Per-User Profile Replication in Mobile Environments: Algorithms, Analysis, and Simulation Results*

Narayanan Shivakumar, Jan Jannink, Jennifer Widom
Department of Computer Science
Stanford University, Stanford CA 94305
{*shiva, jan, widom*}@*cs.stanford.edu*

Abstract

We consider per-user profile replication as a mechanism for faster location lookup of mobile users in a Personal Communications Service system. We present a minimum-cost maximum-flow based algorithm to compute the set of sites at which a user profile should be replicated given known calling and user mobility patterns. We then present schemes for replication plans that gracefully adapt to changes in the calling and mobility patterns. We show the costs and benefits of our replication algorithm against previous location lookup approaches through analysis. We also simulate our algorithm against other location lookup algorithms on a realistic model of a geographical area to evaluate critical system performance measures. A notable aspect of our simulations is that we use well-validated models of user calling and mobility patterns.

1 Introduction

In a Personal Communications Service (PCS) system, users place and receive calls through a wireless medium. Calls may deliver voice, data, text, facsimile, or video information [15]. PCS users are located in system-defined *cells*, which are bounded geographical areas. When a user places a call, the PCS infrastructure must route the call to the *base-station* located in the same cell as the callee. The base-station then transmits the data in the call to the PCS unit through the wireless medium.

We consider the problem of locating users who move from cell to cell while carrying PCS units. When user A places a call to user B, the *location lookup* problem is to find callee B within "reasonable" time bounds, so that the call can be set up from A to B. In this paper we make the following assumptions about the structure of the system. Users are located in geographical *zones*, which may be cells or groups of cells. Each zone has a database that stores *profiles* of users in the form $< PID, ZID >$, where PID and ZID uniquely identify the PCS unit (say the telephone number of the unit) and the current location (zone ID) of the unit, respectively.[1] Each PID maps to a *home* zone, whose database always maintains an up-to-date

*Research supported by the Center for Telecommunications and the Center for Integrated Systems at Stanford University, the Anderson Faculty Scholar fund, and equipment grants from IBM and Digital Equipment Corporation.

[1] User profiles actually contain more information, but for our purposes we are interested primarily in location information.

copy of the user's profile.

In current cellular standards such as GSM and IS-41 [18], the home zone is referred to as the *Home Location Register* (HLR). When user A calls user B, the lookup algorithm must initiate a remote lookup (query) to the HLR of B, which may be at a remote site. We call this the *pure HLR* scheme. Since performing remote lookups can be slow due to high network latency, current systems usually improve the HLR scheme by maintaining *Visitor Location Registers* (VLR). The VLR at a zone stores the profiles of users currently located in that zone. The modified lookup strategy in this HLR/VLR scheme is:

1. Query database in caller's zone.

2. If callee's profile is not found, query database in callee's home zone.

This optimization is useful when a callee receives many calls from users in the zone he is visiting, since remote lookups to the callee's HLR are avoided. VLRs can be viewed as a simple, limited *replication scheme*, since each user's profile is replicated at the zone the user is located in when he is not in his home zone.

In this paper we propose a more general replication scheme. We could choose to replicate each user profile at all zones for fastest possible lookup, but the resulting storage and update costs would be prohibitively high. Instead, we propose a *per-user profile replication scheme* based on calling and mobility patterns. Our scheme respects storage and update limitations while speeding up location lookups. In our scheme, the decision of where profiles should be replicated is based on a *minimum-cost maximum-flow* [1] algorithm.

In our approach, we still maintain an up-to-date copy of a user profile at the user's HLR. Our algorithm computes additional sites at which the user's profile is replicated (see Section 2 for details on our notion of replication consistency). Consequently, the lookup algorithm in our replication scheme is the same as in HLR/VLR: look in the local database before querying the HLR. While our scheme does not guarantee that a user's profile will be stored in his current zone (as VLR does), a trivial modification of our replication scheme can also guarantee VLR behavior. We assume that the home location of each user keeps track of which sites contain replicas of the user's profile. When a user crosses zones, the user's home location will initiate the updating of the profile replicas.

In Section 2, we present our replication algorithm given fixed calling and mobility (*traffic*) patterns. In Section 3, we present an incremental algorithm that dynamically adapts to changes in traffic patterns. In Section 4, we propose two different modifications to the incremental algorithm that "gracefully" evolve the replication plan. In Section 5, we compare the advantages and associated costs of replication against *caching* [12, 15] and against the traditional HLR schemes. We see that replication has significantly better response times and generates less network traffic. In Section 6, we consider some issues in implementing our replication strategy in a large-scale wireless network. In Section 7, we report on large-scale simulation results we have obtained that compare system performance for our replication scheme against several other location lookup algorithms. In Section 8, we show how our replication scheme can be used to enhance non-HLR lookup schemes. We also propose *service information replication*, an additional application of our replication framework. We conclude in Section 9 with directions for future research.

1.1 Related Work

Several previous approaches have considered enhancements or alternates to the HLR/VLR lookup algorithm. Awerbuch and Peleg [3] propose a formal model for online tracking of users by decomposing the PCS network into *regions* and using a hierarchy of *regional directories*. This elegant framework shows how to trade off search and update costs while tracking users. Badrinath et al. [4] consider *per-user placement*, which uses *partitions* (cells commonly visited by a user) to control network traffic generated by frequent updates. Jain and Lin [14] propose using *forwarding pointers* to reduce the number of database updates, while keeping lookup costs low. Ho and Akyildiz [13] propose a *local anchoring* scheme for highly mobile users to reduce the number of remote HLR updates. Our replication approach is orthogonal to most of these schemes, and could be used to complement one or more of them.

Jain et al. [12, 15] propose *per-user caching* where zones cache the last known location of certain users for faster lookup. Replication is different from caching in that replication always keeps all copies up-to-date, and there is no invalidation problem. An additional difference is that we consider storage constraints in our databases, while [12, 15] do not. We see in later sections (through analysis and simulations) that our scheme generally performs better than caching in terms of lookup time and network cost.

Anantharaman et al. [2] use dynamic programming to optimize mapping a database hierarchy to the network configuration, based on fixed calling and mobility patterns. Their architecture does not consider communication costs, and it does not adapt to changes in patterns.

In our approach, we choose the "best" zones for replication of user profiles based on calling and mobility patterns. In that sense, our problem is similar to the classical *database allocation* [20] and *file allocation* [10] problems, in which databases or files are replicated at sites based on lookup-update or read-write access patterns. Some popular formulations used in these two problems have been based on the knapsack problem [7], branch-and-bound [11], and network flow algorithms [8]. However, these schemes do not adapt to changes in access patterns. Wolfson and Jajodia propose an on-line algorithm for dynamic data replication in distributed databases using a "no-knowledge" approach [25]. While this algorithm converges to the optimal replication plan when traffic traces are regular, unlike our scheme theirs does not exploit the relative stability of calling and mobility patterns of users (see Section 2) for fast convergence. Their scheme also does not consider storage constraints.

For readers familiar with the *transportation problem* [5], the difference in our work is that we do not specify which sites are the destinations of user replicas. Rather, we choose replication sites dynamically based on capacity and cost versus benefit considerations.

A preliminary version of this paper appeared in [21]. That paper included only algorithms and analysis. Here, we also report on our recent and fairly extensive simulation results, which corroborate our analytical claims in [21] that per-user profile replication is a promising and feasible approach.

