

Adaptive, Distributed Location Management in Mobile, Wireless Networks

Kevin Lee¹, Hong Wing Lee¹, Sanjay Jha^{1,2} and Nirupama Bulusu²

¹*School of Computer Science and Engineering, The University of New South Wales (UNSW) and*

²*National Information and Communication Technology Australia Limited (NICTA)*

{kevinl, honglee, sjha, nbulusu}@cse.unsw.edu.au

Abstract

Location management refers to the problem of updating and searching the current locations of mobile nodes in a wireless network. To make it efficient, the sum of update/lookup costs of a location database must be minimized. Previous work relying on fixed location databases is unable to fully exploit the knowledge of user mobility patterns in the system so as to achieve this minimization. We present a novel location management architecture that can adapt dynamically to user pattern changes. The key idea is to use a hierarchy of agents that act as mobile location databases that rapidly replicate to other base-stations in response to a change in user patterns. We provide an analytical cost model for location management and through simulations demonstrate our algorithm's ability to handle location updates and call finding.

1. Introduction

Fundamental to the efficient operation of future cellular networks [1], self-organizing mesh networks [2,3,4] and emerging pervasive sensor networks is an efficient location management scheme. Location management refers to updating and searching the “whereabouts” of mobile nodes in a network.

Various location management schemes have been designed to lower costs and improve the efficiency of search and update times [7,8,9,10]. However, most existing schemes rely on fixed sets of location databases that are inflexible and immobile, and cannot readily exploit the user mobility patterns inherent in virtually all wireless networks. In many systems, the databases are centralized, causing more reliability problems.

Recent years have seen extensive research on novel self-organizing wireless network architectures[2,3,4]. Terminodes[2] proposes a wide-area, autonomous, self-organized, wireless multimedia network, independent of any fixed infrastructure. The Multihop cellular architecture [4] provides localized ad-hoc

networking. Mobile hosts within a cell forward packets to the base-station to expand cell coverage while maintaining the base-station's transmission range. In such self-organizing networks, it is hard to maintain a fixed, centralized location database.

Location management schemes typically use two extremes, (i) up-to-date and exact location information at all sites, requiring huge bandwidth for each location update but minimizing lookup cost; or (ii) storing no location information at any site, requiring no update but with prohibitive lookup cost. In this paper, we explore the alternative - to find a point between the two extremes, where update/lookup costs are balanced[1]. Our design goals for location management are:

Tunable: Can be tuned to exploit user mobility patterns and uneven regional distribution for efficiency.

Flexible: Provide greater flexibility in terms of deployment of location information so as to achieve a balance of update and lookup costs.

Adaptive: Dynamically adapt to a desirable balance by automating tracking of user patterns, and exploit a change in user patterns to the extent of the fluctuation.

Our design depends on exploiting user mobility patterns to optimize location management costs. A user mobility pattern can be described by - *locality* and *regional call-to-mobility ratio*. *Locality of movement* refers to the pattern where users usually move in a fixed set of locations (e.g. home, workplace). *Locality of call distribution* characterizes distribution of callees and callers. Most calls are generated between regions (eg. user's home and office). *Regional Call-to-Mobility Ratio* measures call volume against frequency of location change. Low call-to-mobility users change location frequently and make relatively fewer calls.

Our goal is to optimize location management by balancing the update and lookup costs. There are two design aspects. The first is *placement of location databases*. The key idea here, as in caching, is to

exploit the locality of mobile hosts by placing the database in the caller/callee vicinity to minimize the amount of update or lookup network traffic. This may require duplicated databases, called *replicates*, that are placed in the vicinity of likely callers. The second aspect is the *number of replicates* used. To further balance search and location update cost, number of replicates is based on call-to-mobility ratio.

The paper’s key contributions are: We propose a novel location management architecture that relies on a hierarchy of location database agents to help provide a flexible and distributed system, wherein agents rapidly replicate to other base-stations in response to change in user patterns (Section II). We propose an analytical cost model to formulate the location management optimization problem, show it is NP-complete, and suggest an approximation algorithm (service ability) to solve the optimization (Section III). We demonstrate via simulation the algorithm’s ability to handle location updates and call finding (Section IV).

