Efficient and Flexible Location Management Techniques for Wireless Communication Systems *

Jan Jannink, Derek Lam, Narayanan Shivakumar, Jennifer Widom, Donald C. Cox

Departments of Computer Science & Electrical Engineering
Stanford University, Stanford, CA 94305

jan, shiva, widom}@db.stanford.edu, {dlam, dcox}@wireless.stanford.edu

Abstract

We consider the problem of managing the information required to locate users in a wireless communication system, with a focus on designing and evaluating location management techniques that are efficient, scalable, and flexible. The three key contributions of this paper are: (1) A family of location management techniques, *HiPER* (for *Hierarchical ProfilE Replication*), that efficiently provide life-long (non-geographic) numbering with fast location lookup; (2) *Pleiades*, a scalable event-driven wireless system simulator with realistic calling and mobility patterns derived from several months of real traffic traces; and (3) multi-day simulations comparing our proposed location management techniques with current and previously proposed techniques on a realistic geographical and network topology.

1 Introduction

In a wireless communications system, mobile users place and receive calls through a wireless medium. Users are located in system-defined *zones*, that correspond to bounded geographical areas. When a user places a call, the wireless network infrastructure must perform several tasks, including authenticating the caller, locating the callee, and routing the call to the *base-station* in the same zone as the callee.

We consider the problem of managing the information required to locate wireless users who move from zone to zone. When user A calls user B, the *location lookup* problem is to find the location of callee B within "reasonable" time bounds, in order to complete the call setup between A and B. In current cellular standards such as GSM and IS-41 [EIA91], each user has a *home* database termed the *Home Location Register (HLR)* which maintains the current location of the user as part of the user's *profile*. When user A calls user B, the wireless infrastructure initiates a potentially remote query to the *HLR* of B. Since performing remote lookups may be slow due to high network latency, current systems augment the above *pure HLR* scheme by maintaining *Visitor Location Registers (VLR)* in every zone. *VLRs* store copies of profiles of users not at home and currently located in that zone. The modified lookup strategy for this *HLR/VLR* scheme is to look for a copy of the callee's profile in the local *VLR* before performing a remote query to the callee's *HLR*.

One desirable feature that current protocols (such as IS-41 and GSM) fail to offer is *Life-Long Numbering*. Users in the future may want the flexibility of having the same telephone number for their entire life, irrespective of whether they change service providers, or shift their residence. In IS-41 and GSM, a user's number determines his home location (HLR), so life-long numbering is not possible without decoupling HLRs from geographic locations or service providers.

The primary contributions in this paper are:

1. *HiPER* (*Hierarchical ProfilE Replication*), a new family of *location management techniques* (LMTs) that efficiently provide life-long numbering. We use a hierarchical database architecture as in [AP91, KVP94, Wan93] to achieve life-long numbering. We then exploit *locality* in user calling and mobility patterns [JL95, SJW95] to selectively replicate user profiles at various databases in the hierarchy to reduce lookup latency and communication costs, at the expense of increased storage and update costs.

2. *Pleiades*, an event driven simulator that can simulate multi-day calling and mobility activity of several million users over large geographical areas. Our simulator has a flexible *simulation*

*Research supported by the Center for Telecommunications and the Center for Integrated Systems at Stanford University, and by equipment grants from Digital and IBM Corporations.

scripting language (SSL) and an event generator with time-varying user calling and mobility events. A unique aspect of the calling models used in our simulations is that they were derived from over six months of calling activity from the Stanford University Campus; similarly our mobility models were derived from vehicular traffic activity data collected over a period of 8 months in the San Francisco Bay Area, and corroborated with published traffic studies from Europe and the United States [HY93].

3. Simulations that compare our proposed family of LMTs against traditional LMTs, such as HLR/VLR, and centralized database architectures. In addition to using the realistic call and movement models described above, for the simulations we used a realistic geographical model of the San Francisco Bay Area along with its freeways and bridges to simulate accurately the performance of the LMTs in a close approximation of a real geographical topology. We report performance numbers of average and peak system requirements in terms of storage, number of database lookups and updates, and network bandwidth and number of network message hops. We also report the percentage of location lookups serviced by local databases to compare the call setup latency of the different LMTs.

The paper is structured as follows. We cover related work next, in Section 1.1. In Section 2, we describe hierarchical database organization, present the intuition behind profile replication, and describe our HiPER LMT family. In Section 3, we describe the architecture of Pleiades, our extensible simulator, and its scripting language. We also present our user calling and mobility models and the telephone call and vehicle traffic data we used to derive them. In Section 4, we compare performance numbers of our proposed LMTs and other conventional LMTs on a realistic model of the San Francisco Bay Area. We conclude and discuss future work in Section 5.

1.1 Related Work

Profile replication or caching has been considered before to improve the lookup performance of HLR/VLR. Replication is different from caching in that replication always keeps all copies up-to-date, and there is no invalidation problem. Wolfson and Jajodia have proposed an on-line algorithm for dynamic data replication in distributed databases using a "no-knowledge" approach [WJ92]. While this algorithm converges to the optimal replication plan when traffic traces are regular, unlike our schemes theirs does not exploit the relative stability of calling and mobility patterns of users for fast convergence. In [SW95, SJW95], we showed how to augment the HLR/VLR scheme with selective off-line replication using network-flow algorithms. Jain et al. [HJM94, JLLM94] propose *per-user caching* where zones cache the last known location of certain users for faster lookup. The latter two schemes are based on the HLR/VLR scheme, while in this paper we consider how to perform selective replication on database hierarchies so we can offer life-long numbering efficiently.

Several previous studies [IB92, KVP94, PMG95] have published simulation results for a variety of conventional and proposed LMTs. Our simulations differ in that:

1. We simulate multi-day user calling and mobility activity of several millions of users over large geographical areas, a simulation scale we have not seen in previous work. Also, most previous work is based on simplistic models that incompletely characterize human calling and mobility patterns. In [LJCW96], we empirically showed that using realistic calling and mobility models is important, since peak system performance measures differ by more than 30% from those based on inaccurate models.

2. Our simulations report time-varying database and network requirements of several LMTs on a realistic geographical area. Also, our results include database lookups and message costs for calls that are not completed when the caller is busy, along with costs for retried calls; previous work has usually ignored these costs.

2 The HiPER Family of LMTs

We first describe a basic hierarchical database framework and its profile lookup and update algorithms. We then introduce HiPER in terms of its database lookup and update algorithms. We then describe the parametrization of HiPER, its replication placement algorithm, and a few additional issues.