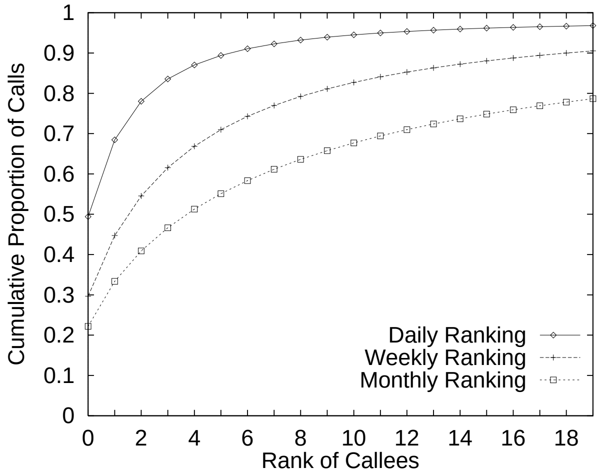

Figure 1: Locality in calling patterns.

2 The Replication Strategy

In Section 2.1, we use collected data to motivate why replication-based location lookup algorithms may lead to significant performance improvements. In Section 2.2, we address some practicality concerns that must be resolved before replication can be used in a real system. In Section 2.3, we define parameters used in subsequent sections. We present the minimum-cost maximum-flow based algorithm to compute the replication plan for user profiles in Section 2.4. In Section 2.5, we show how to compute the parameters defined in Section 2.3.

2.1 Motivation for Replication

We have obtained actual calling traces from the Stanford University Communication Services for a six-month period from March to September 1995 to study user calling patterns. These traces correspond to 19,592 distinct callers in our campus (university offices, student housing, and residential homes). The data set contains caller number (encrypted for privacy reasons), callee number (again encrypted), time of call, and call duration, for all calls destined to locations outside our local exchange. In Figure 1, we rank the callees by the number of calls they receive from a certain caller and report the cumulative distribution of the calls made by callers to those callees on a daily, weekly, and monthly basis. For instance, we see from Figure 1 that more than 70% of the calls made by callers in a week are to their top 5 callees.

Replication attempts to exploit this observed *locality* in user calling patterns. For example, if we choose to replicate the top five callee profiles at the caller's site, then nearly 70% of the calls made in a week are serviced by lookups on local databases, reducing the latency of location lookup and the bandwidth requirement for remote lookups. Of course, the decision for replication is not quite so simple since we have not factored in how the callees move, and how system resource constraints (outlined below) affect the decision of which profiles to replicate; this replication decision is the subject of the remainder of this section.

2.2 Issues in Replication

Replicating user profiles can reduce lookup time significantly, since the location of the callee is more likely to be obtained by a single lookup on the local database rather than a high latency remote lookup. However, the associated cost of replication is the update cost incurred in maintaining consistent replicas every time a user moves. In traditional distributed databases, replica consistency is maintained by using distributed locking protocols, which can be very expensive [20]. By contrast, we may use a "looser" form of replica consistency, which does not rely on locking and provides fast propagation of updates to replicas. Similar to HLR/VLR handoff, forwarding pointers are maintained briefly at the old location of a user to handle incoming calls from remote sites that have not yet received the update [14].

Every time a user moves, the user's HLR must initiate updates to all the replicas of the user's profile, and the network must carry the packets generated by the updates. Hence, the larger the number of replicas, the more work performed by the HLR and the network. Consequently, we expect the system may impose a limit on the number of zones at which a user's profile can be replicated, in order to limit the network activity and work performed by the user's home database.

An additional consideration is the higher storage requirement of databases in zones to store the replicated profiles. We expect that zones may impose a limit on the number of replicas maintained in their database, in order to bound storage requirements and guarantee fast lookup and update response times.

2.3 Preliminaries

We will develop our replication algorithm first with the following parameters; we will show how to compute the more complex of these parameters in Section 2.5.

- Let M be the number of zones and Z_j be the j^{th} zone for $j = 1, 2, \ldots, M$.
- Let p_j be the maximum number of profiles serviceable by the database associated with zone Z_j.
- Let N be the number of PCS users and P_i be the i^{th} PCS user for $i = 1, 2, \ldots, N$.
- Let $C_{i,j}$ be the expected number of calls made from zone Z_j to user P_i over a set time period T.
- Let U_i be the number of moves made by P_i over time period T.
- Let r_i be the maximum number of sites at which P_i's profile can be replicated.
- Let α represent the cost savings achieved when a local lookup succeeds as opposed to a remote lookup.
- Let β be the cost of updating a profile replica.

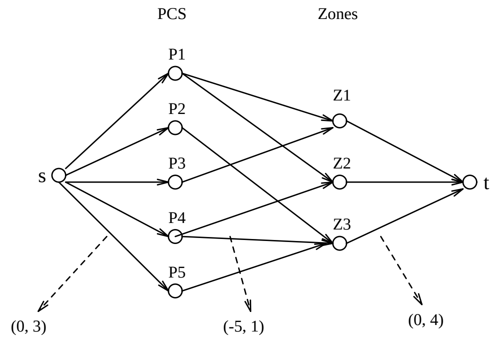

Figure 2: Example of a Flow Network.

We say that replicating a profile at a zone is *judicious* if the cost savings due to replication exceeds the cost inccured. Hence, using the parameters above, it is judicious to replicate P_i at Z_j if:

$$\alpha * C_{i,j} \geq \beta * U_i$$

Let $R(P_i)$ denote the *replication set* of user P_i, that is the zones at which P_i's profile is replicated according to the algorithm we now present.

2.4 Computing the Replication Plan

Construct the following *flow network* [9] $F = (V, E)$, where V and E are the vertices and edges in the flow network. In general, the edges in a flow network have two associated attributes: (*cost*, *capacity*).

1. $V \leftarrow \phi$, $E \leftarrow \phi$.
2. Add source s and sink t to V.
3. Add all P_i and Z_j to V for $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, M$.
4. Add to E directed edges from source s to all P_i with $(cost, capacity) = (0, r_i)$, and from all Z_j to sink t with $(cost, capacity) = (0, p_j)$.
5. For every $< P_i, Z_j >$ pair, if $\alpha * C_{i,j} \geq \beta * U_i$ (that is, it is judicious to replicate P_i at Z_j) then add an edge to E from P_i to Z_j with $(cost, capacity) = (\beta * U_i - \alpha * C_{i,j}, 1)$.

In Figure 2, we present as an example a small system with three zones and five users. We first discuss the capacity attributes of edges, and subsequently the cost attributes. The capacity attribute 3 on edge (s, P_5) denotes that P_5's profile can be replicated at no more than three zones. The capacity attribute 1 on edge (P_4, Z_3) indicates that P_4 should be replicated at most once in Z_3. The capacity attribute 4 on edge (Z_3, t) indicates that the database associated with Z_3 can store at most four profile replicas. The cost is captured on edge (P_4, Z_3), where -5 indicates that replicating P_4 in zone Z_3 will yield a net cost savings of five over not replicating. The cost attributes on edges (s, P_5) and (Z_3, t) are set to zero since they have no bearing on the system cost.

Computing the minimum-cost maximum-flow (henceforth called *min-cost max-flow*) [9] on flow network F finds an assignment of user profiles to databases in zones such that the number of useful replicas is maximized while the system cost is minimized. (Recall that in addition, each profile is guaranteed to be stored at the user's HLR.) We present a sketch of the classical min-cost max-flow algorithm [9].

Conceptually, an edge from u to v with capacity k corresponds to k *virtual edges* from u to v, each of capacity 1. An *edge reversal* on the edge from u to v means that one virtual edge from u to v is reversed to be directed from v to u. An *augmenting path* is a directed path along virtual edges from source s to sink t. The *cost* of a path is the sum of costs along its edges. One popular min-cost max-flow algorithm [9] is:

- Repeat until no more augmenting paths can be found:
  1. Find the augmenting path from s to t with least cost.
  2. Do an edge reversal on each virtual edge in this path.

When the above procedure terminates on our flow network (due to no more augmenting paths), we determine that P_i's profile is replicated at Z_j if there is a directed virtual edge from Z_j to P_i. That is $R(P_i) = \{Z_j \mid (Z_j, P_i) \in E\}$.

