSF Presidential Young Investigator Award CCR-91-58478.

1 Introduction

Mobile computers using wireless communication is a rapidly expanding reality in computer systems. This trend grows with the availability of powerful portable computers and the construction of infrastructure for wireless communication used by cellular phones. In the near future, new computer services will accompany us wherever we are: mail, local news, local information, etc. (See, e.g., [9, 12, 11].)

Location management is an important task for mobile systems. Whenever a connection is to be established with a mobile computer, knowledge of the location of this mobile computer is required. Mobile systems have to cope with frequent location updates and inquiries. Thus the distributed database used for location management, as well as the update and search (inquiry) procedures, have great influence on the performance of the system. The location database is maintained by location servers which are located in a physical wired network of fixed location hosts. Any fixed host may act as a location server, i.e., have a process dedicated to maintaining the location database and answering inquiries concerning the location of a mobile host. The task of the location servers is to update the location database whenever a mobile host decides to relocate, and to answer location inquiries.

A solution that uses a centralized location database (one that is maintained in a single site by a single location server) implies a severe bottleneck in the performance, since every update or search is executed by this single location server. Moreover, the cost (i.e. number of operations in the database) of an update for a relocation is fixed and does not depend on the distance between the previous and current locations. For example, a movement within a city and a movement to a new continent are treated the same, yet many movement are likely to take place within a city, while movements to another continent are rare.

One suggestion for a distributed location database is a tree structure of location servers (e.g., [9, 10]). The leaves of the tree are *mobile support stations*. Each mobile support station is responsible for the communication with mobile hosts within a small geographic region called a *cell*. Each mobile support station maintains a list of the mobile hosts within its cell. Several mobile support stations are connected to a single location server, which is their parent in the tree structure that is embedded in the fixed network for maintaining location information. Every non-leaf location server, s_i, has a list, $list_j$, of the mobile hosts for every j'th child of s_i. $list_j$ contains the identifiers of the mobile hosts that are in the cells of the leaves of the subtree rooted at s_j. The tree structure of location servers is both distributed and hierarchical and hence fits better than the centralized solution discussed above.

Another location management scheme is based on *home location servers* (e.g., [9]). In this scheme, each mobile host is registered in a fixed home location server located in the fixed network. The current location of the mobile host is maintained at this home location server. Whenever a mobile host, m, moves from one cell to another (or after some constant number of moves when forwarding pointers are used), the address in the fixed home location server has to be updated. This approach might be efficient when the mobile host is almost always close to its home location server. However, for the cases of high mobility and when m is far from its home location server,

many long distance updates have to take place. Moreover, in some cases a search inquiry is for a mobile host with special attributes within a geographic region, e.g., search for a medical doctor within a short distance. The tree structure naturally supports such inquiries. An inquiry message may be sent directly to the location server that is a root of a subtree including every mobile support station in some geographical region. In contrast, for the home location server it is hard to find a medical doctor that is present in the region but has a distant home location server.

This paper investigates the properties of the tree structure and suggests a modified "tree" structure for location management. We define the *level* of a node s in a tree to be the number of tree links in the path from the root of the tree to s, e.g. the level of the root is 0. The first observation made is related to the upper levels of the tree structure — where *upper* (*lower*) levels are the levels close to (far from, respectively) the root. An initiator of a location inquiry for a highly mobile host is uncertain about the location of the mobile host. The tree structure is hierarchical: a node in level i has information regarding more mobile hosts than a node in level $i+1$. Naturally, a location inquiry for highly mobile host is initiated by sending an inquiry message to the root of the location servers (or to a location server with relatively upper level in the tree). This fact implies a heavy load on the root of the location server tree, which in turn implies a bottleneck on the performance of the location servers. For example, a search for a mobile host within Manhattan may start with the root of the subtree of the location servers that maintain the location data of all the mobile hosts in Manhattan. Since the number of mobile hosts in such a dense area may be big, this location server will be overloaded with inquiries. We suggest a new tree structure for such cases that preserves the benefits of the tree and balances the inquiry load. Interestingly, the solution is based on the well known k-*ary* butterfly network (cf. [14]).

The second observation is related to the lower levels of the tree structure. The leaves of the tree correspond to geographic regions. Movements between neighboring geographic regions have high probability of happening. In order to minimize the location updates, one would like to guarantee that any two neighbors will have the same parent in the tree; however this would result in there being a single parent for all leaves in the tree, which is a non-hierarchical structure. A tree structure with $h > 1$ levels may not be symmetric in the following sense: some neighbors have a common parent while others do not. In fact, there can be leaves corresponding to neighboring geographic regions whose only common ancestor is the root of the tree. Thus, the maximal delay for completing a location update depends on the number of levels in the tree. An inquiry initiated during such a slow location update might not result in the address of the mobile host.