2.1 Hierarchical Profile Management

In a basic hierarchical model [KVP94, Wan93], each leaf-level database services a zone and stores profiles of users located in that zone. Each database in the higher levels of the hierarchy stores "pointers" (user ID + database ID) to the next lower level database that stores the user's profile or has a pointer to a lower level database. There is a conceptual "root" database that stores a pointer for every user. In practice, the root may be distributed into several databases so that no one database needs to store all user pointers or service all root level queries and updates. This simple hierarchical LMT provides lifelong numbering since there is no concept of a "home site" for a user as in HLR/VLR.

When user A calls user B, the location lookup for B's profile first propagates up the hierarchy from A's zone to the first database that contains a pointer to B's profile. The query then propagates down the hierarchy following pointers to B's profile until it is found in the leaf-level database that services the zone in which B is currently located.

When user A moves from zone Z_i to zone Z_j, Z_i ships a copy of A's profile to Z_j, and deletes its own copy. The databases along the path to the *least common ancestor* of Z_i and Z_j are then updated so that the pointers to A's profile reflect the new location of A. In practice, we find that the number of databases to update is low due to the high degree of locality in user mobility (see Sections 3 and 4).

2.2 Basic Replication

With the lookup strategy in the hierarchical approach, there may be several database lookups and network hops, hence the cumulative lookup time may be too high. Hence we propose selectively replicating user profiles at additional databases in the hierarchy to avoid the sequence of many lookups, some of which may be at remote databases. The idea behind profile replication is to reduce the latency of profile lookup at the expense of increased update and storage cost. In contrast with replication schemes in databases [OV91], our notion of replication simply requires updates to be propagated to each profile replica, rather than performing expensive distributed locking [OV91]. In [SW95] we showed how to use network flow algorithms to augment HLR/VLR with profile replicas based on estimated user *local call to mobility ratio* or *LCMR* information [JL95], and capacity limitations on databases. Essentially, a user's profile is replicated at a set of databases such that the benefit of local lookups at these databases outweighs the update cost due to the user's mobility. In the next subsection we extend this notion of "smart," off-line replication to hierarchies so that profile replicas, may be assigned to internal nodes of the hierarchy in addition to leaves.

2.3 Parametrizing HiPER

We parametrize our LMT family as HiPER(N, R_{min}, R_{max}, L), where N defines the maximum number of replicas of each user profile, R_{min} and R_{max} (for *replication selectivity*) together determine when a site may be a candidate for a profile replica, and L determines the maximum height in the hierarchy at which profile replicas can be placed. In the following subsections we consider each of these parameters in greater detail.

2.3.1 Number of Replicas (N)

In practice, we expect the number of profile replicas to be bounded. This restriction may be to satisfy storage constraints or to bound the number of global updates due to a user move. In this section, we consider one simple way to compute an appropriate N based on global storage considerations; a similar model can be developed for update costs.

Let p be the size of a pointer (user ID + database ID) in the hierarchy. Let P be the size of a user profile, and let H be the height of the hierarchy. In the basic hierarchal organization, the storage required for a single user is $P + p(H-1)$. In *HiPER*, with a maximum of $N-1$ additional replicas per user, the storage required for a single user is no greater than $NP + p(H-1)$. Let S be the maximum cumulative storage that can be allocated in the databases for each user. Then we can compute N to be

$$N = S + \frac{(S-1)p(H-1)}{P} \tag{1}$$

2.3.2 Replication Selectivities (R_{min}, R_{max})

A user profile i should not be replicated at site j if the cost of replicating the profile exceeds the benefit of replication. While there are many ways of making such decisions based on different cost models, one common cost model is to minimize network communication costs. For this, we compute the local call-to-mobility ratio [JL95], $LCMR_{i,j}$, where $LCMR_{i,j} = C_{i,j}/M_i$ if $C_{i,j}$ is the number of calls to user i from zone j and M_i is the number of moves of user i in a given time period. We can then decide to replicate user i's profile at database j only if the $LCMR_{i,j}$ exceeds some minimum threshold, R_{min}. For instance, [HJM94] proposes $R_{min} = 5$ for offline-caching, and [SW95, SJW95] propose $R_{min} = 0.25$ for offline-replication in HLR/VLR. For a hierarchical database, it is impossible to derive one such parameter R_{min} for databases at different levels in the hierarchy, since the change in communication costs due to replication of a user profile now depends on the actual database topology. However, with an additional parameter R_{max} we can achieve our goal of choosing potential sites for replication as follows:

1. If $LCMR_{i,j} < R_{min}$, never choose to replicate i's profile at database j.

2. If $LCMR_{i,j} \geq R_{max}$, always choose to replicate i's profile at database j, if constraints on N (Section 2.3.1) and L (Section 2.3.3) are satisfied.

3. If $R_{min} \leq LCMR_{i,j} < R_{max}$, whether i's profile should be replicated at database j depends on the actual database topology. In Section 2.4, we propose an algorithm that decides when such databases should be chosen as sites of replicas of i's profile.

We now show how to compute R_{min}^{opt} and R_{max}^{opt}, the optimal values of R_{min} and R_{max} for choosing potential sites of replication in a hierarchical database organization, if communication costs are to be minimized.

Let j be a database in the hierarchy, and let l_j refer to j's level in the hierarchy, where leaf-level databases have level zero. Let $parent(j)$ refer to the parent database of j, and $children(j)$ refer to the set of children databases of j. If j is a leaf-level database, $LCMR_{i,j} = C_{i,j}/M_i$ as above. If j is a non-leaf database, $LCMR_{i,j} = \sum_{k \in children(j)} C_{i,k}$. Let $\delta_{j,k}$ be the network cost of a lookup message that traverses the hierarchy from leaf database j to database k while performing lookups at intermediate nodes; let b_l be the network cost of the lookup message if j and k are adjacent in the hierarchy. Let b_u be the network cost of each update message. Consider two cases:

1. i's profile is replicated at database j.

   The communication cost incurred due to the replica at j is $b_u M_i$ since there are M_i updates sent to j for i's moves, and there is no lookup cost due to the local replica.

2. i's profile is not replicated at database j.

