ELSEVIER

Performance Evaluation 20 (1994) 287–300

PERFORMANCE EVALUATION
An International Journal

Optimization of a database hierarchy for mobility tracking in a personal communications network

V. Anantharam [a,*], M.L. Honig [b], U. Madhow [b], V.K. Wei [b]

[a] *School of Electrical Engineering, Phillips Hall, Cornell University, Ithaca, NY 14853, USA*
[b] *Bellcore, 445 South Street Morristown, NY 07960, USA*

Abstract

The concept of personal communications is based on the ability of mobile users to maintain communication with a wireline network (such as the public switched telephone network) by means of a wireless link. Each user has a unique *name* (or personal number), and calls are routed to a *person* rather than to a stationary telephone. In order to set up such a call, the wireline network has to determine the current location of the user and route the call accordingly. Keeping track of a mobile user requires updates at databases within the network, and call routing requires accesses at these databases. In order to reduce the update and access load on each database, a message routing scheme based on a hierarchy of databases is proposed. We consider the problem of how to assign databases in this hierarchy so as to minimize the total rate at which accesses and updates occur, given estimates of mobility and calling rates between cells, and subject to a constraint on the access and update rates for each database. We show that an optimal hierarchy can be computed using a dynamic programming algorithm.

Key words: Personal communication; Mobility tracking; Wireless network

1. Introduction

A Personal Communications Network (PCN) employs a wireless connection to a wireline network that enables users to maintain communication while moving from one place to another. Each mobile user can communicate with a stationary radio port, or *base station*, using a wireless link. Provided that the current location of a user is known (or can be found) by the wireline network, calls to a mobile user can be routed to the appropriate base station using the conventional wireline network. The base station then transmits the call to the mobile user using a wireless link.

User mobility causes two main problems with call set-up and routing that are not encountered in a network containing only stationary users. Firstly, call set-up requires at least one *database access* to find the current location of the user being called. Secondly, if the network

* Corresponding author.

maintains an up-to-date list of the location of each mobile user, then each location change requires one or more *database updates*. One possible solution to these problems is to use a single database which stores the location of all the users in the network. This database must then be updated (accessed) every time a user changes location (makes a call). This solution becomes infeasible if a single database cannot handle the number of updates and accesses generated in this manner.

In this paper, we propose a hierarchical arrangement of databases for mobility tracking which reduces the access and update load on each database, relative to the centralized scheme just described. This scheme was motivated by recent studies which estimate the amount of updates, queries, and signalling traffic expected to arise in a PCN [6,7]. In [6], it is assumed, as in GSM [4], that each user is identified with a particular node in the signaling network, called a Home Location Register (HLR), which contains the user's current location. Each location change (call set-up) then requires that the HLR be updated (accessed). This system becomes inefficient if a user travels far from his HLR, or if there is a relatively large amount of traffic between two HLRs far apart in the network. In each case, update and query traffic must travel long distances over the signalling network. The hierarchical scheme presented here has the capability of greatly reducing the distance over which signalling traffic must travel in these situations.

We adopt the following general formulation of the problem of mobility tracking. The user is assumed to be located at one of the nodes of a tree network (i.e., a spanning tree extracted from an arbitrary connected network). Each node in this tree routes calls and signalling traffic, in addition to possibly containing a database which maintains a list of all users currently located at nodes in its associated subtree. For any given user, a pointer is maintained at each database on the path from the root to the node at which the user is located (all other databases have blank entries corresponding to the user). The lowest database on this path points to the node at which the user is located. Any other database on the path points to to the next database on the path. Call set-up is therefore established by following a sequence of pointers to the current location of the user. When the user moves to a nearby location, only databases which are relatively low in the hierarchy need to be updated. Similarly, call set-up between nearby users need not access databases relatively high up in the hierarchy. The tree structure therefore serves to balance accesses and updates among databases. However, an associated disadvantage is that a single location change or call may cause updates or accesses at several databases.

The preceding tradeoff leads to the following optimization problem. Given estimates of the rates of calls and movements between the nodes of the network, and given an upper bound on the rate at which each database can be updated or accessed (called 'capacity'), find a placement of databases at nodes in the tree so as to minimize the *total* number of updates and accesses per unit time in the network. If the capacity of the databases is sufficiently large, then the solution is always a centralized scheme in which a single database is placed at the root node. While the simple cost function considered here is appropriate for illustration, a number of other cost functions can be handled within the same framework. As mentioned in Section 3, one such cost function is the average call set-up delay due to queueing at the databases and communication links. Introduction of communication costs may lead to a more complicated solution than the centralized scheme even when the databases have an arbitrarily high capacity.

The type of signaling network considered here may cover geographical regions of various sizes, and may reflect geographical hierarchies. For example, the root node may cover an entire country, nodes at the next level in the tree may be gateways to networks covering states, nodes at the next level may be gateways to Metropolitan Area Networks, and so forth. Because cells in a PCN can be quite small, the wireline signaling network typically does not keep track of user locations down to the cell level. Rather, the nodes of the signaling network at which users can reside might represent wire centers or Local Area Networks (LANs). Once a user is identified to be at a particular node in the signaling tree, the particular cell in which he or she currently resides can be found, for example, by wireless broadcast.