Some techniques of using *forwarding pointers*, instead of updating the location servers' data, have been proposed to cope with the delay in the updates (e.g., [8, 2]). The pointer chain is extended whenever the mobile host moves, until at some point an update of the hierarchical database is executed and the pointers are eliminated. Pointers solve the problem of the long delay for updates but introduce new problems—longer search, pointer management including periodic updates in order to eliminate too long pointers chains. Moreover, it is hard to recover following corruption of a pointer (due to a transient fault, say crash and recovery of a location

server). We propose to use the fact that mobile hosts move from neighbor to neighbor to ensure fast updates without the use of pointers.

We suggest adding connections to the lowest level of the tree such that any two location servers that correspond to neighboring geographic regions will have either a common parent or a direct link connecting them. Note that when the regions are hexagons, at most six additional connections per leaf are required. We show that the modified lowest level of the tree solves the problem of unsuccessful inquiries due to update delay.

Handoff takes place when a mobile host moves from one cell to another during a communication session. The information transmitted to the previous mobile support station is easily forwarded to the new mobile support station through their common link. Note that handoffs and location management serve different purposes—handoff is not required unless a communication session is in progress while a mobile host moves from one cell to another, whereas location management is always required. We show how *handoff* executions are improved with the lower level modification. The improvement is by the simplicity of executing *local broadcast* (of distance i) in the presence of the additional links. *Local broadcast* of distance i is a broadcast initiated by a mobile support station that reaches every mobile host within cells of distance i from itself.

A *self-stabilizing* system [5] can be started in an arbitrary initial state and regains its consistency by itself. A self-stabilizing system can recover from *transient faults*, faults which change the state of one or more components of the system. The self-stabilization property is very important for on-going tasks such as topology update [17, 6]. We view the location management task for mobile systems as a topology update under special settings — a subset of the system components (the mobile hosts) change their location frequently. Our goal is a self-stabilizing location management scheme that copes with transient faults. In other words a self-stabilizing location database must guarantee that starting with *any* correct or incorrect location data for the mobile hosts (resulting from arbitrary transient faults, e.g., message loss, memory corruption) eventually each mobile host can be found using the location database.

We present a location management scheme that does not use forwarding pointer. The self-stabilization property is achieved easily when these forwarding pointers are eliminated. Beyond the self-stabilizing property, the modifications of the upper and lower levels of the tree also result in a more robust mobile system that can tolerate location server crashes.

In the next section we describe in detail the distributed mobile environment. In Section 3 we present the modifications for the upper levels and lowest level of the tree. In Section 4 we present a self-stabilizing location management. Concluding remarks are in Section 5.

2 Distributed Mobile Environment

A mobile network is composed of a *fixed network* and a *wireless network* that interact with each other. The fixed network is a standard point-to-point communication network in which the communication between *fixed* location hosts is carried by physical wire links. Some of the fixed hosts are equipped with wireless transceivers. These fixed hosts are called *mobile support*

stations (or *base stations*). Each mobile support station is capable of transmitting and receiving messages from a limited geographical region around it. This region is called the *cell* of the mobile support station. Thus, the geographic area in which the wireless network service exists is partitioned into cells and a mobile support station is located in every such cell.

The wireless network consists of *mobile hosts* which have the capability to exchange messages with a *mobile support station*. The mobile host can communicate with a mobile support station that is within a short distance from itself. The mobile host may move from one cell to another while communicating with the fixed network (or with another mobile host through the fixed network). At every instance the communication is carried by the mobile support station of the cell in which the mobile host is currently located.

Data on the location of mobile host is maintained in the fixed network by *location servers*. Any fixed host may act as a location server, i.e., have a process dedicated for maintaining the location database and answering inquiries concerning the location of a mobile host. In particular, mobile support stations may also act as location servers. The precise characterization of the location database and the update and search procedures is up to the system designer. Our focus in this paper is an hierarchical location management strategy that is based on a tree structure of location servers which we now describe. (The idea of using a tree structure has been proposed in, e.g., [9, 10]; in this paper we suggest specific schemes for update and search).

Each mobile support station is also a location server that maintains a list of mobile hosts within its cell. Whenever a mobile host joins a cell, it is included in the list; whenever a mobile host leaves the cell, it is removed from the list. Every mobile host periodically sends an *alive* signal to the mobile support station. If the signal of a mobile host listed in some mobile support station does not reach the mobile support station for a predefined period of time, the mobile support station assumes that the mobile host is not (active) in the cell and removes it from the list.

The location servers of the mobile support stations are connected with other location servers within the fixed network. The location servers are organized in a tree structure, where the location servers of mobile support stations are the leaves of the tree. Every Δ leaves are connected to a single parent location server. A non-leaf location server, s_i, has Δ subtrees (each rooted in one of its children). There is a single *root* location server that does not have a parent, while every other location server has a single parent. s_i maintains Δ lists; the j'th list, $list_j$, is associated with the j'th child of s_i. $list_j$ is the list of the mobile hosts that are in the cells of the leaves of the subtree rooted at this j'th child.