   The communication cost incurred is only due to lookups. The minimum lookup cost is when there is a replica of i's profile at the next higher level in the hierarchy, and this cost is $b_l C_{i,j}$. The worst possible communication cost is when i's profile can only be obtained by traversing the entire hierarchy from j to the database having i's profile (through the least common ancestor of the two databases). This cost is $C_{i,j}(\delta(j, LCA_{j,k}) + \delta(LCA_{j,k}, k))$, where $LCA_{j,k}$ is the least common ancestor between j and k, and database k services a zone in which i is located.

The above costs simplify to $[b_l C_{i,j}, b_l C_{i,j}(2E[LCA] - l_j)]$, where $E[LCA]$ is the expected LCA over the set of zones i visits over some time period; $E[LCA]$ indicates the expected number of hops before a replica is found.

Replicating i's profile at j always incurs less communication cost than when not replicating if

$$b_l C_{i,j} \geq b_u M_i \tag{2}$$
$$\Updownarrow$$
$$\frac{C_{i,j}}{M_i} \geq \frac{b_u}{b_l} \tag{3}$$
$$\Updownarrow$$
$$LCMR_{i,j} \geq \frac{b_u}{b_l} \tag{4}$$

Thus, we choose $R_{max}^{opt} = b_u/b_l$.

Similarly, replicating i's profile at j always incurs more communication cost than when not replicating if

$$b_l C_{i,j}(2E[LCA] - l_j) \leq b_u M_i \tag{5}$$
$$\Updownarrow$$
$$\frac{C_{i,j}}{M_i} \leq \frac{b_u}{b_l(2E[LCA] - l_j)} \tag{6}$$
$$\Updownarrow$$
$$LCMR_{i,j} \leq \frac{b_u}{b_l(2E[LCA] - l_j)} \tag{7}$$

Thus, we choose $R_{min}^{opt} = b_u/(b_l(2E[LCA] - l_j))$.

In the range $LCMR_{i,j} \in (R_{min}^{opt}, R_{max}^{opt})$, the relative performance of replication versus non-replication depends on the actual assignment of profiles to the topology. We shall consider how to choose sites of replication in Section 2.4.

2.3.3 Maximum Level of Replication (L)

In the last subsection, we saw how to compute possible sites of replication for a user profile based on communication costs. However, notice that if a database j is chosen as a site of replication for a user i due to a large $LCMR_{i,j}$ value, all ancestors of j will also be chosen as sites of replication, since $LCMR_{i,j}$ at a non-leaf database j is $\sum_{k \in Child(j)} LCMR_{i,k}$. Hence while replicating user i's profile at higher levels reduces communication cost, if we follow this approach naively, databases at higher levels may suffer too many updates due to moves of users whose profiles are replicated at those databases. Due to this "tension" between reducing network communication costs and reducing update activity at higher-level databases, we set a cap L on the maximum level at which profiles may be replicated.

2.4 Computing Sites of Replication

An algorithm in the HiPER family of LMTs executes the following off-line replica allocation algorithm over a model of the database topology and estimated LCMR information, and then propagates the replication plan to the actual databases.

The replica allocation algorithm proceeds in two phases for each user profile i. Conceptually, in the first phase, it allocates a profile replica to all databases j with $LCMR_{i,j} \geq R_{max}$. This allocation is performed as long as the number of allocated profile replicas $n \leq N$. The allocation in Phase 1 is done in a bottom-up traversal from level 0 to level L so that replicas are as close to the leaf level zones as possible, thereby reducing the lookup latency at sites with most calling activity. As soon as a database j is assigned a replica for user i, j's parent's LCMR value for i is reduced by $LCMR_{i,j}$ since lookups from below j's database for i will be serviced at j, and hence should not be used in making a replication decision for j's parent.

If, at the end of Phase 1, $n < N$, in Phase 2 the databases below level L and with the largest non-negative $LCMR_{i,j} - R_{min}$ are chosen as sites of replication. The allocation in Phase 2 proceeds in a top-down fashion (from level L) since allocating replicas higher up in the hierarchy maximizes the "coverage" area of the replica. Note that we can optimize the second phase based on the observation that if the $LCMR_{i,j}$ of some database j at level k is lower than R_{min}, we can "prune away" databases that are descendants of j since they will have lower LCMR values.

Let $\mathcal{L}_k$ denote the set of databases at level k. After executing the following algorithm, $\mathcal{R}_i$ will contain the set of databases that should contain replicas of user i's profile. $\mathcal{T}_i$ is a temporary set used in the second phase to compute the set of databases that have LCMR's in the range (R_{min}, R_{max}) for user i.

- **Phase I**

  $n = 1$, $\mathcal{R}_i \leftarrow \emptyset$
  Compute $LCMR_{i,j}$ for each database j at level 0
  For each level l_k of the hierarchy from 0 up to L
      For each $j \in \mathcal{L}_k$
          If $LCMR_{i,j} \geq R_{max}$
            $\mathcal{R}_i \leftarrow \mathcal{R}_i \bigcup \{j\}$
            $LCMR_{i,parent(j)} =$
              $LCMR_{i,parent(j)} - LCMR_{i,j}$
            $n = n + 1$
            If $n = N$ Exit

- **Phase II**

  $\mathcal{T}_i \leftarrow \mathcal{L}_L$
  For each $j \in \mathcal{T}_i$
      If $LCMR_{i,j} \leq R_{min}$
        $\mathcal{T}_i \leftarrow \mathcal{T}_i - \{j\}$
      Else $\mathcal{T}_i \leftarrow \mathcal{T}_i \bigcup children(j)$
  $\mathcal{T}_i = \mathcal{T}_i - \mathcal{R}_i$
  Select into $\mathcal{R}_i$ $N - n$ databases from $\mathcal{T}_i$ with largest $LCMR$ value

2.5 Discussion of HiPER

As mentioned earlier, HiPER's parametric nature allows us to make tradeoffs between communication and update costs versus space requirements and lookup latency. Since the interaction between the various parameters is complex, it is hard to gauge the impact of each parameter on the various system costs. However, in Table 1 we indicate general trends of each of the various costs, when the value of one parameter is increased (indicated by $\uparrow$) while the other parameters are fixed. Notice that it is impossible to independently characterize the impact of each of the parameters on communication costs since the impact depends on the actual values of R_{min} and R_{max} (recall Section 2.3.2). If, for instance, R_{min} and R_{max} are R_{min}^{opt} and R_{max}^{opt}, increasing N or L will reduce communication costs. If R_{min} or R_{max} are poorly chosen, communication costs may increase when N or L are increased. However, notice that even in the latter cases, the lookup latency will always be reduced.