The mobility tracking scheme presented here has been independently proposed in [10], although no attempt at optimization is described. Hierarchical tracking and routing in mobile telephony has also been considered in [1] and [9], although the approaches taken in these references are different from that described in this paper. In [1], a distributed hierarchy for mobility tracking based on graph partitioning is presented; however, estimates of call and mobility rates are not used to optimize database locations. Furthermore, the user's current location may not be accurately known at all times, so that a restricted broadcast is required for location queries. In [9], a hybrid scheme combining aspects of the hierarchical and HLR strategies is proposed, and a queueing analysis is presented for a model similar to the example presented in Section 4 of this paper. The authors assume a preassigned database configuration, and do not attempt any optimization.

Bar-Noy and Kessler [2] have recently considered the problem of selecting a subset of nodes in a tree, which represents the signaling network, so as to optimize the use of *wireless* resources for tracking mobile users. In contrast, our focus is on optimization within the *wireline* network. Despite this difference in application, the associated mathematical optimization problems turn out to be similar. Both can be solved using dynamic programming, and the complexity analyses for the algorithms are very similar.

A mathematical description of the mobility tracking model and the optimization problem considered are presented in Section 2. A dynamic programming solution to this optimization problem is presented in Section 3. Section 4 contains a simple example which illustrates the algorithm, and Section 5 contains our conclusions.

2. System model and problem formulation

Given a network represented by a connected graph, we construct a rooted spanning tree of this graph which forms the basis for the database hierarchy. The choice of this tree given the network graph may depend on traffic and mobility patterns within the network, and is not considered here. Users located at nodes within the network are *internal* users, and users located outside the network are *external*. Each node is capable of relaying messages up and down the tree, and has a list of the users that it currently contains. Some of the nodes also have databases, each of which contains entries for users at nodes in the subtree rooted at the node with the database. Each user is addressed by a unique identifier, or *name* (e.g., a personal phone number). We assume that the root of the tree always contains a database.

Requests for call set-up can arrive, or be generated at, any node in the network. If the called party is at the originating node, call set-up occurs without any database access. If the called

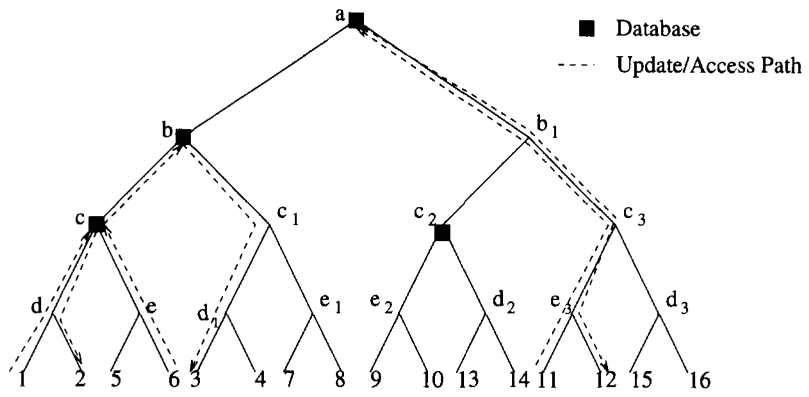

Fig. 1. Illustration of update and access paths in hierarchical scheme.

party is not at the originating node, the call set-up request is routed up the tree. Each database encountered along this route looks up the entry corresponding to the called party. If the entry is blank, meaning that the user is not in the subtree rooted at the node at which this database is located, the call set-up request continues up the tree. Otherwise, the entry is a pointer to a database further down the tree, and the call set-up request follows a chain of pointers down the tree to the node at which the called party is located. Call set-up requests to an external user reach the root without encountering any pointers, and are thus routed out of the network.

The preceding routing scheme requires that database entries accurately specify the location of the user at all times. Movement of a user from one node to another must therefore be accompanied by updates in the appropriate databases. Let the set of ancestors and descendants of a node in the tree be defined in the usual manner, with the convention that a node is its own ancestor and descendant. Also, let LCA (j, k) denote the least common ancestor of j and k, that is, LCA (j, k) is the unique node in the tree which is an ancestor of both j and k and is not a descendant of any other such ancestor. Consider now the movement of a user from node j to node k. Node k accepts the user by entering his name (say, U) in its list of local users, and then sends a message *up* the tree with the information that U has moved to k. Each database encountered by this message, if it does not already have an entry for U, adds an entry for U which points to the previous database on the path of the message; the first (lowest) such database adds a pointer to k, the user's new location. This message finally encounters a database which already has an entry for U (this node is the youngest ancestor of LCA (j, k) which contains a database). This database terminates the upward message, and initiates a message *down* the tree to node j, telling databases encountered on the way to erase their pointer entries for U. Note that a call set-up request originating at node k for a user at node j follows the same path as a movement from j to k, except that accesses rather than updates are required at each database encountered.