We now present the outline of the update and search procedures that we use for the tree structure. These procedures are the base for the update and search procedures of the modified tree structure. The update procedure is invoked whenever a mobile support station receives an indication that a mobile host joined or left the cell for which the mobile support station is responsible. The indication for a join event of a mobile host m (that is not listed in the mobile support station list) is an alive signal from m that is directed to the mobile support station. The indication for a leave event of m is the absence of an alive signal for a long period (the length

of the period is a function of how frequently alive signals are sent and the time required for an update, as discussed in the sequel).

When a mobile host m joins a cell, say c_1, possibly due to movement from another cell c_2, m's identifier is added to the list of the mobile support station of c_1. This addition is reported to the mobile support station's parent by sending a *join* message with m's identifier. When a non-leaf location server, s_i, receives a *join* message from its child, s_j, for a mobile host, m, that does not exist in any of s_i's lists, then s_i includes m in $list_j$ and (when s_i is not the root s_i) sends a *join* message to s_i's parent. Otherwise, when s_i finds m's identifier in one of its lists, say $list_k$, s_i (1) deletes the identifier of m from $list_k$, (2) sends a *delete* message to its k'th child and, (3) adds the identifier of m to $list_j$. Note that in this case no *join* message is sent.

Note that if m were in c_2 just before m joins c_1, then m's identifier appears in every $list_j$ of every s_i such that the link between s_i and s_i's j'th child is in the path from the root to c_2. After m moves to c_1, the update procedure ensures that m's identifier appears in every $list_j$ of every s_i such that the link between s_i and s_i's j'th child is in the path from the root to c_1. Further note that for every location server s_i, an identifier of a mobile host m may appear in at most one list of s_i, thus at any given time there is at most one path from the root to a mobile support station such that m appears in every list along this path. Still the update procedure needs to "collect garbage", i.e., eliminate appearances of m's identifier that are not related to the path from the root, e.g., the part of the previous path to c_2 that is not part of the path to c_1. Obviously when a location server s_i deletes m's identifier from $list_k$ and at the same time adds m's identifier to $list_j$, s_i may send a *delete* message towards c_2 that removes the identifier of m in the path from s_i towards the mobile support station of c_2.

A search inquiry for the location of a mobile host m starts with a message sent by an initiator (fixed or mobile) host to the root of the location server tree. When such an inquiry message arrives at a location server s, s searches for m in its lists. If s is not a leaf and m is found in $list_j$ then s sends the inquiry message to its j'th child. This procedure repeats itself until the inquiry message reaches a mobile support station. The mobile support station verifies that m is in its cell. Upon successful verification, the address of the mobile support station is sent to the host that initiated the inquiry.

3 The Modified Tree

In the next two subsections we present the modification we propose for the upper and lower levels. Then we present the update and search procedures for a tree with modified upper and lower levels. For simplicity we assume that every location inquiry arrives at the root of the location server tree. In reality a location inquiry for mobile host m might be addressed to a root of a subtree (say the one that is responsible for New York City) in case the initiator is sure of the area m is in. The approach we suggest can be applied to subtrees as well.

3.1 Modifying the Upper Levels

In this subsection we present an alternative structure for the upper levels of the tree, i.e., the levels that are closer to the root. The new structure of the upper levels improves the performance of the tree structure by spreading the load of queries. To measure this improvement, we suggest the following analysis. Let l_o be the average load (queries per time unit) of outside queries that reach the root of the tree. Let Δ be the out-degree of a node in the tree. First we assume that the out-degree of every non-leaf node in the tree is Δ; later we relax this assumption. We assume that the load of the queries that reaches every node is evenly divided between its children. Thus, the average load for a node in level 1 is l_o/Δ. Similarly the average load per node in level i is l_o/Δ^i. Let h be the level of the leaves of a tree T. We present a modified tree structure for T in which the maximal average load per node is l_o/Δ^i, for every $1 \leq i \leq h$. The modification requires $i\Delta^i - ((\Delta^i - 1)/(\Delta - 1))$ additional nodes, which is proven necessary.

Interestingly the modified upper levels for a tree with $\Delta = 2$ is the well-known butterfly network; Figure 3 illustrates the correspondence between the binary tree structure and the butterfly. For out-degree $\Delta > 2$ the modified upper levels form a *k-ary butterfly* (cf. [14]). In Figure 4 we show that the k-ary butterfly is a composition of trees with out-degree $\Delta = k$. The example in Figure 4 is for the case $k = 3$.

The definition of the modified tree is naturally expanded for trees with uniform out-degree at each level and different degrees in different levels. We generalize the k-ary butterfly to a *set-ary butterfly* as depicted in Figure 5. This generalization may reflect better the current structure of the location server tree which might have different out-degrees in different levels of the tree.

Note that Figures 3 through 5 present *only* the upper levels of the tree while the rest of the tree is left unchanged. The number of upper levels of the tree to be modified is a function of the desired distribution of the inquiry load. If the system designer must guarantee that any location server has average load of no more than the average load of a node in level $i + 1$ then i levels of the tree have to be modified.