Some parameter settings for HiPER allow us to model a variety of techniques, including the Simple Hierarchy of [KVP94, Wan93] and different degrees of replication. Let n be the number of databases and k the number of levels in the network hierarchy. Then:

- HiPER(1, 0, 0, 0) yields the unreplicated hierarchical scheme of [Wan93, KVP94].
- HiPER(n, 0, 0, 1) yields a scheme with full replication at leaf databases.

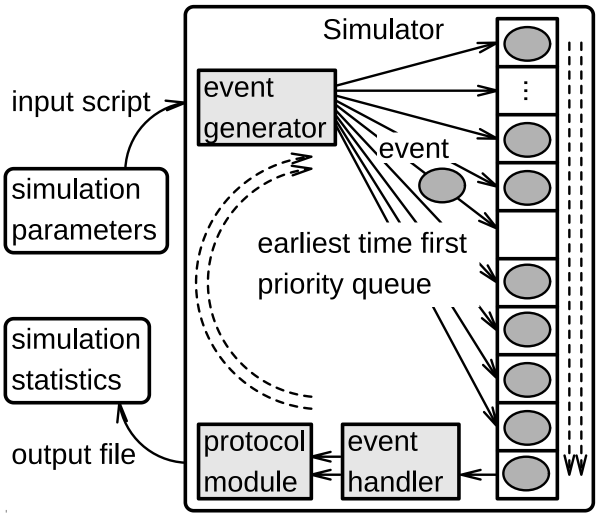

Figure 1: Pleiades Architecture

- HiPER(n, 0, 0, k) yields a scheme with full replication at all databases.

Notice that the replication selectivities of the above schemes do not depend on R_{min}^{opt} or R_{max}^{opt} as derived in Section 2.3.2. This indicates why the above schemes are sub-optimal with respect to communication costs when compared to our proposed selective replication with $R_{min} = R_{min}^{opt}$ and $R_{max} = R_{max}^{opt}$.

3 Pleiades Simulator

In this section, we first present the architecture of *Pleiades*, the extensible event-driven simulator we have built to experiment with and compare various LMTs. We then explain how we use indexing structures to make our simulations scale to millions of users. We then describe our flexible *Simulation Specification Language* (SSL) and show how we can specify complex database architectures, arbitrary network topologies, and arbitrary geographical registration areas. We also describe our user calling and mobility models, and how we instantiated our models to obtain realistic calling and mobility patterns for our simulations.

We developed Pleiades in C++ in about 10,000 lines of code. Simulation scripts specifying the calling patterns, mobility patterns, network topologies, and geographical entities typically range between 50 and 350 lines. We have implemented several LMTs, including HLR/VLR and HiPER, in approximately 150 lines of code each. Since Pleiades is object-based, we expect adding new LMTs will require approximately the same amount of coding.

3.1 Architecture

Pleiades consists of an event generator, an event handler, an earliest-time-first priority queue, and a protocol module, as shown in Figure 1. Initially, the simulator constructs the geographical and network topologies according to the input simulation script (described below in Section 3.2). The simulator then populates the zones with users, each of whom has a specified movement and calling pattern, from a set also defined in the simulation script.

The simulator cycles through a sequence of time windows in which call and move events are generated for the users according to the calling and movement patterns assigned to them in the simulation script. Each event is enqueued in proper temporal sequence. Once a full window of events is enqueued, the event processor passes them in sequence to the profile management module, which generates the simulation statistics. The statistics describe the movement and calling activities of the users along with the database and network loads they incur.

We implemented the time prioritized queue as a specialized B^+-tree with insert times logarithmic in the number of distinct time values stored, rather than in the number of events stored. The operations specified on the B^+-tree are `insert_with_priority_x`, and `extract_top_priority`. Events generated with the same priority (time when they need to be dequeued) are stored using `insert_with_priority_x` in a linked list under the B^+-tree. Our `extract_top_priority` operation achieves unit-time access (rather than logarithmic time as in traditional heap structures) to the highest priority entry in our B^+-tree queue thereby making it suitable for fast event processing. Our B^+-tree services over 25,000 insertions and deletions per second on a 20 MIPS SPARCstation, allowing us to simulate very large user populations (currently over 3 million users). These large-scale simulations help us compare performance characteristics of the different protocols in a more realistic fashion than in previous simulation studies.

3.2 SSL: Simulation Specification Language

An SSL script instantiates the following two key components of our simulations: (1) the network and geographical topologies, and (2) user behavior. For a

Parameter	Change	Impact			
		Storage	Update Costs	Comm. Costs	Lookup Latency
N	↑	↑	↑	↑↓	↓
R_{min}	↑	↓	↓	↑↓	↑
R_{max}	↑	↓	↓	↑↓	↑
L	↑	↑	↑	↑↓	↓

Table 1: Impact of Varying Parameters on Performance Measures.

detailed description of the SSL grammar and a sample simulation script see Appendix A.

The first component in an SSL script specifies a topology of sites along with their population characteristics and geographical connectivity (e.g., freeways between counties), as well as the inter-connecting network topology. It also specifies the database topology of the LMT to be used. The important commands are:

- The *Frame* declaration reserves memory for all objects defined in the simulation, and determines time granularity parameters.
- The *Site* specifier declares the population density and distribution, as well as defining its physical adjacency to other sites.
- The *Link* function establishes a network link between two sites with a cost that may vary during the course of a simulation due to traffic conditions.

The second component describes the set of users, as well as individual movement and calling patterns. The important commands are:

- The *User* command defines classes of users and the probability with which users in different classes will be found at the various defined sites. Each user class also has a specific movement and calling pattern (described below in Sections 3.3 and 3.4).
- The *Call* and *Move* definitions are similar. Each establishes a frequency distribution and the time window during which the distribution is valid. For example, we may define a business calling pattern, with a valid time frame from 9 A.M. to 5 P.M. and a bursty distribution averaging 3 calls per phone per hour. Likewise, we might define a commuter movement pattern, with a valid time frame from 6 A.M. to 10 A.M. and an average commute time of 30 minutes.

3.3 User Movement Model

Our user mobility model is general enough to handle a large class of user movement patterns. The key aspects of our model are:

1. We represent several classes of movements commonly observed in humans. For example, users can make random walks, repetitive roundtrip movements and return back to home. Each of these movement patterns can be coupled to varying movement velocities and probabilities of occurrence.
2. We can simulate temporal changes in the above user movement patterns as a function of our simulation time, for example, to simulate different movement models during the course of the day.