Fig. 1 illustrates the paths corresponding to updates for movements (or equivalently, accesses for calls) between nodes of the tree. For simplicity, we restrict attention to movements between leaves. Node c is the youngest ancestor of LCA $(1, 2) = d$ containing a database, and this is the only database updated for a movement from leaf 1 to leaf 2. Similarly, the youngest ancestral database of LCA $(11, 12) = e$ is the database at the root node a, and this is the only database involved in a movement from 11 to 12. Finally, LCA $(6, 3) = b$ contains a database, so that movements from 6 to 3 do not involve any databases higher than the one at b. However, every such movement also involves the database at c, since it lies on the path from 6 to 3.

Clearly, the number of updates and accesses generated by movements and call set-up requests is determined by the particular placement of databases in the spanning tree. A large number of databases significantly increases the number of updates and accesses caused by mobility, which may create unacceptable delays. On the other hand, a small number of databases may not be sufficient to handle the number of updates and accesses generated in the network. We now formulate this tradeoff as a combinatorial optimization problem. We first describe the problem in terms of location updates only, ignoring accesses generated by call set-up requests. As shown at the end of this section, accesses can also be handled within the same framework. We show in the next section that this problem can be solved with a dynamic programming algorithm. This algorithm can be applied to solve other versions of this problem, including a formulation in which the objective function to be minimized is the average delay arising from a nondeterministic queueing model.

Assume that each database can handle up to C updates per unit time; the number C is referred to as the *database capacity*. We assume that estimates are available for the rate of movements between base stations. In particular, m_{jk} is the number of movements per unit time between node j and node k (if node j does not contain any users, then $m_{jk} = m_{kj} = 0$ for all $k \in V$, where V denotes the collection of the nodes of the tree).

Let A denote the subset of nodes containing databases. Our optimization problem is to choose A so as to minimize the total rate of database updates, subject to the constraint that the rate of database updates at any node in A does not exceed the database capacity C. In order to solve this problem, we need to develop a more precise formulation of the constraints and the objective function, which requires some additional notation as follows.

For any node i, define

$$\alpha_i = \sum_{(j,k):\mathrm{LCA}(j,k)=i} m_{jk}. \tag{1}$$

This sum is over distinct pairs, so each pair is only counted once. The interpretation of the above quantity is as follows. Even if there are databases at all the children of i, the lowest ancestral database of i must be updated due to the movements contributing to α_i (and no higher database needs to be updated due to such movements). For instance, if there is only one database (so that $A = \{r\}$, where r denotes the root node), then the update rate it sees is $\sum_{i \in V} \alpha_i$. On the other hand, if the children of the root contain databases, then the database at the root only sees an update rate of α_r. This is a simple illustration of how the hierarchy distributes the update load among the databases.

For disjoint subsets $R, S \subset V$, we write $M(R \to S)$ for the rate of movements from R to S and $M(R \leftrightarrow S)$ for the rate of movements between R and S. Thus

$$M(R \to S) = \sum_{i \in R} \sum_{j \in S} m_{ij}, \qquad M(R \leftrightarrow S) = \sum_{i \in R} \sum_{j \in S} (m_{ij} + m_{ji}), \qquad M(i \to j) = m_{ij},$$

For each node i, we define the quantity

$$\phi_i = M(V_i \leftrightarrow V - V_i), \tag{2}$$

where V_i denotes the set of descendants of i. If there is a database at i, this is the update rate seen by the database due to movements between its descendants and all other nodes in the tree. These updates arise simply because the database at i happens to lie in the path between the nodes involved in such movements. These movements thus constitute the penalty associated with the hierarchical scheme as compared to a centralized scheme with a single database at the root.

We now have all the notation needed for a mathematical formulation of the problem. The net update rate, which is the objective function in the optimization, is given as follows.

Lemma 1. *The criterion to be minimized, i.e., the rate of database updates per unit time, is given by*

$$\sum_{i \in V} \alpha_i + \sum_{i \in A} \phi_i. \tag{3}$$

Note that the first term is a constant independent of A, and it is the second term which is to be optimized over choices of A. The cost of placing a database at node i is thus ϕ_i. Note that $\phi_r = 0$, so that the root always contains a database.

Proof. The highest database that must be updated due to a movement from node j to node k is the lowest ancestor of LCA (j, k) which contains a database. It is clear from Eq. (1) that the first term above corresponds to these updates. Any other databases that lie in the graph-theoretic path from j to k, i.e., databases at nodes that have either j or k, but not both, as descendant, must also be updated. We see from Eq. (2) that this corresponds to the second term above. $\square$

We now must express the capacity constraint in our optimization problem. For nodes i and j, let $j \preccurlyeq i$ denote that j is a descendant of i, and let $j \prec i$ denote that j is a *proper* descendant of i (i.e., $j \preccurlyeq i$ and $j \neq i$). Given the set of databases A, we can define, for any $i \in V$, the following subset of nodes:

$$W_A(i) = \{ j \preccurlyeq i : \nexists\, k \in A,\ j \preccurlyeq k \prec i \}. \tag{4}$$

This set, referred to as the *waffle* of A at i, is the set of descendants j of i such that there is no database between j and i. Note that, by convention, $i \in W_A(i)$. If a database is placed at i, it sees, for each $j \in W_A(i)$, all updates due to movements between nodes for which j is the least common ancestor; this corresponds to a movement rate of α_j. In order for a subset A of databases to satisfy the capacity constraints, therefore, we must have, for all $i \in A$,

$$\phi_i + \sum_{j \in W_A(i)} \alpha_j \leqslant C. \tag{5}$$

The following lemma enables us to simplify the above constraint.

