MLDS: A Flexible Location Directory Service for Tiered Sensor Networks

Sangeeta Bhattacharya *, Chien-Liang Fok, Chenyang Lu, Gruia-Catalin Roman

Department of Computer Science and Engineering
Washington University in St. Louis, USA

Abstract

Many location based services, such as a those used in healthcare facilities to track medical personnel and equipment, need to keep track of mobile entities across wide areas. These services can be realized in a cost-effective and efficient manner using tiered sensor networks comprised of multiple wireless sensor networks connected by an IP network. To simplify the realization of such services, we present MLDS, a Multi-resolution Location Directory Service for tiered sensor networks. MLDS provides a rich set of spatial query services ranging from simple queries about entity location, to complex nearest neighbor queries. Furthermore, MLDS supports multiple query granularities which allow an application to achieve the desired tradeoff between query accuracy and communication cost. We implemented MLDS on Agimone, a unified middleware for sensor and IP networks. We then deployed and evaluated the service on a tiered sensor network testbed consisting of tmotes and PCs. Our experimental results show that, when compared to a centralized approach, MLDS achieves significant savings in communication cost while still providing a high degree of accuracy, both within a single sensor network and across multiple sensor networks.

Key words: Sensor Network, Location Directory, Spatial Query, Mobile Agent

1. Introduction

Many location based services require the capability of keeping track of a large number of mobile entities over a wide area. These services can be realized in a cost-effective and efficient manner using distributed wireless sensor networks connected via IP networks. Let's consider the specific example of co-ordinating doctors over multiple makeshift clinics, set up after a natural calamity. Such clinics are often short of doctors, requiring them to move between the various clinics. In such a scenario, there is often a need to keep track of the doctors, as they move within and between clinics, so that it is possible to find a particular doctor or the nearest available doctor when the need arises. In addition to tracking doctors, tracking staff members and medical equipment would also highly improve the efficiency of such clinics. Wireless sensor networks connected via IP networks help automate people and asset tracking in such scenarios by providing the capability to sense and identify mobile entities over wide areas. Moreover, in scenarios like the one mentioned above, existing infrastructure (e.g. phone lines and cell phone towers) is often destroyed or overloaded, requiring the deployment of wireless sensor networks connected via ad-hoc IP networks to realize such wide-area location based services. However, to fully realize such services, it is essential to develop a location directory service that can (a) efficiently maintain the location of mobile entities as they move *across multiple sensor networks* and (b) support a broad range of *spatial queries* concerning the mobile entities. Our goal is to realize exactly such a service.

The primary contribution of our work is the design, implementation, and empirical evaluation of MLDS, the first Multi-resolution Location Directory Service for *tiered sensor networks*. The key contributions of our work include
– Design of MLDS, which efficiently maintains location information of mobile entities across multiple sensor and IP networks *and* supports a rich set of multi-granular spatial queries
– Implementation of MLDS on tiered sensor networks composed of resource constrained sensor networks and IP networks, and
– Empirical evaluation of MLDS on a tiered testbed of 45 nodes. Our empirical results show that MLDS can main-

* Corresponding author. Address: One Brookings Drive, Box 1045, St. Louis, MO 63130. Phone: 1-314-935-7537

Email addresses: sangbhat@cse.wustl.edu (Sangeeta Bhattacharya), fok@cse.wustl.edu (Chien-Liang Fok), lu@cse.wustl.edu (Chenyang Lu), roman@cse.wustl.edu (Gruia-Catalin Roman).

tain a high degree of accuracy at low communication cost, both within a single sensor network and across multiple sensor networks.

MLDS adopts a hierarchical architecture with multi-resolution information storage in order to support multi-granular queries at considerably low communication cost. While MLDS' hierarchical directory structure bears some resemblance to the cellular network architecture and to the Internet's Domain Name System (DNS), its novelty lies in the fact that it is specifically designed and implemented for tiered sensor networks consisting of resource constrained sensor platforms. In particular, our goal is to minimize communication cost without considerable loss in accuracy. By minimizing communication cost we not only minimize the network energy consumption and therefore extend the network lifetime but we also improve the system performance (i.e. achieve low query latency and high query success rate). Further comparisons between our work and location management services used in cellular and IP networks is given in the related work section (Section 7).