The min-cost max-flow algorithm guarantees the following.

1. **The number of replicated profiles at zones does not exceed the maximum serviceable capacity of their databases.**

   If p_j profiles have already been assigned to Z_j, then by the flow algorithm there must have been p_j edge reversals from Z_j to sink t. Since there were exactly p_j virtual edges from Z_j to t in the original flow network, no further augmenting paths that will assign more profiles to Z_j will be found from source s to sink t.

2. **The profile of a user is not replicated at more than the specified maximum number of sites of replication.**

   This is guaranteed by the structure of F with reasoning similar to the first case: not more than r_i augmenting paths can be found from s to P_i.

3. **The "system savings" is maximized.**

   Recall that only the edges from P_j to Z_i have costs. The final replication plan will have a cumulative cost of

   $$\sum_{i=1}^{N} \sum_{j=1, Z_j \in R(P_i)}^{M} \beta * U_i - \alpha * C_{i,j}$$

   The min-cost max-flow algorithm minimizes this cost by making it as negative as possible. This means that the system maximizes its savings by replicating the profiles at the indicated sites.

2.5 Computing Parameters

We now consider how to estimate the parameters used in our algorithm in a practical implementation of the approach. Jain et al. [15] present two efficient strategies for estimating the *local call to mobility ratio* (LCMR) of mobile users, used for making caching decisions. We propose a somewhat different model, more appropriate for replication.

Let $\sigma_{i,k}$ be the expected number of calls user P_i makes to user P_k in a given time period. We expect such information to be relatively stable [15]. Let $\lambda_{i,j}$ be the expected percentage of time user P_i spends in zone Z_j. Again we expect such information to be relatively stable [4]. Locational distributions $\lambda_{i,j}$ can be maintained by keeping track of the time elapsed in a zone between *registration* (when the user enters the zone) and a *deregistration* (when the user leaves the zone). Assuming that $\sigma_{i,k}$ and $\lambda_{i,j}$ are independent, we estimate the number of calls originating for a user P_i from zone Z_j to be:

$$C_{i,j} = \sum_{k=1}^{N} \sigma_{k,i} * \lambda_{k,j}$$

The number of updates U_i for a user P_i can be estimated by the average number of registrations (or deregistrations) performed by P_i in the given time period.

3 Dynamically Changing the Replication Plan

Our algorithm determines the best replication plan given fixed calling and mobility patterns, as specified in the parameters of Section 2.3. In a real system, these traffic patterns will change with time. Zones that did not have a replica of a user's profile might observe an increased lookup pattern for that user, and may want to replicate the profile. Other zones may want to drop replicas if the replication is no longer judicious. When a traffic pattern τ changes from τ_{old} to τ_{new}, one naive approach is to recompute the entire min-cost max-flow based algorithm using τ_{new}. Clearly this approach can be expensive in a large system. In Sections 3.1 and 3.2, we present an incremental min-cost max-flow algorithm that adapts to changes in traffic patterns.

3.1 Incremental Max-Flow

Let $F(\tau_{new})$ denote the flow network for traffic pattern τ_{new}; similarly for $F(\tau_{old})$. In this subsection, we consider how to incrementally compute the max-flow (but not necessarily min-cost) of $F(\tau_{new})$, given that we have already computed the min-cost max-flow for $F(\tau_{old})$.

When the traffic patterns change, edges may be inserted and deleted from the flow network because they become judicious, or because they are no longer judicious. For instance, suppose user P_i starts to receive numerous calls from users in zone Z_j, or moves less often. Then the edge from P_i to Z_j may become judicious, even though it was not judicious earlier. Similarly, an edge between P_i and Z_k may be deleted, if it is no longer judicious.

When an edge is deleted, the key idea is to try to retain the maximum flow if possible. Effectively, we try to retain the maximal use of the available databases by replicating another user's profile when a user's profile is dropped. There are three cases to consider:

1. **The deleted edge is a forward edge from P_i to Z_j.**

   In this case the solution remains unchanged, because this corresponds to Z_j not having been chosen as a site of replication for P_i's profile.

2. **The deleted edge is an edge from Z_j to P_i.**

   This edge indicates that P_i had been replicated at Z_j. Two subcases must be considered.

   (a) **Satisfiable "Vacant Slot":**

   In our flow network, we can find an augmenting path from P_i to Z_j. We reverse the edges on the augmenting path, compensating for the loss of one unit of flow through the deleted edge. By "pushing" one new unit of flow from P_i to Z_j, we maintain the maximum flow.

   (b) **Unsatisfiable "Vacant Slot":**

   Suppose we cannot find an augmenting path to satisfy the "vacant slot" left by Z_j. This could happen if there are no more judicious profiles for replication, or if users whose edges are judicious are already replicated at their maximal number of sites. In this case, the flow from source to sink is reduced by one unit, but this flow is the maximal possible on the new flow network. For "bookkeeping", we still need to reverse one virtual edge each from s to P_i and from Z_j to t.

When a new edge is introduced into the flow network from a user P_i to a zone Z_j, we attempt to increase the net flow from the source to the sink if possible. This is achieved easily by finding a new augmenting path from s to t using the new edge (if such a path exists), and performing edge reversals on the new path.

3.2 Incremental Min-Cost Max-Flow

In the previous section, we showed how to obtain a max-flow solution for τ_{new} given the min-cost max-flow solution for τ_{old}. We now show how to incrementally compute the min-cost max-flow solution for τ_{new} given the max-flow solution for τ_{new}. This is achieved by adapting the *cycle canceling algorithm* [1]. Briefly, the cycle canceling algorithm works as follows.

1. Compute a max-flow of the flow network.

2. Repeat until no more negative cycles are found:

   - Find negative cycles through the sink and perform edge reversals of edges in the cycle.

Intuitively, the cycle canceling algorithm finds a max-flow solution in the first step, and then tries to improve the cost of the max-flow solution by finding negative cycles through the sink. By pushing flow through these negative cycles, the flow to the sink is maintained: when no more negative cycles can be found, we have reached min-cost, and the algorithm terminates.

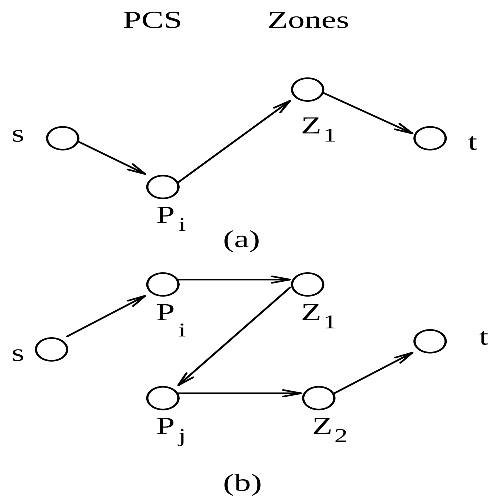

Figure 3: Possible augmenting paths.

Since we already have an incremental max-flow solution from the previous section, we merely bypass the first step and find negative cycles until we reach the min-cost max-flow solution. Each negative cycle corresponds to reassigning user profiles to zones so that the system savings is increased.

4 Graceful Evolution of Replication

The incremental algorithms presented in the previous section allow us to obtain an optimal replication plan for a new traffic pattern. However, it may be expensive to evolve to the new replication plan. If the traffic patterns change substantially, the new replication plan may require numerous changes in replication sites for profiles. Moving all the profiles may be costly, especially if the profiles also contain call forwarding information, quality of service parameters, authorization information, etc. In this section, we propose two modifications to our incremental algorithms that evolve the replication plan taking the cost of evolution into account in addition to other factors.