Lemma 2. *Fix $i \not\subset A$, and let k be the parent of i. Then the potential update rate seen by a database at k is at least as large as the rate that would have been seen by a database at i; that is,*

$$\phi_k + \sum_{j \in W_A(k)} \alpha_j \geqslant \phi_i + \sum_{j \in W_A(i)} \alpha_j$$

Using induction on the above, a constraint violation at i would cause a constraint violation at the first ancestor of i that contains a database. Thus, Eq. (5) must hold for *all* nodes i, not just for nodes with databases.

Proof. The proof is simply a matter of adding up the appropriate movement rates. Clearly, $\sum_{j \in W_A(k)} \alpha_j \geqslant \alpha_k + \sum_{j \in W_A(i)} \alpha_j$, so that it suffices to show that $\alpha_k + \phi_k \geqslant \phi_i$. Letting D_l and V_l denote the children and descendants of node l, respectively, we can express the movement rates in the following notation:

$$a = M(V_i \leftrightarrow V - V_k), \qquad b = M(k \leftrightarrow V - V_k), \qquad c = \sum_{j \in D_k, j \neq i} M(V - V_k \leftrightarrow V_j),$$

$$d = \sum_{j \in D_k, j \neq i} M(V_i \leftrightarrow V_j), \qquad e = M(k \leftrightarrow V_i), \quad f = \sum_{j \in D_k, j \neq i} M(k \leftrightarrow V_j).$$

It is easy to see that $\phi_k = a + b + c$, $\alpha_k = d + e + f$, and $\phi_i = a + d + e$, which implies the required result. □

At this point, it is worth restating the problem formally as follows.

Problem statement. Minimize $\sum_{i \in A} \phi_i$ subject to $\phi_i + \sum_{j \in W_A(i)} \alpha_j \leqslant C$ for all $i \in V$.

The dynamic programming algorithm for solving this problem is given in the next section.

We have ignored database accesses so far in our mathematical formulation. In the remainder of this section, we show how both accesses and updates can be handled in the above framework. Let c_{jk} be the rate of calls originating at node j for users at node k (as explained earlier, this includes calls from external users). Calls from internal users at node j to external users make their way to the root, and are therefore included in c_{jr}. Calls originating at j for users at j do not require database accesses, so we set $c_{jj} = 0$ for all $j \in V$. With these conventions, for any j, $k \in V$, a call from j to k requires accesses at precisely those databases which would be updated as a consequence of a movement from j to k. We assume now that the cost of a database access is C_a, and the cost of a database update is C_u, and that the load at a database is the weighted sum of the access and update rates it sees. The database capacity is redefined to be the largest such load it can sustain. It is easy to see that the preceding development applies without change if we replace m_{jk} by $C_u m_{jk} + C_a c_{kj}$ for all j, $k \in V$, since a call from k to j follows the same (directed) path through the tree as a movement from j to k.

3. Algorithm for optimal database placement

Assume that α_i and ϕ_i have been computed for each node $i \in V$, and that they are integers. Assume also that the capacity C is an integer. These are not very restrictive assumptions, since they are equivalent, by means of appropriate scaling, to assuming a finite precision representation of these quantities. In practice, the estimates of the movement and calling rates are expected to be fairly coarse, so that the number of bits of precision is expected to be small.

The optimization problem is solved by the following strategy. As before, let $V_i = \{j \in V : j \preccurlyeq i\}$ denote the set of nodes in the subtree rooted at i. Let A_i denote the set of databases $V_i \cap A$ contained in this subtree. These databases contribute $\sum_{j \in A_i} \phi_j$ to the objective function; we call this quantity the *resource accumulation* at i, denoted by $R(i)$. The choice of A_i also affects the evaluation of the constraint (see Eq. (5)) at the parent of i, denoted by P_i, since $W_A(P_i)$ depends on A_i. Only part of the sum $\sum_{j \in W_A(P_i)} \alpha_j$ in Eq. (5) depends on A_i; this is the quantity $\sum_{j \in W_A(P_i) \cap V_i} \alpha_j$, and is called the *constraint accumulation* at i, denoted by $C(i)$. The effect of choosing A_i on the choice of databases further up the tree is thus summarized by specifying the quantities $R(i)$ and $C(i)$. For each node i, we minimize, over choices of A_i, the resource accumulation $R(i)$, given a value of the constraint accumulation $C(i)$. For a constraint accumulation $C(i)=\theta$, where $\theta = 0, 1, \ldots, C$, the minimum resource accumulation at i is denoted by $R^*(i, \theta)$, and the optimizing choice of database locations is denoted by $A_i^*(\theta)$. As shown below, given $R^*(k, \theta)$ for $\theta = 0, 1, \ldots, C$, and for each $k \in D_i$, we can compute $R^*(i, \theta)$, $\theta = 0, 1, \ldots, C$, for the parent node i by *glueing* together the solutions to the subproblems for the children nodes. This dynamic programming approach is completely specified, therefore, by specifying the glueing procedure.