It is important to note that different LMTs optimize for different movement behaviors. For instance, the HLR/VLR scheme assumes the user returns home often, while replication assumes repetitive movements. Hence, to be fair while comparing the different techniques, we require a realistic instantiation of our movement model. Our modeling of user movement is based on a transportation survey, the *NPTS* [HY93, Kit95], conducted in 1990. This data is similar to vehicle traffic statistics obtained from roadside measurements in Europe [ECM95]. In addition, we have obtained actual movement statistics [MTC90] from around the San Francisco Bay area collected over an eight month period in 1989-1990 to correlate with survey data. Using this data we instantiated our movement model for a typical San Francisco Bay area commuter and non-commuter, basing our movement distances on the values quoted in the surveys. For further details see [LJCW96].

3.4 User Calling Model

The key aspects of our calling model are:

1. We represent a variety of call types, such as *simple* calls, and *retry* calls when calls are reattempted if busy. The calling types can be coupled to varying probabilities of occurrence, along with how often users place calls to other users and how long each call lasts.

2. As in the movement model, we can simulate calling patterns that change over the course of a day.

3. We simulate calling locality in callee distributions by maintaining, for each caller, a list of the people they call most often with the probabilities of making calls to each of them. When a call is generated, the callee is selected either randomly from the set of all users, or from the user's callee list according to its probability distribution.

In [LJCW96], we showed that the above components can be treated independently, since we observed very low correlation between calling rates and callee probabilities.

Again, to be fair in comparing various LMTs, we instantiated our call models with accurate parameters we derived from call traffic data [Tel95] for a six-month period from our university telephone exchange. This exchange serves the entire campus, including university offices, student housing, and residential households. The data was preprocessed at the source to protect the anonymity of the callers and callees. The data set covers 19,605 distinct callers and contains encrypted caller and callee identifications, time of call, and call duration for each outbound call. While this data constitutes a specialized sample of call traffic, the composition of our campus allows us to infer call traffic patterns for both business and residential settings. We used each caller's busy and idle periods to determine average phone usage, the distribution around the average, and the burstiness of the calling behavior. We calculated empirical callee distributions by ranking callees according to the frequency with which they are called during reference time periods of one day, one week, and one month (four weeks). For more details refer to [LJCW96].

4 Simulation Results

In this section we first describe the geography and network topologies we used in our simulations. We then present simulation results for several LMTs.

4.1 San Francisco Bay Area Simulation

We performed our simulations on a geography that accurately models the San Francisco Bay Area. A map of the entire Bay Area is provided in Figure 2 for the reader's reference. The Bay Area consists of nine counties with a 1990 population of 6,023,877 [UpC94], and is serviced by four area codes. Regions corresponding to different area codes are represented by different shades in Figure 2; bridges, freeways, ferries, and public transportation systems are also included. Figure 3 is an overlay map depicting the relationship between our simulation model and the physical geography [USG95]. Registration areas are represented as polygons in this figure. We simulated a 3-level hierarchy of databases. Dots in the middle of polygons (registration areas) represent the databases servicing that area. Medium sized dots are intermediate databases servicing a set of lower level databases. The four large dots indicate a distributed root. Network links are represented by lines connecting the database dots.

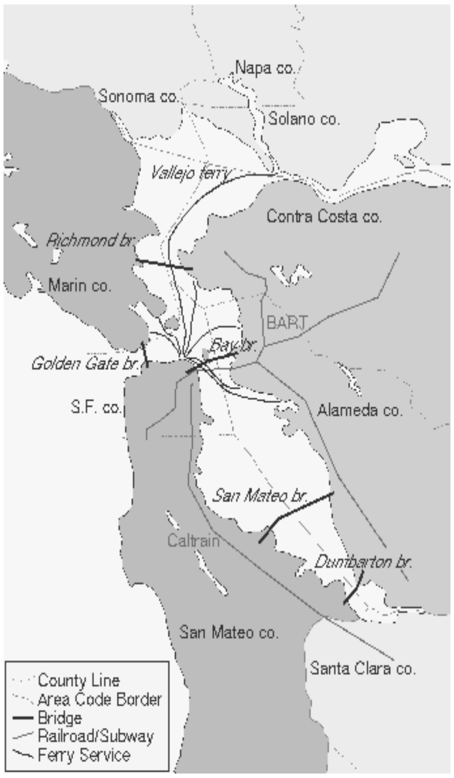

Figure 2: San Francisco Bay Area

In our simulation, we populated registration areas with users based on 1990 census information

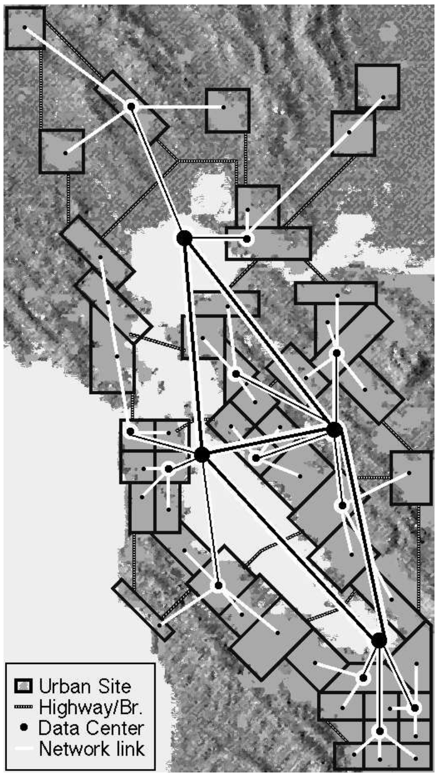

Figure 3: Simulation and Network Topologies

from [UpC94]. We assumed that 50% of the population would subscribe to wireless services. We divided our user population into 41% commuters and 59% non-commuters. This proportion is derived from the national average in [HY93] and the peak-to-total traffic figure for the Bay Area. We specified connectivities between transportation routes such as highways and bridges. Using traffic volume statistics from [MTC90], we estimated movement between area codes and fine-tuned our simulation parameters to produce similar large scale movement behavior. See [LJCW96] for more details.

We performed simulations corresponding to a 5 day period, and report results corresponding to the second 24-hour period (which we found were comparable to the results from the 3rd through 5th days) to avoid simulation transients. Each simulation generated some 22.5 million events, and consumed approximately 1 hour of processor cycles on an INTEL Pentium 120 MHz processor with 32 MB RAM running Linux.