4.1 Tempered Min-Cost Max-Flow

Intuitively, when we augment flow along augmenting paths as described in Section 3.1, we either (a) perform a direct assignment of a profile to a database or, (b) perform an assignment of a profile to a database by displacing another profile to an alternate database. For instance, in Figure 3(a) we see a direct assignment of P_i's profile to Z_1's database. In Figure 3(b), we see an indirect assignment of P_i's profile to Z_1 that results from displacing P_j from Z_1 to Z_2. The length of the augmenting path specifies the number of reallocations of profiles. Hence, when choosing a min-cost augmenting path A during our min-cost max-flow computations, we can factor in the cost of reassignments as:

$$cost = \gamma_1 * \sum_{e \in A} cost(e) + \gamma_2 * |A|$$

where γ_1 and γ_2 are scaling factors representing the relative costs of the two terms in the equation. The first term is the savings in cost due to a reassignment of replicas, while the second term "tempers" the savings by the evolution cost. The second term can be substituted by a more complex formula that captures the actual cost in terms of transmitting profiles from one site to another. Using the above cost definition, we factor the complete cost of reassignment into the augmenting path computation. The same principle also can be used for negative cycles. Note that if $\gamma_2 = 0$, this algorithm degenerates to the min-cost max-flow solution as defined in Section 3.2.

In the traditional min-cost max-flow algorithm, min-cost paths are located using shortest path algorithms [1]. With the new definition of cost, this is no longer possible since the cost of reaching a vertex (from a vertex labeling perspective [9]) is no longer a constant: it depends on the length of the path chosen. Hence, to compute min-cost paths we will have to resort to branch-and-bound techniques [9], which may be exponential at worst. However, if γ_1 and γ_2 are reasonably large, we can find min-cost paths (and cycles) efficiently with pruning.

4.2 Evolution with Mean Cycles

We consider a second scheme to factor in evolution cost with some advantages discussed below. The second scheme uses the *minimum mean cycle canceling algorithm* [1]. This algorithm is a special case of the cycle canceling algorithm seen earlier. It operates by augmenting flow along cycles with the minimum *mean cost* in the flow network: the mean cost of a directed cycle W is defined [1] to be:

$$\sum_{(i,j) \in W} cost_{i,j} / |W|$$

Intuitively, the benefit of moving from one solution to another is reduced as the number of replica reallocations increases.

Note that we are no longer able to factor in the true cost of network transfers due to reallocation of replicas, like we could in Section 4.1. We now can only approximate the reallocation cost with the divisor $|W|$. But using the minimum mean cycle canceling algorithm has two advantages that make it useful for dynamic evolution: (1) This algorithm has a provable polynomial bound [1] as opposed to the worst-case exponential bound of the algorithm in Section 4.1. (2) In the mean cycle algorithm, we always augment flow along the minimum mean cycle. Intuitively, we always choose to augment along the cycle that gives us maximal[2] benefit with the minimal number of reassignments. We can use this property in a real network in order to perform the reallocation with maximal benefit and minimal number of reassignments first. The reallocation with lesser benefit that induce several reassignments can be performed later, when the network can handle the associated high amounts of data transfer.

[2] Recall that in the cycle-canceling algorithm we choose negative cycles, but not necessarily the min-cost negative cycle.

Case	Scheme	Lookups (bytes)	Updates (bytes)	Time (secs)
1	HLR	$C_{i,j} * (q + a)$	0	N
	Caching	$C_{i,j} * (q + a)$	0	$L + N$
	Replication	$C_{i,j} * (q + a)$	0	$L + \epsilon + N$
2	HLR	$C_{i,j} * (q + a)$	0	N
	Caching	0	0	L
	Replication	0	0	$L + \epsilon$
3	HLR	$C_{i,j} * (q + a)$	0	N
	Caching	$(1 - \rho) * C_{i,j} * 2 * (q + a)$	0	$L + (1 - \rho) * 2 * (L + N)$
	Replication	0	$u * U_i$	$L + \epsilon$

Figure 4: Comparison of different schemes.

5 Analysis of Replication and Caching

In this section, we provide an analytical comparison of the caching schemes described in [12, 15], our replication scheme, and a simple HLR scheme (recall Section 1) in terms of the number of bytes generated on the network and the location lookup time. This analysis gives us some insights that we shall use in the subsequent experimental section. The lookup algorithm in [12, 15] is:

1. If P_i's profile is not cached in Z_j, lookup at HLR of P_i.
2. If P_i's profile is cached in Z_j, forward call to location Z_k specified in profile.
   (a) If P_i is located in Z_k (valid cache entry), then complete call.
   (b) If P_i is not located in Z_k (invalid cache entry), lookup at HLR of P_i, forward call to new location, and update cache entry in Z_j to new location of P_i.

To ensure a fair comparison of replication and caching, we assume an abstract boolean-valued function *store*(P_i, Z_j). This function determines whether it is worthwhile to store P_i's profile at Z_j, for both the caching and replication schemes. We expect *store*(P_i, Z_j) to be some function of the traffic patterns and storage constraints. This function could use LCMR [15], our notion of judiciousness defined in Section 2.1, or some other model.

We compare the network cost in number of bytes due to lookups and updates, and the time required for location lookup. We assume for the analysis that the database in a zone is partitioned: the HLR component stores profiles of users who have that zone as their home location, and the excess capacity is used to store profiles of users who have a different home-location. We use the following parameters.

1. Let L be the time to service a lookup locally, and let N ($N >> L$) be the network latency for a remote lookup.

2. Let ϵ be the increase in response delay for a lookup submitted to a database that stores replicas as opposed to cache entries (since caches do not need to process updates).

3. Let q be the size (in bytes) of a remote lookup request and a be the size (in bytes) of the lookup result.

4. Let u be the size (in bytes) of a remote update request (specifying the new location of a user in the replication scheme).

The number of remote updates in caching is always zero, since in the caching scheme the entry is merely invalid if a user has moved, as described above.

In the following analysis of caching and replication, we consider the number of bytes generated and the response time from the view of a single database in time period T. Let a user in zone Z_j place a call to user P_i. The cases to consider are:

1. **The profile of P_i is not stored at zone Z_j.**

   In this case the number of bytes generated by both schemes is $(q + a)$. The response time for the caching scheme is $(L + (N + L))$ while the replication scheme requires $(L + \epsilon + (N + L))$. Both schemes have to lookup the local database before executing the remote lookup.

2. **The profile of P_i is stored at site Z_j and P_i has not moved in time period T.**

   In the caching scheme, this means that the cache entry for the user is valid. The number of remote lookups is zero, and the time to answer the lookup is L. In the replication scheme, the number of bytes due to updates is zero since the callee did not move. The number of remote lookups is also zero, and the time to satisfy the lookup is $(L + \epsilon)$.

3. **The profile of P_i is stored at site Z_j, and P_i has moved U_i times in a given time-period since the last call for P_i from Z_j.**

   In general, the cache entry will be invalid. However, there could be some "lucky hits", where the call goes through to the current location of P_i because P_i came back to the zone indicated in his profile (even though he may have visited several other zones meanwhile). Let ρ be the probability of such a "lucky hit". If $C_{i,j}$ is the number of calls from Z_j to P_i in time period T, the number of remote lookup bytes is $(1 - \rho) * C_{i,j} * ((q + a) + (q + a))$ (to reach the last known location of P_i, then to reach the HLR of P_i). The expected time for lookup is $\rho * L + (1 - \rho) * (L + (L + N) + (L + N))$. In replication, the number of remote lookups is zero since we always maintain up-to-date copies. The total bytes due to remote update packets is $U_i * u$, and the time to lookup the local database is $L + \epsilon$.