The constraint accumulation at i is given by

$$C(i) = \begin{cases} \sum_{j \in W_{A_i}} \alpha_j, & i \notin A_i, \\ 0, & i \in A_i. \end{cases} \tag{6}$$

The resource accumulation at i is given by

$$R(i) = \sum_{j \in A_i} \phi_j. \tag{7}$$

The minimum resource accumulation for a given constraint accumulation θ is given by

$$R^*(i, \theta) = \min_{A_i} R(i) \quad \text{subject to } C(i) = \theta. \tag{8}$$

Given the set of minimum resource accumulations at all the children of i, the following glueing procedure yields the set of minimum resource accumulations at i. Letting D_i denote the set of children of i as before, we have, for $\theta = 1, \ldots, C$, that

$$R^*(i, \theta) = \min\left\{ \sum_{l \in D_i} R^*(l, \theta_l) \colon 0 \leqslant \theta_l \leqslant C \text{ for } l \in D_i \text{ and } \alpha_i + \sum_{l \in D_i} \theta_l = \theta \right\}, \tag{9}$$

where we set $R^*(i, \theta) = \infty$ when the minimization range is empty. If $\{\theta_l^*, l \in D_i\}$ are the values that achieve the above minimization, then the optimal set of databases is given by

$$A_i^*(\theta) = \bigcup_{l \in D_i} A_l^*(\theta_l^*). \tag{10}$$

For $\theta = 0$, which corresponds to placing a database at i (see Eq. (6)), we have

$$R^*(i, 0) = \min\left\{ \phi_i + \sum_{l \in D_i} R^*(l, \theta_l) \colon 0 \leqslant \theta_l \leqslant C \text{ for } l \in D_i \text{ and } \phi_i + \alpha_i + \sum_{l \in D_i} \theta_l \leqslant C \right\}, \tag{11}$$

and

$$A_i^*(\theta) = \{i\} \bigcup_{l \in D_i} A_l^*(\theta_l^*), \tag{12}$$

where $\{\theta_l^*, l \in D_i\}$ achieve the minimization above.

The optimal database placement corresponds to $A_r^*(0)$, and the minimum cost incurred is $R^*(r, 0)$, where r denotes the root node. The above algorithm gives us a means of computing this quantity by starting from the leaves, and successively glueing solutions together until we reach the root. For any leaf l, we have $R^*(l, \theta = 0) = \phi_l$, $R^*(l, \theta = \alpha_l) = 0$, and $R^*(l, \theta) = \infty$ for all other θ.

The complexity of the algorithm is determined by that of the glueing operation specified in Eq. 9–12. Let d_i denote the number of children of i, i.e., $d_i = |D_i|$. Considering $d_i = 2$, the computation in Eq. (9) takes time $O(C)$ for each θ, and hence time $O(C^2)$ for all θ in the range $1 \leqslant \theta \leqslant C$. The complexity of the computation in Eq. (11) is also bounded by $O(C^2)$. This analysis extends for arbitrary d_i as follows. The subsolutions for the first two children, k_1 and k_2 (say), can be combined into a single subsolution using the following rule:

$$R^*((k_1, k_2); \theta) = \min\{R^*(k_1, \theta_l) + R^*(k_2, \theta_2): 0 \leqslant \theta_1, \theta_2 \leqslant C \text{ and } \theta_1 + \theta_2 = \theta\},$$

where $1 \leqslant \theta \leqslant C$. This computation is $O(C^2)$, and the resulting subsolution can be viewed as the subsolution for a single child which replaces k_1 and k_2. Since this effectively reduces the number of children of i by one, we obtain, continuing in this fashion, that the time for combining all the subsolutions is $O(d_i C^2)$. Summing this complexity over the nodes of the tree, and using the fact that $\Sigma_{i \in V} d_i = |V| - 1$, we obtain an overall complexity of $O(|V|C^2)$. It is worth noting, however, that we have reduced a problem with real-valued parameters to one with integer parameters by truncating the precision. While the resulting solution suffices for most engineering purposes, obtaining an efficient and 'purely combinatorial' (i.e., independent of the precision) algorithm remains an open problem.