For our simulations, we studied the following LMTs:

1. *HLR/VLR*,
2. *Centralized* database architecture in which all user profiles are stored in one centralized database,
3. *Full Replication* in which all user profiles are replicated in all databases,
4. *Simple Hierarchy* of [KVP94, Wan93] as described in Section 2.1, and
5. $HiPER_{5,3}$, one instance of HiPER that represents an intermediate point between Full Replication and Simple Hierarchy, with $N = 5$, $L = 3$, $R_{min} = R^{opt}_{min}$, $R_{max} = R^{opt}_{max}$, and $b_l/b_u = 1$. We chose $N = 5$ since five replicas are sufficient to satisfy most lookups (over 90%) locally [SJW95] due to calling locality, and $L = 3$ since that is the maximum number of levels in our hierarchy. We set $b_l/b_u = 1$, assuming that the cost of an update message and a lookup message are equal.

Note that HLR/VLR does not support life-long numbering, while the latter four schemes do.

4.2 Performance of Profile Management Techniques

We first consider the system-wide database query, database update, and network messaging loads for the different profile management techniques. We then examine the percentage of calls that are serviced by (fast) local lookups for each of the techniques. Finally, we consider the storage requirements of HLR/VLR, the Simple Hierarchy, and $\text{HiPER}_{5,3}$.

Table 2 reports the peak levels of database operations and network signaling for each LMT. Note that the peaks for database lookups and updates do not occur at the same time. Hence, the peak for total database load is not the sum of the peaks for lookups and updates. In general, Table 2 confirms HiPER's competitiveness with other LMTs. Although the centralized scheme performs best, it does not scale since all the operations take place at a single database. As expected, the peak number of global database lookups in $\text{HiPER}_{5,3}$ is less than the Simple Hierarchy. It is interesting to note that the peak number of database lookups in $\text{HiPER}_{5,3}$ is close to that of HLR/VLR indicating that most lookups are serviced within one or two hops from the caller's zone. Hence $\text{HiPER}_{5,3}$ brings the benefit of life-long numbering at a cost competitive with HLR/VLR.

In Figures 4 and 5, we see the number of global lookups and updates (on a log scale) for

ops/s	Central DB	Full Repl.	Simple Hierarchy	HiPER$_{5,3}$	HLR/VLR
DB Lookups	120	120	212	132	129
DB Updates	9.53	858	34	49	24.6
DB Totals	114	954	231	166	145
Network Msgs	104	2930	155	134	115
Network Hops	302	3720	180	306	326

Table 2: Summary of Peak Protocol Requirements

the above LMTs for the 24-hour period we measured. Again we see that the number of lookups in Simple Hierarchy far exceeds those of HiPER$_{5,3}$ and HLR/VLR. Also notice that HiPER$_{5,3}$ and HLR/VLR are close to the minimum of the number of lookups achieved in the Centralized and Full Replication schemes. As expected, the number of global updates in HiPER$_{5,3}$ is higher than in HLR/VLR and Simple Hierarchy but much lower than in Full Replication. Of course in absolute terms these update are all easily supportable in modern database systems [GR93].

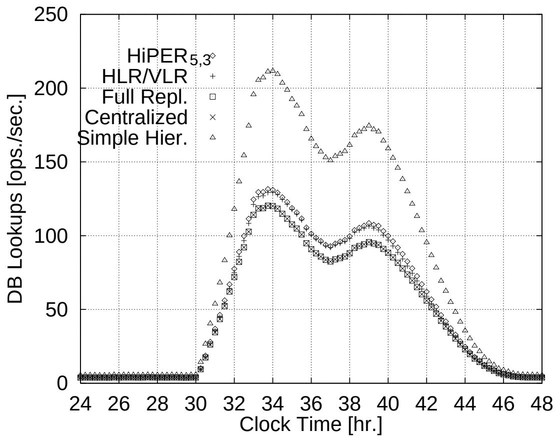

Figure 4: Profile Lookup Rate

In Figures 6 and 7, we see the network bandwidth requirements and the number of message hops of the various LMTs over the studied 24-hour period. An interesting observation is that HiPER$_{5,3}$ incurs a low number of network hops despite the extra messages due to updates; it performs better than Simple Hierarchy and close to HLR/VLR. We also see that the number of message hops in Simple Hierarchy is lower than HiPER$_{5,3}$ due to a lower number of updates, and lower than HLR/VLR since updates are not remote.