The h upper levels of a tree form a tree T_h with h levels. To modify T_h we use the following construction. Let T_i be a subtree of T_h with depth i such that the leaves of T_i are leaves of T_h. The modification starts with trees of depth one, i.e., $h = 1$. T_1 is modified to MT_1. T_1 contains a single root and Δ_1 leaves which are connected to this root. The construction replaces the root with Δ_1 "root" nodes and connects each such node with all of the leaves, i.e., each such node has Δ_1 outgoing links. The construction of MT_{i+1} is defined recursively by MT_i and the out degree Δ_{i+1} of a root of a subtree T_{i+1}. Let n_i be the number of roots in a MT_i. The root of T_{i+1} is replaced by $\Delta_{i+1}n_i$ roots. Each root of MT_{i+1} is connected with a single root of each of the Δ_{i+1} MT_i's. If $r_{i+1,k}$ is the k'th root of MT_{i+1} then $r_{i+1,k}$ is connected to the $(k-1)$ mod $n_i + 1$ root of each of the Δ_{i+1} MT_i's. The modified tree structure supports the same search procedure as the original tree. To balance the load of outside inquiries we assume that an inquiry is sent to one of the "roots" of the modified tree that is chosen according to some portion of the identifier of the mobile host m that is to be located. Thus each root of the modified tree has to maintain the location information (in the lists) for only some portion of the mobile hosts. This in turn

keeps the number of update messages small—only the embedded tree for the mobile host for which the location update message has been sent is involved in the update.

The division of geographic regions into cells has great influence on the amount of processing needed for location updates. For example, it is important for the mobile support stations that cover Japan to have a common ancestor that is relatively close to the leaves of the tree. On the other hand, the only common location server for mobile support stations in Japan and, say, England, could be the root. With such a division the number of update messages arriving at the root is small. Therefore we also suggest that in some cases it is better to send a copy of the update messages to every root (and not only to a single root). The benefits of having identical data in each root is the possibility to place each root in a different (fixed) location. Each root will handle inquiries from its surrounding area. For example, one replicated root may be located on the west coast and another on the east coast, while both roots maintain the location data for every mobile host. If the inquiry initiator is on the west (respectively, east) coast, it starts the inquiry with a message to the root on the west (respectively, east) coast.

The next Theorem states the number of nodes (location servers) added by our construction is minimal. The Theorem is proved for the case of fixed Δ tree, similar result can be obtained for a tree with different Δ's in different levels. We compare our modified tree structure MT, that corresponds to a tree T, with any other solution G such that (1) each node in G corresponds to a single tree level in T, (2) there exists at least one node in G that corresponds to the level of the root in T, and (3) for each node r in G that corresponds to the level of the root in T, r is a root of an embedded tree in G with the same structure as T. We use l_o to denote the (original) average load of inquiries on the root of T, and l_m to denote the maximal average inquiries load on a node in the modified tree. (Note that it is possible for a solution G to have the maximal average load on a non-root node, e.g., if the tree is a binary tree and the root is replicated three times while the rest of the tree is left unchanged.)

Theorem 3.1 *The number of nodes in any G such that $l_m \leq l_o/\Delta^i$ is greater than the number of nodes in T by at least $i\Delta^i - ((\Delta^i - 1)/(\Delta - 1))$.*

Proof: The number of nodes (location servers) in every level must be at least Δ^i. The j'th level, $0 \leq j < i$ has only $\Delta^j < \Delta^i$ nodes. The number of nodes to be added is $\sum_{j=0}^{i-1}(\Delta^i - \Delta^j)$. This implies the theorem. ■

3.2 Modifying the Lower Levels

The primary motivation for the use of the hierarchical tree structure was to localize the effect of "short" moves of mobile hosts—ideally a small portion of the location database is updated upon such a "short" move. However the tree is an asymmetric structure—there are some mobile support stations corresponding to neighboring cells that have a common parent in the location server tree, yet there are other mobile support stations corresponding to neighboring cells which have only one common ancestor in the tree, namely the root. Thus, when a mobile host moves

from one cell to a neighboring cell it causes in one extreme only the common parent of the neighboring mobile support stations to be updated, while in the other extreme some location server at every level of the tree has to be updated.

The maximal delay for completing a location update is related to the number of levels of the tree. An inquiry initiated during such a slow location update might not find the address of the mobile host. One possibility to cope with this problem is to delay the update and use *forwarding pointers* to keep track of the location of the mobile host. When a mobile host m moves from one cell c_1 to a neighboring cell c_2 the original database is not updated; instead the mobile support station of c_1 has a pointer for m that points to the mobile support station of c_2. If m subsequently moves to another cell, say c_3, c_2 will have a pointer for m that points to the mobile support station c_3, and so on, for further moves. In order to avoid searches through too long chains of pointers, updates of the tree location database are done periodically and the pointers are eliminated. Thus, in the long run forwarding pointers cause overhead in time and space.