This completes the specification of the algorithm and its complexity. We note that dynamic programming is applicable to many alternative problem formulations as well. At each node i, we preserve all possible subsolutions that can be part of the overall optimal solution. As we go up the tree one level, we combine the subsolutions from lower down in the tree and reduce the number of possibilities by discarding subsolutions which, for the same constraint accumulation, have a higher resource accumulation (and hence cannot be part of the overall optimal solution). This approach can be applied to other problems by suitably modifying the structure of the subsolution to reflect the characteristics of the problem. For instance, consider a nondeterministic formulation of the problem in which that m_{ij} and c_{ij} are the rates of Poisson processes modeling movements and calls, respectively, from i to j. Databases are modeled as M/D/1 or M/M/1 queues with a given capacity C, and the objective function to be minimized is the average delay for a movement or call (or equivalently, a weighted sum of the delays encountered at the databases). The constraints are implicit, in that the delay goes to infinity if the capacity of a database is exceeded. It is convenient to assume that the service time for accesses and updates are identically distributed (otherwise the delay formula as well as the structure of the subsolutions is more complicated). With this assumption, we can set $C_a = C_u = 1$, and use exactly the same formula as before for the constraint accumulation. However, instead of accumulating ϕ_i when a database is placed at i, we now accumulate the average delay for the corresponding queueing model, weighted by the input rate, to determine the resource accumulation. Thus, the glueing formulas (9)–(12) are unchanged except for Eq. (11), which is replaced by

$$R^*(i, 0) = \min\left\{K\left(\alpha_i + \phi_i + \sum_{l \in D_i} \theta_l, C\right) + \sum_{l \in D_i} R^*(l, \theta_l):\right.$$
$$\left. 0 \leqslant \theta_l \leqslant C \quad \text{for } l \in D_i \quad \text{and } \phi_i + \alpha_i + \sum_{l \in D_i} \theta_l \leqslant C\right\}. \tag{13}$$

If $D(\lambda, \mu)$ is the average delay for a queue with input rate λ and service rate μ, then $K(\lambda, \mu) = \lambda D(\lambda, \mu)$. For two typical queueing models, we have [5]

$$D(\lambda, \mu) = \begin{cases} (1 - \lambda/2\mu)/(\mu - \lambda), & M/D/1, \\ 1/(\mu - \lambda), & M/M/1. \end{cases} \tag{14}$$

Another important variation that can be handled using our algorithm is the introduction of communication costs over the links. The amount of traffic in either direction on a given link (from node i to its parent P_i, for instance) is specified completely by the placement of databases in the subtree rooted at i. As before, the cost function to be minimized may be either the sum of the traffic on the links or of the link delays arising from a queueing model. In either case, dynamic programming can be applied as before (with some expansion of the information preserved in the subsolutions and a resulting increase in complexity) to find the optimal placement of databases given the link capacities, the database capacities, and the calling and mobility rates. Note that, with the introduction of communication costs, a centralized scheme with a database at the root may not be optimal even if there is no constraint on database capacity.

While exploring the appropriate variations of our problem in specific contexts is the ultimate goal from a system designer's viewpoint, our purpose in this paper is restricted to presenting the basic idea of hierarchical routing and its optimization using dynamic programming. In the example of the next section, therefore, we consider only the original problem formulation leading to Eqs. (9)–(12).

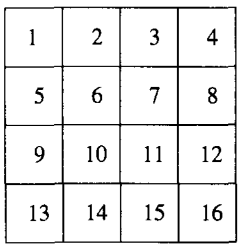

Fig. 2. Coverage area subdivided into location areas.

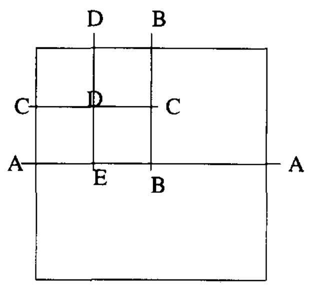

Fig. 3. Boundaries used for tree construction.

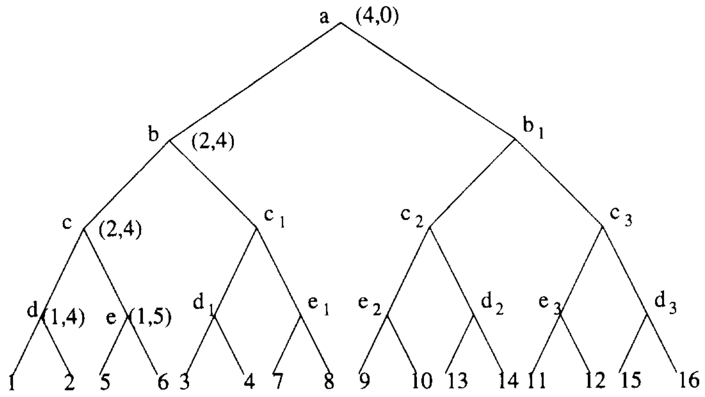

Fig. 4. Tree of switching nodes and location registers.