2. Location Management Algorithm

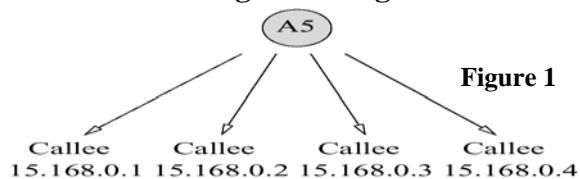


Figure 1

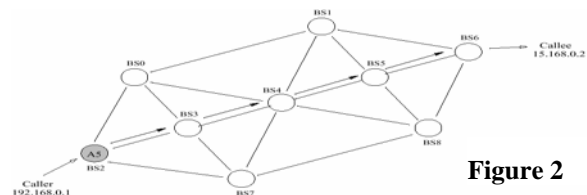


Figure 2

2.1 Network model

The backbone network consists of many base-stations. A cell is the area covered by a base-station’s signal. Each mobile host can directly communicate with a cell’s base-station, and through it, with other mobiles. When it moves to another cell, the mobile’s point of attachment (communicating base-station), and hence location are changed. The address of the base-station covering it is the mobile’s Care-of Address (COA). In our system, location is stored as COAs.

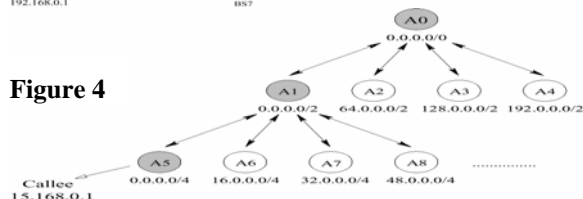
2.2 System Architecture

Location information is stored in location databases (*agents*), residing in base-stations, bound to the base-station they are available at. Each agent stores locations of several mobiles. An agent is a table with each entry storing the Care-of Address (COA) of another entity. When a host wants to locate another

mobile, it can read from the table and know which base-station, (which COA), it should forward the call to. E.g., in Fig.1: Agent A5 has four entries. Each entry stores the location of a callee: 15.168.0.1, 15.168.0.2, and so on. The notation above means that A5 has a pointer (COA) for each callee. When a caller wants to call 15.168.0.2 as in Fig.2, it must find out where it is. The base-station BS2 associated with the caller, looks up in A5, and finds out that the COA points to BS6. BS2 forwards the call to BS6, which finally reaches the callee. In this case, the agent A5 has



Figure 3



a binding to the base-station BS2.

A5 may not be bound to BS2, as in Fig.3. In Fig.3, the agent A1 does not know where the callee is, but it knows where the callee’s agent A5 is: it is bound to BS2. BS2 can then forward the call to BS8, where the location of the callee is known, and it is redirected towards the callee. Sometimes A1’s location is not known. We then require another agent A0 to locate it, so it is quite likely that the call must be forwarded thrice before it can reach the callee, as in Fig.4.

Agent A0 can store locations of many other agents, resulting in a tree-like structure between different agents: A5 is a child of A1, and A1 is the parent of A5. The number of children *b* an agent can have is fixed. This gives a tree structure with branching factor *b*. The leaf of the tree stores the locations of the mobiles. The upper level nodes store the locations of the location databases directly below them. Each agent in the tree has an individual unique netmask. Mobile hosts that match its netmask will be in the sub-tree of this particular agent. An agent can move freely within the network, by binding to the base-station it is migrating to. It is possible that more than one agent is bound to a single base-station. Our hierarchical tree is a logical structure as opposed to an actual networking structure.



Figure 5

2.3 Details

New Mobile Registration: The agent hierarchy originally starts with only a root agent. It stores locations of all mobiles, as in Fig.5. When new mobiles are added to the system, the agent expands to form an agent hierarchy. The new mobile number (eg. 192.168.0.2) is sent to the root, which forwards the registration to the child agent with netmask matching the mobile number. The number follows the tree path determined by the unique netmasks to the hierarchy's leaf agent. If the leaf agent is not full, it adds a location entry for this mobile. Otherwise, the leaf expands. During expansion, many new agents numbering the tree branching factor (Fig.6) are produced.

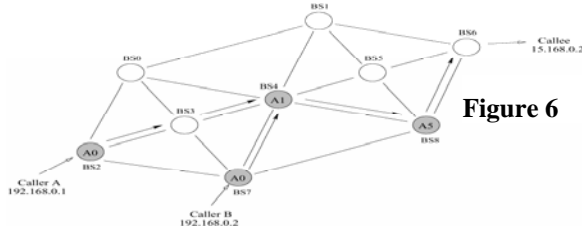


Figure 6

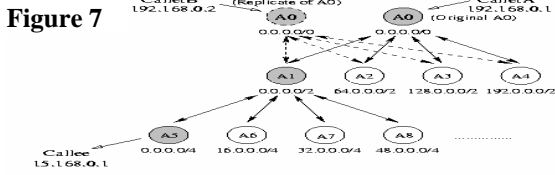


Figure 7

Agent Replication: From Fig.6, if callers are at similar physical locations (residing in nearby base-stations e.g. BS2, BS7), they often share close-to-common physical pathways to the same callee. Both callers access two copies of the original root agent because of agent replicates. Though their physical paths to the same callee are different, the logical path they took was the same (eg. Fig.7). The way the logical tree is “embedded” in the physical network allows two structures to relate to each other to seek an optimal solution. Replication can be applied to all agents in the hierarchy to shorten the network's physical paths. We can exploit this by grouping callers at similar physical locations, addresses so that they share exactly the same logical path, from root agent to the callee. This can significantly reduce number of agents along the path.