In Figure 8, we plot the percentage of calls that

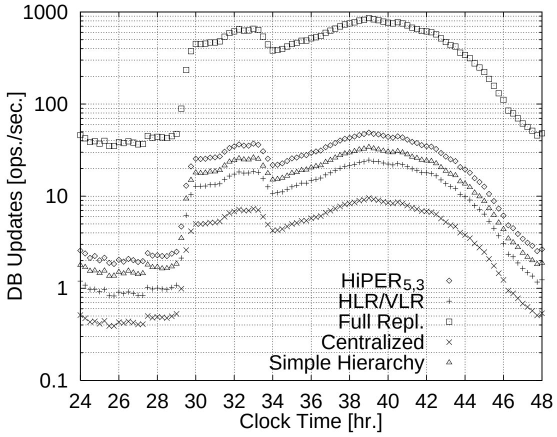

Figure 5: Profile Update Rate

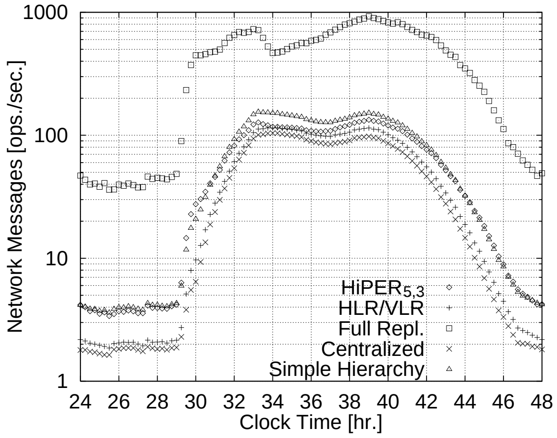

Figure 6: Network Signaling Messages

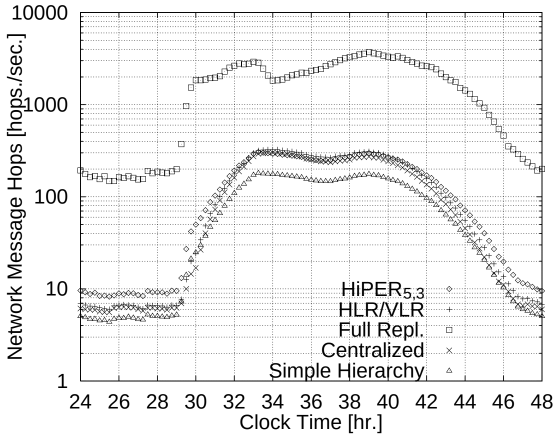

Figure 7: Message Hop Count

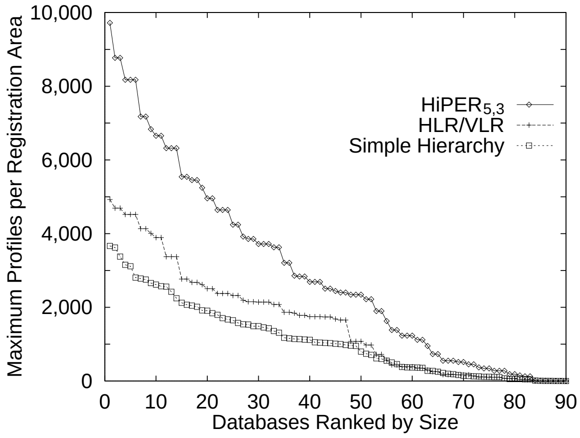

Figure 9: Database Memory Requirements

are serviced by local database lookups for the various schemes, along with the percentage of users currently located at home. Notice that HiPER$_{5,3}$ services more than 90% of the calls by local lookups, thus allowing for fast call setup. This is in comparison to HLR/VLR and Simple Hierarchy in which over 5 times and over 7 times as many lookups are serviced by remote databases, respectively.

In Figure 9, we show the storage requirements of HiPER, HLR/VLR and Simple Hierarchy by graphing the cumulative percentage of databases that store a given number of profiles. We see that some databases in the HiPER$_{5,3}$ scheme may need to store up to twice as many profiles as HLR/VLR. However, we believe that with decreasing disk costs this will not be a significant problem in the future.

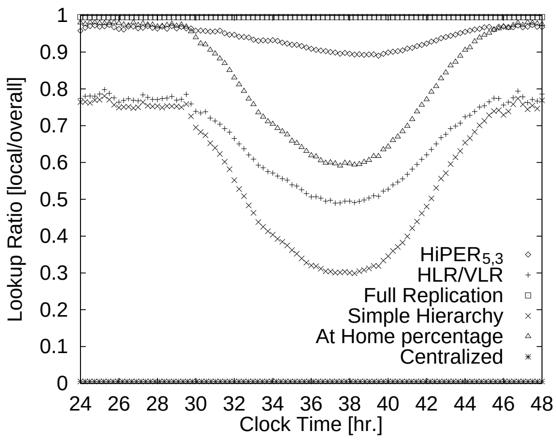

Figure 8: Local Lookup Percentage

Table 3 summarizes some of our results comparing HiPER$_{5,3}$ to HLR/VLR and Simple Hierarchy. The numbers in the table are relative to Simple Hierarchy and are systemwide totals. For example, HiPER$_{5,3}$ requires only 0.16 of the remote lookups that Simple Hierarchy requires during peak activity. Table 2 confirms again that HiPER is competitive with HLR/VLR in terms of key system performance requirements while achieving the benefits of life-long numbering.

5 Conclusions and Future Work

In this paper we made contributions in three areas: (1) We introduced a family of LMTs, HiPER, that can efficiently support life-long numbering. (2) We described our flexible wireless network simulator, Pleiades, that realistically models human behaviors. (3) We presented real time simulation results for our new LMTs and compared it to other basic and current techniques.

From our simulations we see that for the instance of HiPER we considered, more than 90% of calls can be serviced by local database lookups. The penalty paid for the latency improvement is a slight increase in bandwidth requirement, and near doubling of memory requirements and update rates. We believe that as storage prices continue to drop, the flexibility of life-long numbering with fast lookups will become very important.

In the future, we plan to simulate more instances of HiPER to study interesting operational points for N, L, R_{min} and R_{max}. We are also extending our simulator to handle networks that simultaneously support several LMTs. This will become particularly important if and when current strategies like HLR/VLR

Feature	Simple Hierarchy	$\text{HiPER}_{5,3}$	HLR/VLR
Life-Long Numbering	YES	YES	NO
Remote Lookups	1.0	0.16	0.73
Message bandwidth	1.0	0.86	0.74
Storage	1.0	2.65	1.41
Update cost	1.0	1.44	0.72

Table 3: Peak Performance and Features Summary

evolve over time to hierarchical techniques. Since we feel that realistic models give us useful insights in developing new protocol management techniques, we are constantly refining our modeling of human behaviors for calling and mobility patterns. We have released *SUMATRA*, the *Stanford University Mobile Activity TRAces*, that provide realistic calling and mobility activity so other wireless researchers may also benefit from our work in developing comprehensive traffic models. These traces are available at `http://www-db.stanford.edu/sumatra`.

References

[AP91] B. Awerbuch and D. Peleg. Concurrent online tracking of mobile users. *SIGCOM*, 21(4):221–233, September 1991. Zurich Switzerland.

[ECM95] ECMT. *European Transport Trends and Infrastructural Needs.* OECD publications, 2, Rue Andre-Pascal, 75775 Paris CEDEX 16, France, 1995.

[EIA91] EIA/TIA IS-41.3 (Revision B). *Cellular Radiotelecommunications Intersystem Operations*, July 1991.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kauffman, 1993.

[HJM94] 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*, March 1994. Long Branch, NJ.

[HY93] P. S. Hu and J. Young. *1990 NPTS Databook: Nationwide Personal Transportation Survey.* Federal Highway Administration, November 1993.

[IB92] T. Imielinski and B. R. Badrinath. Querying in highly mobile distributed environments. In *Proc. 18th International Conference on Very Large Databases '92*, pages 41–52, August 1992. Vancouver B.C.

[JL95] 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*. IEEE, 1995.

[JLLM94] 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.

[Kit95] R. Kitamura. *Time-of-Day Characteristics of Travel: An Analysis of 1990 NPTS Data*, chapter 4. Federal Highway Administration, February 1995.

[KVP94] P. Krishna, N. H. Vaidya, and D. K. Pradhan. Location management in distributed mobile environments. In *Third International Conference on Parallel and Distributed Information Systems*, pages 81–88, September 1994. Austin, TX.