4. Example

For simplicity, we only account for updates in this example. Consider a square region $\mathcal{S}$ which is subdivided into 16 smaller squares as shown in Fig. 2. The scale is chosen so that the sides of the small squares equal one. Each small square, referred to as a *location area*, represents a geographical area in which a user may be located, and is associated with a leaf of the signaling tree. Each leaf contains a *location register* which contains a list of the users who are currently in that location area. Each internal node of the tree represents a switching node which may contain a database. Fig. 3 displays a representative set of natural boundaries used for the construction of the signaling tree. The connectivity of the tree is determined by placing a switching node for each such boundary. This yields the tree shown in Fig. 4. The symmetry inherent in this example is reflected in the labelling of the tree, so that nodes corresponding to similar boundaries are labelled by the same letter, and are distinguished by a subscript. The root node a corresponds to boundary AA, the node b to boundary BB (there are two nodes of this type), c to CC (four nodes of this type), d to DD (four nodes of this type), and e to DE (four nodes of this type). The labelling for the representative node i of each type includes the values (α_i, ϕ_i) for that type of node. These values are determined using a fluid flow model of mobility in which the rate of movements between two regions is proportional (taking the proportionality constant to be unity) to the length of their common boundary. Thus, the movement rate between location areas 1 and 2 equals one the length of their common boundary, so that $m_{12} + m_{21} = 1$. On the other hand, the movement rate between location areas 1 and 3 equals zero since they have no common boundary. We can now compute the flows between the leaves and hence α_i and ϕ_i for the internal nodes. However, formulas (1)-(2)

Table 1

θ	$S(d)$		$S(e)$	
	$R^*(d, \theta)$	$A^*(d, \theta)$	$R^*(e, \theta)$	$A^*(e, \theta)$
0	3	$\{d\}$	5	$\{e\}$
1	0	$\emptyset$	0	$\emptyset$

Table 2

θ	$S(c)$		$S(c_1)$	
	$R^*(c, \theta)$	$A^*(c, \theta)$	$R^*(c_1, \theta)$	$A^*(c_1, \theta)$
0	4	$\{c\}$	4	$\{c_1\}$
⋆2	8	$\{d, e\}$	8	$\{d_1, e_1\}$
3	3	$\{d\}$	3	$\{d_1\}$
4	0	$\emptyset$	0	$\emptyset$

indicate a more direct method of computation. Each switching node i corresponds to a boundary $\mathscr{B}$ dividing a region $\mathscr{R}$ in two. It is easy to see that α_i equals the length of $\mathscr{B}$ and ϕ_i equals the length of the common boundary of $\mathscr{R}$ and $\mathscr{S}-\mathscr{R}$. This yields the (α_i, ϕ_i) pairs shown in Fig. 3.

It is easy to see, using Lemma 2, that the minimum database capacity for which there is a feasible solution to the optimization problem of interest is $\max_{i \in V}(\alpha_i + \phi_i)$, which equals 6 for this example. On the other hand, the database capacity for which a centralized scheme with a single database at the root is feasible (and hence optimal) is $\Sigma_{i \in V}\alpha_i$, which equals $C = 24$. The nontrivial range of C is therefore $6 < C < 24$. We take $C = 12$ in the example we work through, so that the databases available have half the capacity required for a centralized scheme.

For each node i, we determine a subsolution to the optimization problem which lists in a table $S(i)$ the values of θ, $R^*(i, \theta)$, and $A^*(i, \theta)$, for each $\theta = 0, \dots, C$. Values of θ corresponding to $R^*(i, \theta) = \infty$ are omitted from the table, since they cannot be used to construct an optimal solution.

The subsolutions for the parents of the leaves (i.e., nodes of types d and e) can be computed as follows. For node d, consider $\theta = 0$, which corresponds to placing a database at d. Since $\alpha_d + \phi_d = 1 + 3 \leqslant C = 12$, this is feasible. The resource accumulation equals $\phi_d = 3$. The only other possibility is $\theta = \alpha_d$, which corresponds to not placing a database at d (and hence, zero resource accumulation). The subsolution for node e is generated similarly. See Table 1.

These subsolutions are now combined Table 2 using Eqs. (9)–(12) ($i = c$, $d_i = 2$, $D_i = \{d, e\}$) to obtain the subsolution for node c given by (the subsolution $S(c_1)$ is also listed, and is obtained from $S(c)$ by symmetry).

The subsolutions for nodes c and c_1 can now be combined to yield the subsolution for node b (($i = b$, $d_i = 2$, $D_i = \{c, c_1\}$). There may be several choices for $A^*(b, \theta)$, and we list only one

Table 3

θ	$S(b)$		$S(b_1)$	
	$R^*(b, \theta)$	$A^*(b, \theta)$	$R^*(b_1, \theta)$	$A^*(b_1, \theta)$
0	8	$\{b, c\}$	8	$\{b_1, c_2\}$
⋆2	8	$\{c, c_1\}$	8	$\{c_2, c_3\}$
5	7	$\{c, d_1\}$	7	$\{c_2, d_3\}$
6	4	$\{c\}$	4	$\{c_2\}$
⋆8	6	$\{d, d_1\}$	6	$\{d_2, d_3\}$
9	3	$\{d\}$	3	$\{d_2\}$
10	0	$\emptyset$	0	$\emptyset$

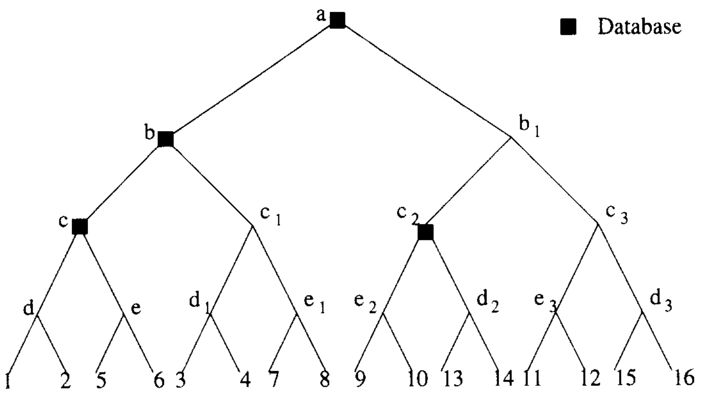

Fig. 5. Optimal database placement.

representative choice (the remaining can be deduced easily by symmetry). As before, the subsolution $S(b_1)$ can be obtained from $S(b)$ by symmetry, cf. Table 3.