We say that two cells c_1 and c_2 are at *distance* i from each other when a move of a mobile host from c_1 to c_2 must physically pass through at least i cells (excluding c_1). A broadcast from a cell c_1 to every cell of distance less than or equal to r from c_1 is called *broadcast with diameter* $2r$. We assume that during the time it takes for a mobile host m to move from a cell c_1 to a cell c_2 at distance i, several complete updates can be executed. The mobile host triggers location updates by sending an alive message to the mobile support station of its current cell. To avoid too frequent location updates and for simplicity of the updates, we assume that the time between every two successive alive messages of a mobile host is long enough to guarantee completion of the location update. The required broadcast diameter for the search procedure, denoted d, depends on the time needed for completion of an update, the size of the cell, and the velocity of the mobile host.

To locate a mobile support station m at any given time, it is enough to communicate with the cell for which the tree location database has a record and the neighboring cells within distance d from this cell. This could be easily done when any two neighboring mobile support stations are connected by either a direct physical communication link or a fast virtual communication link. When a search for a mobile support station m arrives at a location server s_i that is a leaf in the tree (mobile support station) and s_i detects that m is not currently in its cell, s_i broadcasts the search request to cells in distance d. Then s_i receives an answer (found or not-found) from its neighbors and accordingly sends an answering message to the initiator of the search. Note that for this scheme to work, s_i must not delete the location information concerning m for long enough time. The indication of disappearance is delayed for a period of time that ensures the completion of the next update.

Obviously, the additional (relatively short) links between neighboring location servers will result in efficient local broadcasts and will support handoffs. Note that the location management for a mobile host that moves from one cell to another takes place automatically based on the periodic alive messages sent to the closest mobile support station and does not require resources for fast response as handoff does. The modification of the lower level may support fast response

required for handoffs.

We call the following handoff procedure *follow-me handoff*. The follow-me handoff procedure does not use forwarding pointers. If during a communication session every packet addressed to a mobile host is locally broadcast, then even if the mobile host moves to a new cell, the mobile host receives every packet sent to it. At the same time the mobile host notifies its session partner of its new location through the new mobile support station. Note that this notification can be appended to the packets of the communication session. Consequently, the session partner starts sending its packet to the new mobile support station. Thus, after a connection is established the hosts of the session do not need to access the location database.

3.3 Update and Search Procedures

The correct operation of our update and search procedures relies on the following **assumptions**:

(A1) The location database is initiated with all the lists empty.

(A2) The time required to complete an update (send a join message towards the direction of the root and then send a delete message towards the previous cell) is bounded by t_u.

(A3) The time between two successive alive signals of a mobile host m is $t_a > t_u$.

(A4) If mobile host m sends an alive signal at time t while m is in cell c_i, then at time $t + t_a + t_u$ m is in a cell c_j (possibly $c_i = c_j$) that is within distance d from c_i.

(A5) The LocalSearch procedure (as described in the sequel) locally broadcasts the search inquiry to all the cells that are within distance d.

(A6) Disappearance of a mobile host from a cell is indicated by the mobile support station of that cell when no alive signal is received for $t_d > t_a + t_u$ period of time.

A mobile host m is *active* if (1) m sent an alive signal, a_l, in the last period of t_a time, and (2) either m sent the previous alive signal, a_{l-1}, t_a time before m sent a_l or at least t_u time has elapsed from the time m sent a_l. Roughly speaking, the two conditions above make sure that a complete update that is related to the previous or the current location has been executed. The **requirement** for our location management scheme is that any search for an *active* mobile host m will result in communicating with m.

The code of the update and search procedures of a location server s_i in the modified tree appears in Figure 1. The code starts with the description of the actions taken by a mobile support station (a leaf in the location server tree) upon an indication of the appearance or disappearance of a mobile host m. Note that a mobile support station has a single *list* that includes the current mobile hosts in the cell. Note further that the removal of m from *list* (of a leaf) is only for "garbage collection" reasons since a *join* message to another cell results in changing the path to m in the location server tree. In order to ensure that every search succeeds,

it is assumed that the indication of disappearance is delayed for at least the time it takes for an update on the new location to be triggered by an alive signal and then to be completed.

The code continues with the actions taken by a non-leaf location server upon the receipt of *join* and *delete* messages. A *join* message for a mobile host m received from the j'th child causes the addition of m to $list_j$. In case m appears in another list of s_i, say $list_k$, then m is deleted from $list_k$ and a *delete* message is sent to the k'th child. Note that *join* messages are sent only towards the roots while *delete* messages are sent only towards the leaves.

In the modified tree a node may have more than a single parent. Thus, we have to define whether the message is sent simultaneously to every parent or alternatively the precise parent(s) to whom the message is sent. Each root, r, of our modified tree maintains location information on a subset of the mobile hosts (possibly includes every mobile host) defined by the mobile hosts' identifier. Therefore, update messages concerning the location of a mobile host m need to be sent to the parent(s) in the embedded tree rooted at the appropriate root(s). The function parent(m) executed by location server s_i returns the parent(s) of s_i in the embedded tree that is related to the root(s), r, that maintains location information on m.