[LJCW96] D. Lam, J. Jannink, D. C. Cox, and J. Widom. Modeling location management for Personal Communication Services. To appear in *IEEE International Conference on Universal Personal Communications ICUPC '96*, Cambridge, MA, September 1996. ftp://www-db.stanford.edu/pub/jannink/1996/icupc/.

[MTC90] 1990 commute summary. Metropolitan Transportation Commission, Lotus 123 format spreadsheet, August 1990. S.F. Bay area transportation measurements.

[OV91] M. T. Oszu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 1991.

[PMG95] G. P. Pollini, S. Meier-Hellstern, and D. J. Goodman. Signaling traffic volume generated by mobile and personal communications. *IEEE Communication Magazine*, pages 60–65, June 1995.

[SJW95] N. Shivakumar, J. Jannink, and J. Widom. Per-user profile replication in mobile environments: Algorithms, analysis, and simulation results. Technical report, Stanford University, Computer Science Dept., December 1995. ftp://www-db.stanford.edu/pub/shivakumar/1995/profile-journ.ps, Submitted for journal publication.

[SW95] 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, CA, October 1995.

[Tel95] Telephone call traffic data set, 3/95–9/95. S. Phillips, personal communication, October 1995. Encrypted ID numbers.

[UpC94] 1990 U.S. census demographic summaries. UpClose Publishing, http://www.upclose.com/upclose/, 1994. Population of nine S.F. Bay area counties.

[USG95] 1990 urban land use in central California. U.S. Geological Survey, http://geo.arc.nasa.gov/usgs/images/urban2.gif, 1995. Urbanization in S.F. Bay area.

[Wan93] 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.

[WJ92] O. Wolfson and S. Jajodia. Distributed algorithms for dynamic replication of data. In *Proceedings of the Symposium on Principles of Database Systems*, pages 149–163, San Diego, CA, 1992.

A Script Grammar and Sample Script

We begin with a BNF grammar of Pleiades' scripting language (SSL) down to the command granularity, then present a sample script that gives some insight into individual command structure. For reference, the simulation script we used to define the San Francisco Bay area simulations is 334 lines long.

A.1 Script Grammar

- script::=
  structure-block user-block
- structure-block::=
  frame-block site-block opt-link-block
- user-block::=
  move-block call-block user-block |
  call-block move-block user-block
- frame-block::=
  opt-trace-cmd frame-cmd opt-param-cmd
- site-block::=
  site-cmd site-block |
  site-cmd
- opt-link-block::=
  link-cmd opt-link-block |
  opt-link-cmd
- move-block::=
  move-cmd move-block |
  move-cmd
- call-block::=
  call-cmd call-block |
  call-cmd
- user-block::=
  user-cmd user-block |
  user-cmd

A.2 Sample Script

```
# Pleiades simulator script, 4 zone square area
Trace write

# Frame ( name, length, transient, window, sites,
                    users, moves, calls, seed )

Frame ( four˙square, 1998, 558, 6, 4, 781.25, 5, 3, 5191995 )

# Site ( name, population˙coeff., PARENT, borders,
                    neighbor, move˙coeff., ... )
# Neighbors are ordered by direction when applicable:
                    West, North, East, South

Site ( site0, 1.0, site0, 2, site1, 1.0, site2, 1.0 )
Site ( site1, 1.0, site0, 2, site0, 1.0, site3, 1.0 )
Site ( site2, 1.0, site0, 2, site0, 1.0, site3, 1.0 )
Site ( site3, 1.0, site0, 2, site1, 1.0, site2, 1.0 )

# Link ( name, from˙site, to˙site, cost )

Link ( l1˙2, site1, site2, 1 )
Link ( l1˙3, site1, site3, 1 )
Link ( l2˙3, site2, site3, 1 )

# Move ( name , time, range, prob., velocity, v˙dist,
            zones, z˙dist, roundtrip, pattern˙head )

Move ( stop1, 420, 60, 0.9875, 0.00001, 0, 0, 0, 1, Y )
Move ( walk1, 0, 0, 0.00217, 0.01, 0, 1, 0, 0, N )
Move ( ride1, 0, 0, 0.00217, 0.1, 0, 4, 0, 1, N )
Move ( home1, 0, 0, 0.00817, 0.1, 0, 2, 0, 2, N )

Move ( fast1, 120, 60, 1.0, 0.1, 0, 1, 0, 0, N )

Move ( stop2, 480, 0, 0.9875, 0.00001, 0, 0, 0, 1, N )
Move ( walk2, 0, 0, 0.00617, 0.01, 0, 1, 0, 0, N )
Move ( ride2, 0, 0, 0.00417, 0.1, 0, 4, 0, 1, N )
Move ( home2, 0, 0, 0.00217, 0.1, 0, 2, 0, 2, N )

Move ( fast2, 120, 60, 1.0, 0.1, 0, 1, 0, 2, N )

Move ( stop3, 0, 0, 0.9875, 0.00001, 0, 0, 0, 1, N )
Move ( walk3, 0, 0, 0.00217, 0.01, 0, 1, 0, 0, N )
Move ( ride3, 0, 0, 0.00217, 0.1, 0, 4, 0, 1, N )
Move ( home3, 0, 0, 0.00817, 0.1, 0, 2, 0, 2, N )

# Call ( name, time, range, prob., end˙prob., freq.,
           duration, d˙dist, retry, pattern˙head )

Call ( a1˙simple1, 420, 60, 0.6, 0.0, 0.06, 1, 2, 0, Y )
Call ( a1˙again1, 0, 0, 0.3, 0.0, 0.04, 2, 2, 1, N )
Call ( a1˙long1, 0, 0, 0.1, 0.0, 0.02, 8, 2, 0, N )

Call ( lot1, 720, 120, 1.0, 0.0, 0.1, 2, 2, 0, N )

Call ( a1˙simple2, 0, 0, 0.6, 0.0, 0.06, 1, 2, 0, N )
Call ( a1˙again2, 0, 0, 0.3, 0.0, 0.04, 2, 2, 1, N )
Call ( a1˙long2, 0, 0, 0.1, 0.0, 0.02, 8, 2, 0, N )

# User [ quantity, placement ] ( name, moveType,
           callType, calleeDirSize, localCallee% )

User [ 781.25, random ] ( basic, stop1, a1˙simple1, 8, 0.8 )
```