Updating Location Information: For efficiency, every mobile knows its parent agents locations, including replicates. When the mobile moves, it sends notification to its parent agent. Since the mobile knows where all replicates of its parent are, the updates are sent directly to them. The location update cost increases with the number of replicates.

3. Performance Analysis

3.1 Cost Model

The cost model for the optimization is an adaptation of [5], originally developed for Content Distribution Networks (CDNs). We consider a network of I base-stations. We denote each basestation by a number i , $i \in \{1, 2, \dots, I\}$. Each agent has J replicates. Request probability p_{ij} , $j \in \{1, 2, \dots, J\}$ is the probability that base-station i forwards a location request to agent j .

Request rate λ_i is the number of requests that will originate from base-station i to one of the replicates. Update rate μ_i is the number of updates that originate from base-station i to the replicates in a given time period. It is the same for every replicate, as all replicates must be updated when a mobile changes its point of attachment to the network. λ_i and μ_i are subject to the user patterns locality principle; λ_i is influenced by the locality of calls while μ_i is affected by the locality of movement. The result is an uneven distribution of λ_i and μ_i across the network.

In this cost model, cost is calculated as the number of hops that each request or update must travel. Assume that a base-station always picks the replicate closest to itself when requesting locations, i .

$$P_{ij} = \begin{cases} 1 & \text{if replicate } j \text{ is the closest replicate to base-station,} \\ 0 & \text{otherwise.} \end{cases}$$

The matrix is denoted by p . The distance between base-station i and the closest replicate of agent j is measured by the shortest path hop-count between base-station i and the host base-station of that particular replicate. The average number of hops that a lookup must traverse from base-station i is d_{ij} , where j is the closest replicate to base-station i and d_{ij} is the shortest distance to the host of replicate j from basestation i . The nearest replicate is either in basestation i or another basestation. Average number of hops that an

update must traverse from basestation i is $\sum_{j=1}^J d_{ij}$.

The cost, measured by the total hop count incurred for a particular agent from a particular base-station is

$$C_i = \sum_{j=1}^J p_{ij} \lambda_i d_{ij} + \sum_{j=1}^J \mu_i d_{ij} \quad (1)$$

Here, it is assumed that each base-station has infinite storage capacity. It can hold as many agents as it wants. Let $\Lambda_{ij}(p) = p_{ij} \lambda_i$

The total location management cost for all base-stations and all replicates is then

$$C(p) = \sum_{i=1}^I \left(\sum_{j=1}^J \Lambda_{ij}(p) d_{ij} + \sum_{j=1}^J \mu_i d_{ij} \right) \quad (2)$$

The ratio between λ_i and μ_i is influenced by the call-to-mobility ratio. Frequency of mobile movement determines the update rate μ_i while the frequency of calls determines the request rate λ_i . Update rate μ_{ij} is controlled by number of replicates inside the network. As more replicates are installed inside the network, the request rate to each individual agent decreases. This change in request rate is denoted by $\Delta p_{ij} = p_{ij1} - p_{ij0}$ and the matrix of all Δp_{ij} is denoted by Δp . The normal value of Δp_{ij} is -1 if the newly installed replicate replaced replicate j to become closest replicate to base-station i , if j is a newly installed replicate, Δp_{ij} is 1 or 0, otherwise it is 0.

Thus, the resultant change in location management cost from installment of a new replicate j is

$$\Delta C(p) = \sum_{i=1}^I \{ [\sum_{j=1}^J \Lambda_{ij}(\Delta p) d_{ij}] + \mu_i d_{ij} \} \quad (3)$$

The net effect of installing a new replicate is an increase in update costs from all base-stations $\mu_i d_{ij}$ and a decrease in lookup costs with all other replicates

$$\sum_{j=1}^J \Lambda_{ij}(\Delta p) d_{ij} \quad (\Delta p_{ij} \text{ negative in 1 or more base-stns}).$$

The problem therefore is to balance these two costs such that total location management cost is minimized.