The search procedure is activated by an outside initiator that uses the tree structure to locate a mobile host. The initiator sends a message $msg = (search, m, initiator)$ to the root that maintains information on m and waits for a response with the address of the mobile support station of the current cell of m. When the search message reaches a mobile support station, s_i, then s_i tries to locate m by direct communication and then by local broadcast. In the code we use the function LocalSearch for this purpose. The result of the LocalSearch procedure is sent directly to the initiator of the search request.

Correctness Proof:
We prove that the location management requirements hold for each mobile host m. The location data for a given mobile host m might be maintained by more than one root (and hence more than a single embedded tree) when the designer decides to have more than a single replica. Although the proof is written for the case of a single embedded tree for each mobile host m, the proof holds for the case of more than a single embedded tree for a mobile host m.

Messages sent from one location server s_i to another location server s_j obey the FIFO order. Each message arrives at its destination within a bounded period of time. A location database *instance* is a vector of the lists of the location servers and the messages sent between any two location servers. Location servers execute *steps*. A step executed by location server starts with zero or one message received, then zero or more messages sent and ends with a state transition for the location server. A *run* is a (finite or infinite) sequence of instances and steps, $I_1, s_1, I_2, s_2, \cdots$ such that: (1) The application of s_i to I_i results in I_{i+1}. (2) Each step s_i is executed at time t_i. For every two steps s_i and s_j $(j > i)$, $t_{i+1} \geq t_i$. (3) The message delay time is obeyed, i.e., if a message is sent in s_i and received in s_j then $t_j - t_i$ is a (positive) time period that is no longer than the message delay time.

We prove that every instance, I, of the the location database contains a single *location-path* for each active mobile host m. A *location-path* of m in I is a list of location servers $S = s_r, s_1, \cdots s_j$

(* Update Procedure *)
(* Leaf *)
Receive indication for m in cell:
  if $m \notin list$ then
    send $msg := (join, m)$ to parent(m)
  $list := list \cup m$

Receive indication on disappearance of m from cell:
  send *msg:=(non-active,m)* to parent(m)
  $list := list - \{m\}$

(* Non-leaf *)
Receive msg from the j'th child, *msg.type=join*:
  if $\forall list_k$ $msg.m \notin list_k$ then
    send msg to parent($msg.m$)
  else $\forall list_k$ $msg.m \in list_k$ do
    $list_k := list_k - \{msg.m\}$
    send $msg := (delete, m)$ to the k'th child
  $list_j := list_j \cup msg.m$

(* Non-root *)
Receive msg from the j'th child, $msg.type = non - active$:
  if $msg.m \in list_j$ then
    send msg to parent(m)
    $list_j := list_j - \{msg.m\}$

Receive msg from parent, $msg.type = delete$:
  $\forall list_k$, $msg.m \in list_k$ do
    send msg to the k'th child
    $list_k := list_k - \{msg.m\}$

(* Search Procedure *)
Receive msg from parent, $msg.type = search$:
  if *leaf* then
    LocalSearch(m)
    if *found* then send *msg:=(address)* to *msg.initiator*
    else send *msg:=(not-found)* to *msg.initiator*
  else
    if $\exists k$, $msg.m \in list_k$ then
      send msg to the k'th child
    else send $msg := (not - found)$ to $msg.initiator$

Figure 1: Update and Search Procedures for the Modified Tree

that starts with the (appropriate) root location server s_r of the (appropriate) tree and for which s_{i+1} follows s_i in S if $list_k$ of s_i includes m and s_{i+1} is the k'th child of s_i in the location server tree. We say that $list_k$ is *on* the location-path of m. For a given instance I, an identifier m that is not in a list on the location-path of m is called *dangling*.

Theorem 3.2 *Our update and search procedures for a mobile host m fulfill the requirements for location management.*

Proof: By assumption (A3) every update is completed before the next update starts. Thus, by induction on the number of updates (using assumption (A1) as the induction base) whenever an update starts there are no dangling identifiers for m. Thus, when an update starts for a mobile host m the join message reaches a location server s_i along the location-path $S = s_r, s_1, \cdots, s_i, s_j, \cdots s_k$ of m. Then the suffix $s_j, \cdots, s_k$ is replaced by the suffix leading to the mobile support station that received the current alive signal. Then a delete message eliminates the dangling identifiers of m in the suffix $s_j, \cdots s_k$. By assumptions (A2) and (A3) this delete terminates before a new alive signal is sent. Assumptions (A4) and (A5) guarantee that an active mobile host is found. Assumption (A6) ensures that the location-path of an active mobile host is not eliminated. ■

4 Self-Stabilizing Location Management

The update and search procedures presented in Figure 1 do not guarantee self-stabilization. One reason is that every update is triggered by an indication at the mobile support station. In case a transient fault occurs, say, causing omission of an identifier of a mobile host m from the location information at a non-leaf location server, no *join* message will be sent. Thus, if m does not move to a new cell and an update takes place, m is not reachable by a search that starts in the root.

We continue by defining the self-stabilizing location management **requirement**. Starting with any (corrupted or non-corrupted) location database instance I, a self-stabilizing location management scheme ensures that from some time forward every location database instance satisfies the following: for every active mobile host, m, there is a location-path from the (appropriate) root to a mobile support station that is within distance i from m's current location.