From the above case-by-case analysis, summarized in Figure 3 along with values of traditional HLR for comparison, we see that our replication scheme has comparable response time to caching in the first two cases, and significantly lower response time in the third case. Caching and replication schemes generate the same number of bytes in the first two cases and differ only in the third case. In the third case, when we divide the total bytes due to caching by the total bytes due to replication, we see that the replication scheme has a lower number of packets when

$$\frac{C_{i,j}}{U_i} \geq \frac{u}{2*(1-\rho)*(q+a)}$$

If we assume maximal packing of data (that is, the minimum number of bits are used to represent data) we can assume that $q = \lceil log_2 N \rceil$ and $u = a = \lceil log_2 N \rceil + \lceil log_2 M \rceil$. (The remote lookup has to specify the unique ID of the callee, and the answer and update should contain the ID of the callee and his location.) The above equation reduces to:

$$\frac{C_{i,j}}{U_i} \geq \frac{\lceil log_2 M \rceil + \lceil log_2 N \rceil}{2*(1-\rho)*(2*\lceil log_2 N \rceil + \lceil log_2 M \rceil)}$$

We expect the number of users (N) to far exceed the number of zones (M) in any practical system ($\lceil log_2 N \rceil >> \lceil log_2 M \rceil$). Also, a "lucky hit" in the caching scheme requires that there had been no previous call to P_i from Z_j when P_i was in a different zone. For this to hold, the number of calls made to P_i must be very small compared to the rate at which P_i moves. In such a case, metrics such as LCMR would not have stored the profile at Z_j in the first place. Hence, in practice we expect ρ to be negligible. Under these conditions, we see that our replication scheme generates fewer network bytes compared to caching when the LCMR ($C_{i,j}/U_i$) of Z_j is ≥ 0.25. Jain et al. [15] suggest a minimum value of 4 to 5 for their LCMR threshold for caching. For such thresholds, we see that our replication scheme not only has lower lookup time, but also generates fewer bytes in the network. In Section 7 we will quantify through simulation experiments the actual savings of replication over other location lookup strategies with respect to several database and network performance measures.

6 Computing and Maintaining Replicas

In this section, we propose one way that our replication strategy could be implemented in a PCS system at, say, a nationwide level. We consider three critical issues: (1) what the granularity of replication should be; (2) who should compute the replication plan; (3) who should manage the replicas. We use the United States telephone system as a reference point. Telephone numbers are divided into about 200 geographical zones identified by unique area codes. We suppose that each area code is serviced by a database, where that database may be at the root of a local hierarchy of databases, each servicing a smaller zone [24].

We propose that profile replication should occur at the granularity of area codes. Thus, an area code level database, in addition to storing profiles of users in that area code, stores profile replicas of users frequently called from users in that area code. If the area code database is at the root of a hierarchy, databases in the lower levels of the hierarchy service "sub-zones" of the area code zone. When a call is initiated from a sub-zone, if the profile for the callee is not found in the associated lower-level database, the request propagates up the hierarchy. If the request reaches the root and the callee's profile is replicated at that database, the location information is found. Otherwise, the profile request is sent to the HLR of the callee.

This replication granularity has two advantages:

1. Area code regions are large enough that the number of area code crossings is expected to be low when

compared with the number of calls generated. Hence, the number of updates generated due to the callee's mobility is small, and the benefit of replication is high since a caller is likely to make multiple calls before leaving his current zone.

2. Area code regions are small enough that lookups posed within a single database hierarchy (such as requests that propagate to the root before finding the profile replica) are still relatively inexpensive. For example, if a caller C_1 in Los Angeles makes several calls to a callee C_2 in New York, the replication strategy will most likely guarantee that C_2's profile is found in the database hierarchy associated with C_1, avoiding the cross-country lookups generated by, say, the traditional HLR/VLR strategy.

In our current algorithm, we assume a centralized site that periodically executes the min-cost max-flow algorithm, then propagates the replication plan to the area code level databases. Since our replication granularity is relatively coarse, we expect that dynamic reallocation will need to be performed only every few hours at most. Consequently, the centralized site should not become a bottleneck: the reallocations are infrequent, and the number of replication sites is relatively small.

Finally, as discussed earlier, we suggest that each user's HLR database should keep track of the user's profile replicas. When the user crosses area codes, the HLR is notified, so it can propagate the change in location to all profile replicas.

7 Experiments

In our simulations, we consider how the performance of our replication approach compares with other location lookup algorithms when used in a wireless infrastructure for a relatively large geographical area. We performed our simulations on a geography that accurately models the San Francisco Bay Area. The Bay Area consists of nine counties and is serviced by four area codes. A map of the Bay Area is provided in Figure 5 for the reader's reference. (For a more detailed description see [17].) Figure 6 is an overlap map depicting the relationship between our simulation model and the physical geography. Ninety registration areas are represented as polygons in this figure, with higher-level regions corresponding to the four area codes. Dots represent databases and lines connecting dots represent network links.

In our simulation, we populated registration areas with 100,000 users based on 1990 census information from [23]. We divided our user population into 41% commuters and 59% non-commuters (derived from Bay Area vehicular traces and published statistics from Europe and the United States [19]). Using traffic volume statistics from [19], we estimated movement between area codes and fine-tuned our simulation parameters to produce similar large-scale movement behaviour (see [17] for further details).

We implemented our replication algorithm as well as the caching algorithm of Jain et al. [15, 12] as extensions on the pure HLR (rather than HLR/VLR which, as mentioned earlier, is a limited form of caching) location lookup scheme. For both algorithms, we allowed a maximum of four replicas for each user in addition to the profile at the user's HLR. (This limit allowed our simulations to run primarily in main memory and did not affect our findings.) As noted earlier, the caching algorithm does not cleanly incorporate storage constraints in its decision process. Hence, for a fair comparison of the caching and replication algorithms, we assumed no restrictions on the database capacities. We used the LCMR measure as the decision function for choosing sites that are judicious for profile caching or replication.[3] For caching, the LCMR threshold was set to 5 as recommended in [12]. For replication, the LCMR threshold was set to 0.25, since that is the lower bound for replication to outperform caching in network bandwidth requirement (recall Section 5). We chose the granularity of replication and caching to be at the level of the 90 registration areas, rather than the four area codes, since we believe this to be an appropriate granularity of replication for a geographical area of the scale of the Bay Area (while area codes may be more appropriate for a nation wide infrastructure; recall Section 7). We also implemented pure HLR and the HLR/VLR lookup strategies for comparison.

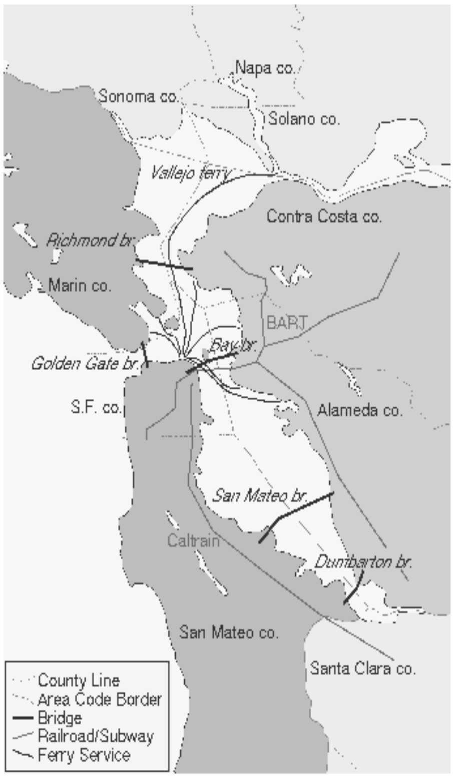

Figure 5: San Francisco Bay Area.