3.2 Efficiency analysis

We assume: (i) Each ancestor has n replicates scattered randomly in the network. The chance of having a replicate in a basestation is n/I , with I basestations. (ii) Mean hop count from any base-station to any replicate of a non-leaf agent is D_n for the topology.

The average cost of locating a particular leaf agent is:

$$(1 - \frac{J}{I}) D_n + \dots + (1 - \frac{J}{I}) (1 - \frac{n}{I})^{L-1} D_n \quad (4)$$

The first term is the cost of requesting the agent one level directly above the leaf agent. On average, there is a $(1 - J/I)$ chance that the leaf agent is not in the base-station and requires a request to the agent one level above, recall that J is the number of replicates of the leaf agent. The second term is the cost of requesting the agent two levels directly above the leaf agent. There is a $(1 - J/I)(1 - n/I)$ chance that both the leaf and its parent agent are not present in the base-station.

Without the cost of locating the leaf agent the average total request cost incurred from a call is

$$\sum_{j=0}^J \Lambda_{ij}(p) d_{ij}, \text{ as derived above.}$$

The average total request cost incurred by a call in our system originating from base-station i is therefore

$$\sum_{j=0}^J \Lambda_{ij}(p) d_{ij} + (1 - \frac{J}{I}) D_n + \dots + (1 - \frac{J}{I}) (1 - \frac{n}{I})^{L-1} D_n \quad (5)$$

The additional cost involved in finding the callee's location depends on the number of replicates the leaf's ancestor agents have, the network topology and the number of leaf's replicates. Since $J = \frac{b^L - 1}{b - 1} \approx b^L$, a larger branching factor b implies a smaller cost. Cost function of (5) is also inversely proportional to b .

3.3 Replication Heuristic (Service-Ability)

The service-ability algorithm proposed here, initiated by an agent, attempts to provide an approximation to this cost optimization. It uses a threshold to determine how many base-stations a replicate can serve.

Service-Ability Algorithm for a particular agent

Topology; $i \leftarrow 0$; agent j

while topology is not empty do

tmp

tmpMetric \leftarrow -1

for all the base-stations b inside the topology do

metric = 1 + CalculateMetric(b , threshold, 0, j)

if metric > tmpMetric do

tmpMetric \leftarrow metric; tmp \leftarrow b

endif

$i \leftarrow i + 1$

endloop

replicate j at tmp

delete tmp from topology

endloop

CalculateMetric (base-station, threshold, hop, j) {

metric \leftarrow 0

for all the neighbours n of base-station do

if f (the call rate p_{ij} , hop + 1) \leq threshold

then // $f(x,y)$ is the service ability

// function

metric \leftarrow metric + 1

metric \leftarrow metric + CalculateMetric (n , threshold, hop+1, j)

endif

endloop

return metric

}

The algorithm does not arbitrarily restrict number of replicates of an agent and ensures all base-stations are adequately serviced, depending on request rate and distance from a replicate. The threshold can be modified to control number of replicates.

Assume that the service ability function f runs at $O(1)$, the distance matrix for routing is already computed for each node, and reading d values is $O(1)$. From the pseudo code, algorithm complexity is $O(n^2)$.

4. Results and Analysis

Simulations show the system is efficient for a low call-to-mobility ratio, high call locality, high locality of movement and rapidly changing user patterns. The

simulated system consists of 1,000 mobiles, 10,000 connections (calls between mobiles), within a fixed network of 49 base-stations. There is also a repeativity R , the calling set size of each mobile, used to simulate call locality. We assume each user only calls a fixed set of mobiles, limited by R (default $R = 5$).

Branching Factor: An important parameter is the branching factor b of the agent hierarchy. In Fig. 8 & 9, we see that as b increases, the tree becomes flatter, and the average hop count that a request must travel from the root agent to the leaf agent decreases. When b is 256, the agent hierarchy has only two levels and an average hop count of 3.8352 as against 4.26 when b is 16. This corresponds to result in Equation (5).