To overcome transient faults each location server s checks periodically that a mobile host identifier does not appear in more than one list of s. If an identifier appears more than once then all but one (say, the one in the list with the largest index) are eliminated. This ensures the existence of at most one location-path for every mobile host.

An identifier m is a *static dangling identifier*, at a location server s in an instance I, if (1) m is a dangling identifier at s in I, (2) there is no *delete* message for m pending in the link from s's parent to s in I, and (3) there is no *join* message for m pending in the link from s to s's parent. Note that non static dangling identifiers are eliminated by the pending delete or join messages.

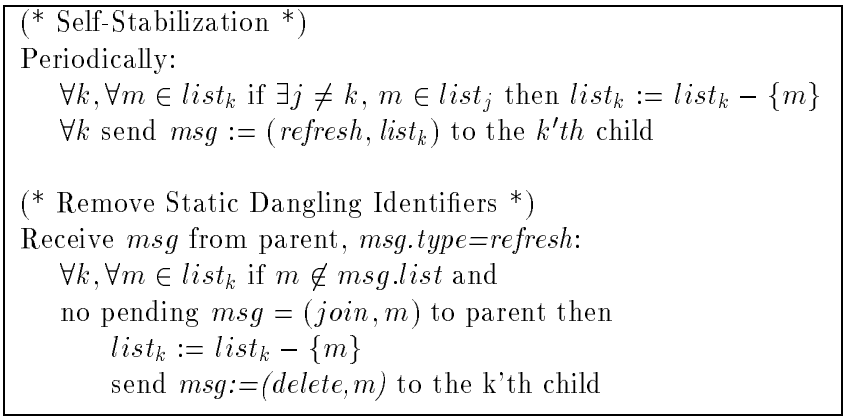

Figure 2: Extension for the Update Procedure

The existence of static dangling identifiers might result in an unsuccessful update — every *join* message reaches a static dangling identifier (or a descendant of a static dangling identifier) instead of reaching the location path. Our extension for the update procedure eliminates static dangling identifiers. Each non-leaf location server, s_i, sends the list of mobile host identifiers, $list_j$, to its j'th child. s_i's j'th child uses the list received from s_i to eliminate every static dangling identifier. For this procedure we need the following assumption:

(A7) A location server that sends a join message to its parent is able to detect whether the join message has been received (an acknowledgment mechanism may be used; note that we assume bounded delay, and hence we do not contradict [7]).

Thus, during a join update a location server s is able to discard *refresh* messages that arrive after m has been included in s's lists and before m is included in s's parent's list.

In Figure 2 we present the portion of the code that is executed periodically to ensure stability.

Correctness Proof:
We now prove that the location management with the extended update procedure is self-stabilizing. To do so we have to show that eventually a location database instance is reached such that there are no dangling identifiers. This ensures the correct update as proved in the previous section.

Lemma 4.1 *Starting from any instance of the location database that follows a finite time t_1 it holds that for each location server s_i, no mobile host identifier m appears in more than one of s_i's lists.*

Proof: A location server repeatedly verifies that each identifier appears in at most one of its lists and deletes identifiers that appear in more than one list when necessary (see the first statement in Figure 2). We now show that following such a verification (and deletion if necessary)

no addition of a mobile host identifier to a list of a location server s_i occurs unless this identifier does not appear in any other list of s_i. This is certainly true when s_i deletes an identifier from its lists. The only addition of an identifier to a list of a non-leaf location server is due to a *join* message. The proof is completed since before adding a mobile host identifier due to a *join* message any other appearance of m is deleted. ■

Note that t_1 is bounded by the maximal time it takes for every location servers to reach and execute the first statement in Figure 2.

Lemma 4.2 *Starting from any instance of the location database that follows a finite time t_2 it holds that no static dangling identifier exists.*

Proof: Let I be the location database instance following t_1. Static dangling identifiers may exist in I. We first show that no update operation increases the number of static dangling identifier. A static dangling identifier may be produced by:

1. Eliminating a mobile host identifier from one list without sending a delete message to the corresponding child location server. The updates of a non-leaf location server s_i are triggered by either *join*, *delete*, *non-active* or *refresh* messages. When a location servers s_i receives a *join*, *delete* or *refresh* message and eliminates an identifier from a list, s_i sends a *delete* message to the appropriate child. By the argument used in the proof of Lemma 4.1, following t_1 the identifier of a mobile host m appears in at most one list of a location server. Thus, when a location server receives a *non-active* message from a child and deletes m no new static dangling identifier is produced.

2. Adding a mobile host identifier to a list while not adding this identifier to the list of the parent. Addition of an identifier to a leaf location server triggers a *join* message sent towards the root. Thus, no static dangling identifier is added. An addition of an identifier due to a *join* message either produces a *join* message towards the parent or eliminates the mobile host identifier from another list. Thus, the number of static dangling identifiers does not grow.