The rest of this paper is organized as follows. We describe the system model in Section 2, the services offered by MLDS in Section 3 and MLDS' design in Section 4. We then present implementation details in Section 5, followed by experimental results in Section 6 and related work in Section 7. Finally, we conclude in Section 8.

2. System Model

MLDS can support multiple wireless sensor networks connected by IP networks. Each sensor network, consisting of stationary *location-aware* sensor nodes and a base station, is assumed to have a unique name that maps to the base station's IP address. We assume that the sensor networks track mobile entities in the physical environment using existing tracking algorithms [14,12,1,24,17] coupled with or without RFID technology. These algorithms may use existing sensor network localization schemes [16,10] to compute the exact location of the mobile entities. Furthermore, in our implementation of MLDS, we assume that mobile entities are represented by light-weight mobile agents [1] in the sensor network. A mobile agent is a software process that can migrate across nodes while maintaining its state. Mobile agents present a convenient way of representing mobile entities (e.g. cars, people) in the sensor network [5]. For instance, in the make-shift clinic example described above, mobile agents may be created to shadow the doctors. Users can then query the locations of doctors by querying the locations of the corresponding mobile agents, through MLDS. Note that even though MLDS is implemented to work with mobile agents, it can be easily extended to work with other programming models for mobile entity tracking such as EnviroSuite [14] and others based on message passing [12,1,24,17]. For example, in EnviroSuite, a mobile entity is mapped to a dynamically instantiated object with a unique ID in the sensor network. MLDS can be easily adapted to keep track of these mobile objects instead of mobile agents.

3. Services

MLDS supports four types of flexible spatial queries that include (i) finding the location of a particular agent, (ii) finding the location of all agents, (iii) finding the number of agents and (iv) finding the agent that is closest to a particular location. To meet the needs of diverse applications, all of these queries support different scopes and granularities that can be specified by the application. MLDS supports two query scopes, (i) *local scope* i.e. within a single sensor network and (ii) *global scope* i.e. across multiple sensor networks. It supports three query granularities, *fine*, *coarse* and *network*. The query result of a fine query is based on the exact locations of the mobile agents while the query result of a coarse query is based on the approximate locations of the mobile agents. The query result of a network query, on the other hand, is based only on the knowledge of the sensor networks that the agents are in. MLDS supports queries issued from both within a sensor network and from outside a sensor network (e.g. by an agent or user on the IP network).

Most of the queries supported by MLDS take in the parameters S and G which specify the scope and granularity of the query, respectively. Mobile agents may be divided into classes and hence some of the queries also take in the parameter C that specifies the "class" of a mobile agent, thereby limiting the query to that agent class. The API of the four spatial queries are as below:

(i) **GetLocation(**id, S, G**)** returns the location of agent with ID *id*.
(ii) **GetNum(**C, S**)** returns the number of class C agents.
(iii) **GetAll(**C, S, G**)** returns the location of all class C agents.
(iv) **GetNearest(**C, L, S, G**)** returns the location of the class C agent that is closest to location L.

4. Design

MLDS is designed for location based services that involve tracking of mobile entities (e.g. people, equipment). Due to the high mobility of entities in these systems, the location information update rate is expected to be much higher than the query rate in these systems. Hence, MLDS is specifically tailored for systems in which the location information update rate is greater than the query rate. In order to optimize the operation of such systems, MLDS adopts a hierarchical architecture with multi-resolution information storage. As a result (1) it can support multi-granular spatial queries which enables applications to achieve the desired

[1] The light-weight mobile agents do not add significant overhead to the system as shown in previous empirical results [5].

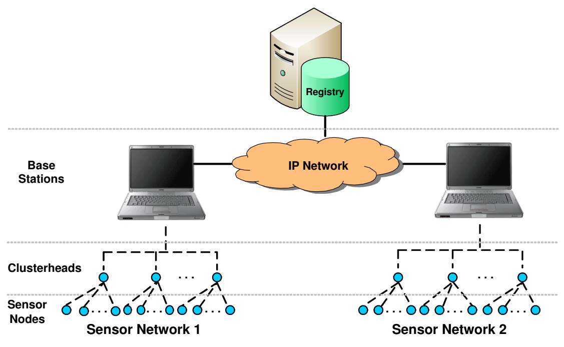


Fig. 1. MLDS Architecture.

tradeoff between location information accuracy and communication cost, (2) location information update is not always propagated to the upper tiers of the hierarchy, which significantly reduces communication cost and (3) queries are answered at the closest tier of the hierarchy that meets the query scope and granularity requirements, thus reducing both communication cost and query latency.