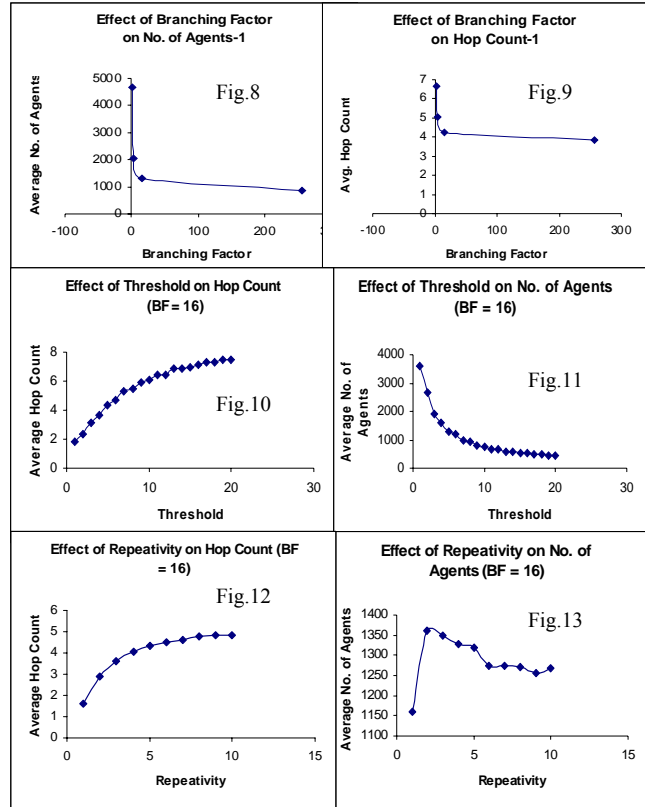
Threshold: Fig. 10 & 11 plot the algorithm threshold versus average hop count. A larger threshold implies an agent services more base-stations resulting in fewer system replicates. When the branching factor is 16, the hop count is around 7 for a threshold of 20. This reflects a network without any replicates. As the threshold is lowered, average hop count decreases dramatically. When the threshold becomes one, every agent has a replicate in every base-station in the network. The hop count is lowered to 1.8 (b is 16). The decrease in hop count is steeper than the decrease in threshold. Simultaneously the number of agents increases to 3578. The update cost (number of agents) for each mobile movement is nearly 10 times that of a network without any replicates. For low call-to-mobility ratios, a high threshold is appropriate, while for high ratios, a low threshold is more efficient. By comparing hop count with number of agents, we see replicates improve average hop count dramatically.

Calling Set Size: This models call locality, the fact that users call a few numbers frequently. In Fig. 12 & 13, it is shown that the system takes advantage of call locality. When the calling set is relatively large, the mean hop count is high. The mean hop count drops by 66.4% to 1.64 (branching factor is 16). The hop count improvement was achieved with virtually no increase in number of agents. The system actually migrates to new base-stations to achieve additional efficiency.

5. Conclusion and Future Work

This paper proposed a novel adaptive location management scheme that uses a hierarchy of location database agents that can self-replicate to other base-stations; developed an analytical cost model of lookup/update costs and a service ability algorithm to approximate this cost optimization. Simulations show that it can rapidly react to user pattern changes. Future work includes development of a protocol to execute the service ability algorithm, evaluating architectural

performance on an unreliable network, automating selection of the threshold values which might vary for different agent levels in the hierarchy, and multicasting location updates. Key ideas may be applied to any application domain involving tracking mobile devices.



6. References

- [1] E. Pitoura and G. Samaras. Locating Objects in Mobile Computing. IEEE Trans. on KDE, 13(4): 571–592, 2001.
- [2] L. Blazevic, L. Buttyan, S. Capkun, S. Giordano, J-P. Hubaux, J-Y. Le Boudec. Self-Organization in Mobile Ad-Hoc Networks: the approach of Terminodes. SSC Technical Report no. 2001, June 2001
- [3] CarNet Project. <http://www.pdos.lcs.mit.edu/grid/>
- [4] D. S. J. De Couto, D. Aguayo, B. A. Chambers, and R. Morris. Performance of Multihop Wireless Networks: Shortest Path is Not Enough, Proceedings of HotNets-I, Princeton, New Jersey, Oct 2002.
- [5] J. Kangasharju, J. Roberts, and K.W. Ross. Object replication strategies in content distribution networks. Proc. of WCW 2001, Boston, MA, June 2001.
- [6] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In Proceedings of the 10th International Conference on Scientific and Statistical Database Management, 1998.
- [7] G. Cho and L. F. Marshall. An Efficient Location and Routing Schema for Mobile Environments. IEEE JSAC, 13(5), June 1995.
- [8] Y. B. Lin. Determining the User Location for Personal Communications Service Networks. IEEE Transactions on Vehicular Technology, 43(3), August 1994.
- [9] J. Jannink, d. Lam, N. Shivakumar, J. Widom, and D.C. Cox. Efficient and Flexible Location Management Techniques for

Wireless Communication Systems. *ACM/Baltzer MONET*, 3(5):361-374, 1997.

[10] O. Wolfson, S. Jajodia, and Y. Huang. An Adaptive Data Replication Algorithm. *ACM TODS*, 22(2):255-314, June 1997.