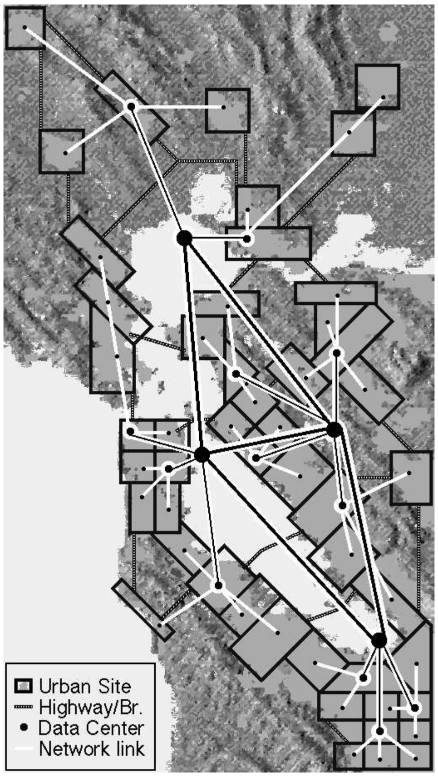

Figure 6: Simulation and Network Topologies.

We chose to focus our simulations to answer the following questions, which we believe are the most important.

1. What is the latency involved when a caller makes a call?
2. How do the various lookup strategies compare for critical performance measures such as database and network loads?

[3] As noted earlier, there are a variety of possible measures for judiciousness. We have chosen the LCMR measure since it appears to be the most popular in current literature.

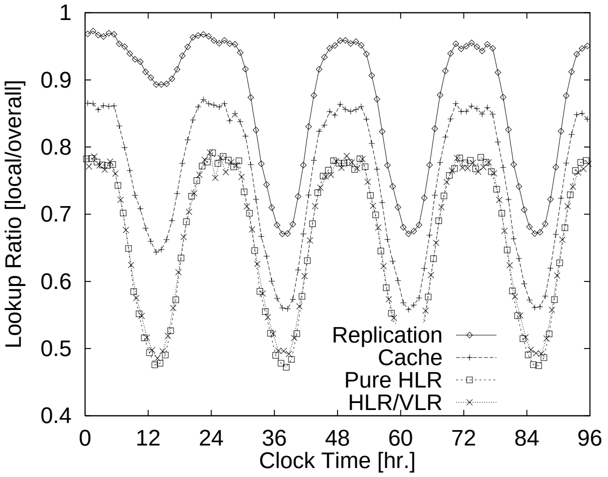

Figure 7: Percentage of Calls Serviced by Local Lookups [4 day Simulation].

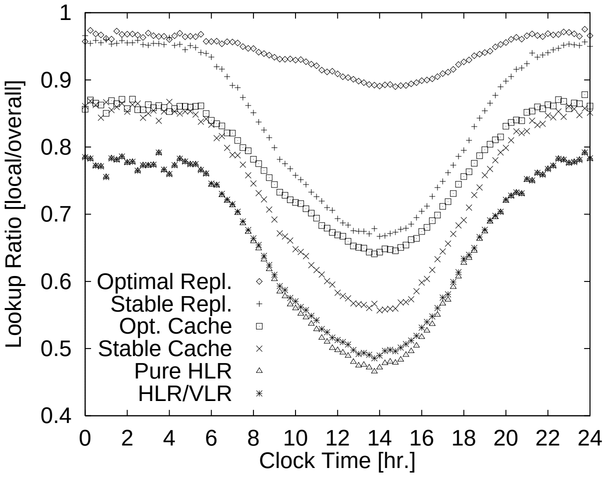

Figure 8: Percentage of Calls Serviced by Local Lookups [1 day Simulation].

3. How sensitive are replication and caching to the accuracy of expected calling and mobility patterns?
4. How often should we change the caching or replication plan, and what is the corresponding impact on system performance?

Other important questions that we have not considered in this paper, but intend to consider as future work, include: (1) How much information should be used to maintain LCMR estimates? (2) How does the performance of replication vary when the number of maximum profile replicas is varied? (3) Should all users have the same maximum number of profiles, or should the maximum be chosen on a per-user basis? (4) How does varying capacity of the databases affect the performance of replication?

To answer the questions we posed, we simulated the replication scheme against caching, pure HLR, and HLR/VLR schemes for a 5-day period; we used the first 24 hours as a "warm-up" period to stabilize the calling and mobility patterns in our simulations. In our experiments, the replication and caching schemes had perfect LCMR information for the first day (after the warmup period), and produced replication and caching plans that were "optimal" for that day. The same replication plan was then used for the three subsequent days, even though the calling and mobility patterns changed significantly. This was done to evaluate the performance of caching and replication for both accurate and inaccurate LCMR estimates. Each 24-hour simulation for the different lookup strategies outlined above generated 370,616 move and 2,357,000 call events, and averaged about eighty minutes to simulate 120 hours of traffic on an HP 9000/755/99 workstation with 512 MB RAM.

We chose to compare the latency incurred during lookups in various algorithms by comparing the percentage of calls serviced by queries on local databases for location lookup. We report this key measure for the different algorithms for the 4-day simulation in Figure 7. Notice that "optimal" replication (the first 24 hours) services about 90–98% of calls in a day using profiles in local databases. In other words, with the replication scheme less than 10% of calls lead to remote location lookup queries, as opposed to the pure HLR and HLR/VLR schemes in which up to 53% of queries need to access remote databases. This corresponds to up to a five-fold reduction in the number of remote queries. Also notice that replication and caching are stable across the 2nd, 3rd, and 4th days, in spite of inaccurate LCMR estimates. Here replication still services 66 – 96% of calls using local databases. This corresponds to up to a two-fold reduction in the number of remote queries, still a significant performance savings over pure HLR and HLR/VLR. We will consider how performance degrades as LCMR estimates become progressively more inaccurate in greater detail below. One surprising observation is that HLR/VLR performs very closely to pure HLR.

Henceforth, we report various performance measures by combining the 4-day simulation into a representative one-day plot by choosing the plots for the stable replication and caching from the second day, and for the rest from the first day. Replication and caching algorithms that have perfect LCMR estimates (plots from the 1st day) are termed "optimal", while those that have inaccurate estimates (from the 2nd, 3rd and 4th days) are termed "stable." For example, we see in Figure 8 the representative one day version of Figure 7.

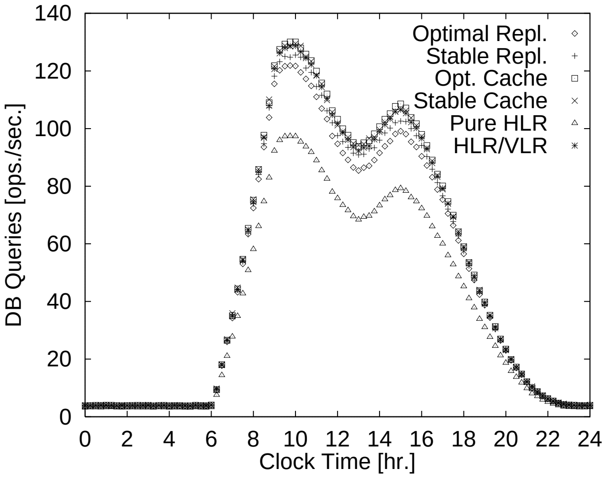

Figure 9: System-Wide Database Lookups.

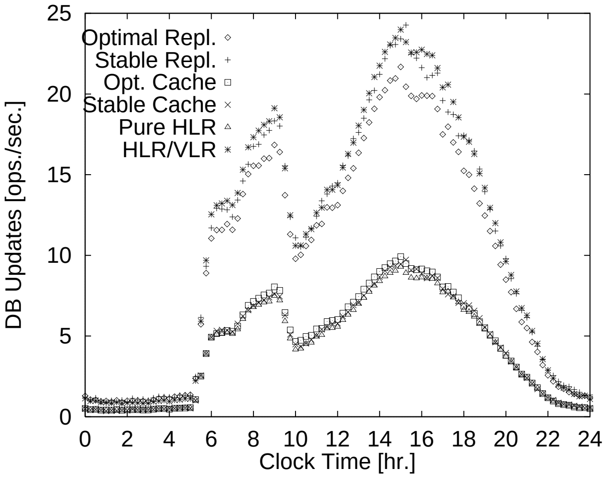

Figure 10: System-Wide Database Updates.