Note that the period of time $t_2 - t_1$ is bounded by the maximal duration of time between two successive executions of the code in Figure 2 by a location server, multiplied by the number of levels of the tree.

Now we prove that the number of static dangling identifiers is reduced. A location server with a static dangling identifier periodically receives a refresh message from its parent and deletes every identifier that does not exist in this list. Thus, the minimal level in which a static dangling identifier exists monotonically grows — until elimination. ■

Theorem 4.3 *The location management scheme presented in Figure 1 and 2 is self-stabilizing.*

Proof: By Lemma 4.2 there exists time t_2 after which there is no static dangling identifier. First note that by the definition of t_a and t_u, following t_2 there exists a time interval of length t_u during which m does not send an alive signal. Let I be the first instance that follows this time interval. I follows a period of time that ensures the completion of an update. Thus, there is no (static or non-static) dangling identifier in I. Therefore, the alive signal following I reaches the single location path — this implies correct update. ■

5 Concluding Remarks

In this paper we have suggested a new structure for location management in mobile networks. The structure is based on the tree location database structure. We suggested replacing the root and some of the higher levels of the tree with another structure that balances the average load of search requests. For this modification we used a *set-ary butterfly* network which is a generalization of the well-known *k-ary butterfly*. We also suggested modifying the lowest level of the tree to reflect better the neighboring relation between geographic cells and to support simple location data management. The modification of the lower lever supports a simple *follow-me handoff*.

Beyond the self-stabilization property of the location database, our modification of the tree structure is more robust than the original tree structure—if the location information of any mobile host is maintained by two embedded trees, then, even if one of the roots is crashed, still the location of this mobile host can be found.

Acknowledgments: Many thanks to Pattabhiraman Krishna and Nitin H. Vaidya for helpful discussions and suggestions.

References

[1] A. Acharya, B. R. Badrinath and T. Imielinski, "Checkpointing distributed applications on mobile computers," Tech. Rept., Dept. of Computer Science, Rutgers University, 1993.

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

[3] 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.

[4] P. Bhagwat and C. E. Perkins, "A Mobile Networking System Based on Internet Protocol(IP)," *Proc. of the USENIX Symposium on Mobile and Location-Independent Computing,* 1993.

[5] E.W. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control", Communications of the ACM 17,11 (1974), pp. 643-644.

[6] S. Dolev, "Optimal Time Self Stabilization in Dynamic Systems," *Proc. of the 7th International Workshop on Distributed Algorithms* (Springer-Verlag LNCS 725), pp. 160–173, September 1993.

[7] Dolev, S., Israeli, A., and Moran, S., "Resource Bounds for Self Stabilizing Message Driven Protocols," *Proc. of the 10th Annual ACM Symp. on Principles of Distributed Computing*, pp. 281-293, 1991.

[8] R. J. Fowler, "The complexity of using forwarding addresses for decentralized object finding," *Proc. of the 5th ACM Symp. on Principles of Distributed Computing,* pp. 108-120, 1986.

[9] T. Imielinski and B. R. Badrinath, "Mobile wireless computing: solutions and challenges in data management," Technical Report, Rutgers DCS-TR-296/WINLAB TR-49, Feb. 1993.

[10] J. Ioannidis, D. Duchamp and G. Q. Maguire Jr., "IP-based Protocols for Mobile Internetworking," *Proc. of ACM SIGCOMM*, pp. 235-245, 1991.

[11] J. Ioannidis and G. Q. Maguire Jr., "The Design and Implementation of a Mobile Internetworking," *Proc. of Winter USENIX,* Jan. 1993.

[12] K. Keeton, B. A. Mah, S. Seshan, R. H. Katz and D. Ferrari, "Providing connection-oriented network services to mobile hosts," *Proc. of the USENIX Symposium on Mobile and Location-Independent Computing,* Cambridge, Massachusetts, August 1993.

[13] P. Krishna, Nitin H. Vaidya and D. K. Pradhan, "Recovery in distributed mobile environments," *IEEE Workshop on Advances in Parallel and Distributed Systems,* pp. 83-88, Oct. 1993.

[14] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, Inc., 1992.

[15] Jean-Paul Linnartz, "Narrowband Land-Mobile Radio Networks," Artech House, 1993.

[16] Charles Perkins, "Providing Continuous Network Access to Mobile Hosts Using TCP/IP," *Joint European Networking Conference,* May 1993.

[17] J. Spinelli and R.G. Gallager, "Event Driven Topology Broadcast Without Sequence Numbers", *IEEE Transactions on Communication*, Vol. 37, No. 5, (1989) pp. 468-474.

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

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

[20] T. Watson and B. Bershad, "Local Area Mobile Computing on Stock Hardware and Mostly Stock Software," *Proc. of USENIX Symp. on Mobile and Location-Independent Computing,* 1993.

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

Figure 3: Butterfly

Figure 4: k-ary Butterfly, $k = 3$

Figure 5: set-ary Butterfly, $set = (3, 2)$

Figure 6: The Lower Level Modification