While rows corresponding to a higher constraint accumulation θ are expected to correspond to a lower resource accumulation, for some values of θ, the equality constraint in Eq. (8) may only be satisfied for a very bad configuration of databases. Thus, for a subsolution $S(i)$, there may be $\hat{\theta} < \theta$ for which $R^*(i, \hat{\theta}) \leqslant R^*(i, \theta)$. In this case, the row corresponding to θ can never be used in an optimal solution (since it is preferable to use another row with both a lower constraint accumulation and a lower resource accumulation), and hence can be deleted from $S(i)$. Two such rows appearing in the above subsolutions are marked by asterisks.

We are finally in a position to compute the optimal solution, which corresponds to $\theta = 0$ for node a, and is obtained from the subsolutions $S(b)$ and $S(b_1)$. We obtain that

$$R^*(a, 0) = 12, \quad A^*(a, 0) = (a, b, c, c_2).$$

The placement of databases is illustrated in Fig. 5. We note that the original tree considered may be thought of as a *logical* tree reflecting geographical considerations. If so, after the optimal database placement is determined, we can remove the remaining nodes of the tree, preserving only the database nodes and the base stations. This gives rise to the tree shown in Fig. 6. Thus, our solution achieves a grouping of the base stations and an interconnection of the databases which is based on both geography and database capacity.

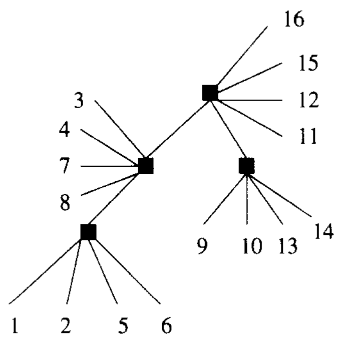

Fig. 6. Optimal database placement after eliminating unused nodes and links.

5. Conclusions

We propose a routing scheme for personal communications based on a hierarchy of databases, and give a dynamic programming algorithm for optimizing the placement of these databases. Dynamic programming can also be used for optimizations involving different objective and constraint functions which may arise in specific design scenarios. One topic for future research is to explore such scenarios in more detail. Another important area for investigation is the comparison of our hierarchical scheme with other methods for mobility management, such as the idea of associating each user with a home location as in GSM. In particular, it may be possible to design a hybrid scheme combining the advantages of these two systems. Other topics include the possibility of reducing the number of updates and accesses by (i) using prior information about a user's mobility profile, and (ii) maintaining only coarse location information, which is refined at the time of a location query by a limited broadcast.

Acknowledgements

We thank Ken Steiglitz (Princeton University) and Seshadri Mohan (Bellcore) for their helpful comments. We also thank Ilan Kessler (IBM) for supplying a preprint of [2]; using techniques similar to those in [2] enabled us to improve our estimate of the complexity of the dynamic programming algorithm.

References

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

[2] A. Bar-Noy and I. Kessler, Tracking mobile users in wireless communications networks, *Proc. Infocom '93*, 1993.

[3] T. Imielinski and B.R. Badrinath, Querying in highly distributed mobile environments, *Proc. 18th VLDB Conf.*, Vancouver, BC, Canada, 1992.

[4] J.A. Audestad, GSM general overview of network functions, *Proc. Int. Conf. Digital Land Mobile Radio Commun.*, Venice, 1987.

[5] L. Kleinrock, *Queueing Systems, vol. 1* (Wiley, New York, 1975).

[6] K.S. Meier-Hellstern, E. Alonso and D.R. O'Neil, The use of SS7 and GSM to support high density personal communications, *Proc. ICC '92*, pp. 356.2.1-356.2.5.

[7] C.N. Lo, R.S. Wolff and R.C. Bernhardt, An estimate of network database transaction volume to support voice personal communications services, Proc. UPC (Universal Personal Communication) '92, 1992.

[8] C.N. Lo, S. Mohan, and R.S. Wolff, Performance modeling and simulation of data management for personal communications applications, to appear in *Proc. 2nd Bellcore Symp. Perf. Modeling*, 1992.

[9] C. Wang, J. Wang and Y. Fan, Performance evaluation of a hierarchical location registration and call routing for personal communications, *Proc. ICC '92*, pp. 356.6.1-356.6.4.

[10] J.Z. Wang, The hierarchical structure of tracing strategy for universal personal communication systems, *Proc. UPC '92*, pp. 09.04.1-09.04.5.