Figures 9 and 10 show the database requirements in number of lookups and updates, respectively, that need to be serviced per second cumulatively by all databases in the system during the 24-hour day. These counts include both local and remote database accesses. As expected, the replication scheme has one of the highest update rates. Somewhat surprising is that HLR/VLR has about the same number of updates as replication. This is because in HLR/VLR up to 3 updates are initiated when a user moves: one to the user's HLR, and one each to the database servicing the previous and new locations if they do not correspond to the HLR. In replication between one and $(k+1)$ updates are initiated when a user moves: one to the user's HLR, and one each to the k potential profile replicas. Since the LCMR has a threshold to determine the judiciousness of potential profile replicas, most users have relatively few profile replicas. Hence the update performance of HLR/VLR is close to that of replication. In all cases the performance requirements (for lookups plus updates) are well within current database technology [22].

In Figures 11 and 12 we show the number of network messages sent per second between databases (for

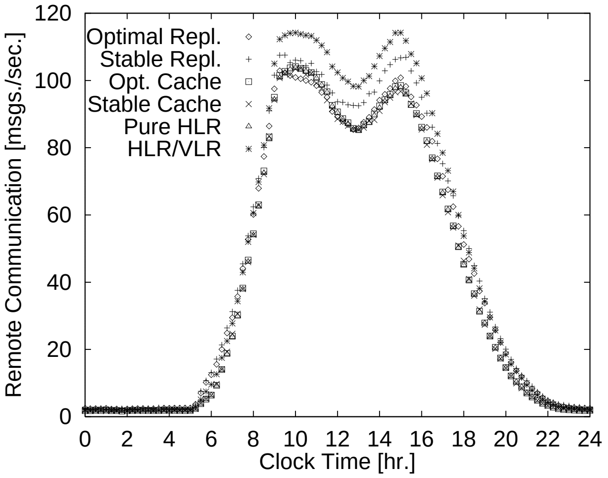

Figure 11: Average Number of Network Messages.

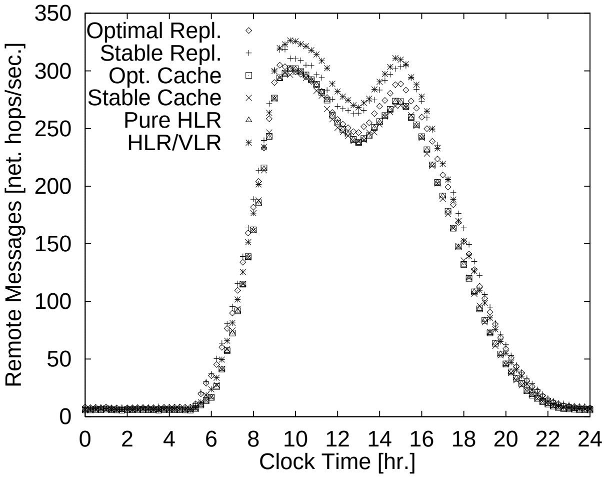

Figure 12: Average Number of Network Message Hops.

both lookups and updates), and the actual number of network hops for these messages according to our topology in Figure 6. To be fair to the HLR schemes, we included the messages required for the replication and caching schemes to check if the callee is busy or not [18]. Here we see that optimal replication performs very well and requires less bandwidth than that required by HLR/VLR, even at peak times during the day when the movement patterns are high. If the messages to check if a callee is busy are not required (in the case of call forwarding and call waiting, say), replication requires only about 25% of the bandwidth required by HLR/VLR. A similar behavior holds for the number of network hops.

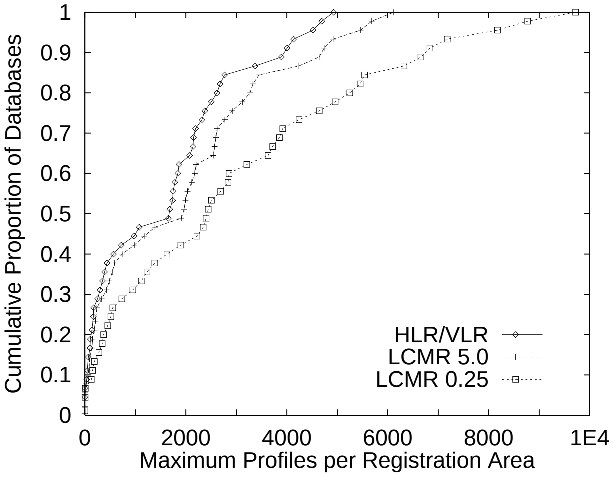

Figure 13: Cumulative Distribution of Maximum Required Storage for Profiles.

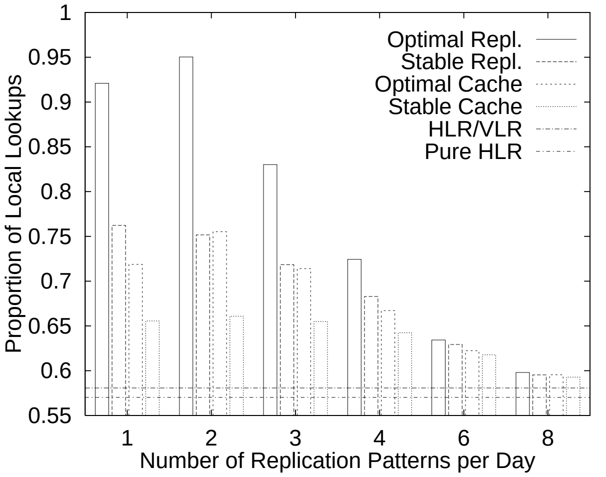

Figure 14: Change in Performance with Number of Reallocations per Day.

In Figure 13 we report the maximum storage requirements of the databases associated with the registration areas. For instance in HLR/VLR, 100% of databases store no more than 5000 profiles. We see that the maximum database storage required for replication is nearly twice that for HLR/VLR due to the additional profile replicas in databases. However, we believe that with decreasing disk and memory prices this is not a significant problem.

For the final experiment, we studied how often replication and caching schemes should be recomputed. One would expect that the more frequently the replication plan is updated, the higher the expected performance of replication, and the higher the cost of computing and performing the reallocation. To study the rate of change in performance with more frequent reallocations, independent of the cost of performing reallocations, we set the cost of performing reallocations to zero. In Figure 14 we show the percentage (averaged for 24 hours) of locally serviced calls for the various algorithms as the number of reallocations performed per day is varied. For instance, performing six reallocations each day corresponds to performing a new reallocation every four hours. For the stable replication and caching, estimates for the new day are made based on the LCMR estimates from the previous day. Surprisingly, we see that the performance of the offline caching and replication peaks when the number of reallocations is twice a day, and then deteriorates if the number of reallocations is increased. We observed this to be the result of two factors:

1. When the time window between two reallocations is too small, the LCMR values are relatively small and hence very few sites are judicious for replication and caching.
2. In a typical day, there is large scale movement of users once in the morning, and once in the evening. By computing two replication plans for a day, we manage to capture the right LCMR estimates for both movements.

From the above graphs, we see that replication provides significant potential savings in terms of:

1. **Low Lookup Latency**: Optimal replication "converts" up to 81% of (slow) remote queries in HLR and HLR/VLR into (fast) lookups on local databases. Stable replication, in spite of its inaccurate LCMR estimates, converts up to 38% of remote queries in HLR and HLR/VLR into local lookups.
2. **Reduced Bandwidth Requirements**: When additional messages need to be sent to determine if the callee is busy, replication requires 15% less bandwidth compared to HLR/VLR. When the additional messages are not required, replication requires only about 25% of the bandwidth of HLR/VLR and pure HLR.

These significant savings come at the price of additional storage requirements for replication (up to 100% more). Surprisingly the database requirements in the number of lookups and updates are comparable for the replication and the HLR/VLR schemes. We also see that replication and caching appear stable even with inacccurate LCMR estimates. We observed that computing one or two reallocations per day is sufficient to handle 90–95% of the calls through local lookups.

8 Other Applications

8.1 Non-HLR Lookup Schemes

Recently, there has been some consideration of *location-independent* numbering schemes [16, 24]. In such schemes, users no longer have a fixed home location (HLR) identified by their number. Location-independent schemes have the advantage that numbers can be kept "for life", without needing to be changed or incurring lookup overhead if the user moves permanently to a new home location. The disadvantage of location-independent numbers is that the second step of the lookup algorithm (recall Section 1) must search for the appropriate database rather than go directly to the HLR. Our replication scheme can be used in such environments to increase the probability of finding a profile locally. In fact, we expect that our replication framework will be particularly useful here, since searching for the database to find the user's profile may be quite expensive.

8.2 Service Information Replication

We have established a framework for replicating location information of users for fast lookup. In this section, we outline a second application of our dynamic replication framework that can also be useful in a PCS environment.

In addition to location information, associated with every user is *service* information such as call blocking, call forwarding, favorite e-mail address, etc. and *quality of service (QOS) requirements* such as minimum channel quality, acceptable bandwidth for mobile computers, etc. As the number of services increase in PCS systems, we expect the service information in profiles to grow fairly large. Currently in the IS-41 and the GSM standards [18], service information may be replicated in the VLR, or it may always be fetched from the user's HLR.

Generally, QOS and service information is required in the zone at which a call is received, not where it is initiated (unlike location information). We can use a very similar dynamic replication framework and algorithms to that presented here for replicating QOS and service information in zones frequently visited by the user. Replication here will reduce the number of packets transmitted from the HLR, or will make the call handoff less expensive, since the new zone no longer necessarily needs to obtain QOS and service information from the previous zone. We expect this information to change infrequently, thereby making the case even stronger for replication.

9 Conclusion and Future Work

In this paper we proposed enhancing current HLR and HLR/VLR schemes with more general replication of user profiles for faster location lookup. We considered how to replicate user profiles given capacity constraints in the databases and network and fixed calling and mobility patterns of users. We subsequently extended our framework to handle dynamic changes in user calling and mobility patterns. We also presented two schemes for replication plans that evolve over time gracefully to minimize network overhead. We have simulated our replication scheme along with several other location lookup strategies on a real model of the San Francisco Bay Area with realistic user calling and mobility patterns, and reported on important performance measures. We noted that our replication algorithm has a significant reduction in call latency and network bandwidth over HLR/VLR assuming both accurate and inaccurate LCMR estimates, at the cost of increased database storage requirements. We also observed that our replication algorithm is stable even while using inaccurate LCMR estimates. We also showed that computing one or two replications per day is sufficient to service 90-95% of the calls locally.

Since the performance improvements due to replication and caching are somewhat dependent on accurate LCMR estimates, we are working on efficient and stable schemes to estimate LCMRs accurately. Our replication framework can be made distributed for certain special cases such as infinite capacity of databases, or if there are no limits on the maximum number of sites for profile replication. We are, however working on a distributed version of our general profile replication algorithm based on the Auctioning algorithm [6]. We are also considering expanding the scope of our simulations for larger user populations, different network topologies, and to study the impact of varying maximum number of user profile replicas and capacity of databases. Another promising area we are looking at is how to choose when to perform the profile reallocations in a given day.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows.* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] V. Anantharam, M. L. Honig, U. Madhow, and V. K. Wei. Optimization of a database hierarchy for mobility tracking in a personal communications network. *Performance Evaluation*, pages 287–300, 1994.

[3] B. Awerbuch and D. Peleg. Concurrent online tracking of mobile users. *SIGCOM*, 1991.

[4] B. R. Badrinath, T. Imielinski, and A. Virmani. Locating strategies for personal communication networks. In *Workshop on Networking for Personal Communications Applications.* IEEE GLOBECOM, December 1992.

[5] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali. *Linear Programming and Network Flows.* John Wiley and Sons, New York, 1990.

[6] D.P. Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14(105-123), 1988.

[7] S. Ceri, G. Martella, and G. Pelagatti. Optimal file allocation in a computer network: A solution method based on the knapsack problem. *Comput. Networks*, 6, 1982.

[8] S.K. Chang and A.C. Liu. File allocation in a distributed database. *Int. J. Comput. Inf. Sci.*, 11(5), 1982.

[9] T.H. Cormen, C.L. Leicerson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press and McGraw-Hill, New York, 1990.

[10] L.W. Dowdy and D.V. Foster. Comparative models of the file allocation problem. *ACM Computing Surveys*, 14(2), June 1982.

[11] M.K. Fisher and D.S. Hochbaum. Database location in computer networks. *Journal of the ACM*, 27(4), October 1980.

[12] H. Harjono, R. Jain, and S. Mohan. Analysis and simulation of a cache–based auxiliary location strategy for PCS. In *IEEE Conference on Networks and Personal Communications*, 1994.

[13] J.S.M. Ho and I.F. Akyildiz. Local anchor scheme for reducing location tracking costs in PCN. In *1st ACM International Conference on Mobile Computing and Networking (MOBICOM'95)*, pages 170–180, Berkeley, California, 1995.

[14] R. Jain and Y. Lin. An auxiliary user location strategy employing forwarding pointers to reduce network impacts of pcs. In *International Conference on Communications (ICC'95)*, 1995.

[15] R. Jain, Y.-B. Lin, C. Lo, and S. Mohan. A caching strategy to reduce network impacts of PCS. *IEEE Journal on Selected Areas in Communications*, 12(8):1434–44, October 1994.

[16] J. Jannink, D. Lam, N. Shivakumar, J. Widom, and D. C. Cox. Data management for user profiles in wireless communications systems. Technical report, Stanford University, Computer Science Dept., October 1995. ftp://www-db.stanford.edu/pub/jannink/1995/profile/.

[17] D. Lam, J. Jannink, D. C. Cox, and J. Widom. Modeling location management for Personal Communication Services. Technical report, Stanford University, Computer Science Dept., October 1995. ftp://www-db.stanford.edu/pub/jannink/1995/model/.

[18] S. Mohan and R. Jain. Two user location strategies for personal communications services. *IEEE Personal Communications*, pages 42–50, First Quarter, 1994.

[19] 1990 commute summary. Metropolitan Transportation Commission, Lotus 123 format spreadsheet, August 1990. S.F. Bay area transportation measurements.

[20] M.T. Ozsu and P. Valduriez. *Principles of Distributed Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[21] N. Shivakumar and J. Widom. User profile replication for faster lookup in mobile environments. In *1st ACM International Conference on Mobile Computing and Networking (MOBICOM'95)*, pages 161–169, Berkeley, California, 1995.

[22] Transaction processing benchmarks. http://www.ideas.com.au/bench/bench.htm, 1995. Monthly TPC-A, TPC-B & TPC-C benchmark results.

[23] 1990 U.S. census demographic summaries. UpClose Publishing, http://www.upclose.com/upclose/, 1994. Population of nine S.F. Bay area counties.

[24] J. Z. Wang. A fully distributed location registration strategy for universal personal communication systems. *IEEE Journal on Selected Areas in Communications*, 11(6):850–860, August 1993.

[25] O. Wolfson and S. Jajodia. Distributed algorithms for dynamic replication of data. In *Proceedings of the Symposium on Principles of Database Systems*, San Diego, CA, 1992.