4.1. *Architecture*

MLDS has a four tiered hierarchical architecture as shown in Figure 1. The topmost tier of the hierarchy is a central registry [2] that stores information about the different sensor networks. The base stations of each sensor network are connected by IP networks and form the second tier of the hierarchy. The other two tiers of the hierarchy lie within the sensor networks and are formed by a clustering algorithm that groups the sensor nodes into non-overlapping 1-hop clusters. The clusterheads of these clusters form the third tier of the hierarchy while the cluster members form the fourth tier. Note that the system is heterogeneous, with nodes at higher tiers having more resources than nodes at lower tiers. For example, the clusterheads are resource constrained sensor nodes; the base stations are more powerful computers such as PCs or stargates; while the registry, is a server or server cluster.

MLDS stores location information at different resolutions at different tiers of the hierarchy. Clusterheads store the exact location of the agents in their cluster while base stations store only the IDs of the clusters that the agents in their network belong to. The registry on the other hand stores the IDs of the networks (denoted by the network base station IP address) that all agents in the system belong to. A base station also maintains the location of the clusterhead and the minimum bounding rectangle (MBR) of each cluster in its network. While the registry also stores the MBR of all the connected sensor networks. The network and cluster MBRs are needed to answer nearest neighbor queries, as explained later in Section 4.3.

[2] A distributed registry will further improve the scalability of MLDS and is part of our future work.

4.2. *Location Information Maintenance*

Since MLDS maintains less accurate information at higher tiers of the hierarchy, location information is not always propagated to the upper tiers, which significantly reduces communication cost. In the following we describe how MLDS maintains agent location information at different tiers of the hierarchy.

A node hosting an agent periodically sends location update messages to its clusterhead, at an interval ΔT. Periodic location update messages are required to maintain the directory in the face of node/agent failures. The location update messages contain the agent ID, class and location [3]. When a clusterhead receives a location update message, it first updates its directory with the agent information. If the agent has just entered its cluster, it then sends a message to the base station containing the agent ID and class, and its own ID (note that the agent location is not sent to the base station). The base station in turn updates its directory on receiving this information and also updates the registry if an entry for the agent did not previously exist in its directory.

Agent location information at a clusterhead expires after a period of $2\Delta T$. Thus, if a clusterhead does not receive location update messages from an agent for a period of $2\Delta T$, it assumes that the agent has left its cluster and removes the agent from its directory. A clusterhead may therefore have stale information for a maximum period of $2\Delta T$. This design trades off accuracy for lower communication cost and is preferred over other options that provide higher accuracy at higher communication costs.

4.3. *Query Processing*

MLDS answers a query at the closest tier of the hierarchy that meets the query scope and granularity requirements. For queries issued from within(outside) a sensor network, the closest tier would be the lowest(highest) tier of the hierarchy that meets the query scope and granularity requirements. This approach reduces both communication cost and query latency. All queries issued by an agent from within a sensor network, also called in-network-queries, are first sent to the clusterhead of the cluster that the agent is in. If the query type is GetLocation or GetNearest, the clusterhead checks if it can answer the query. If it can, it sends the query reply to the querying agent, otherwise it forwards the query to the base station. On the other hand, if the query type is GetAll or GetNum the query is directly forwarded to the base station. The base station processes the query and sends the reply to the clusterhead that sent the query, which in turn forwards the reply to the querying agent. Queries issued by an external agent or user on the IP network, henceforth called out-of-network-queries, are

[3] We set the location of an agent to the location of the host node in our design.

sent to the relevant base stations that process the queries and route the result back to the querying agent/user.

We now explain how MLDS processes a query when the query is issued by an agent within a sensor network. Since a base station processes in-network-queries the same way that it processes out-of-network-queries, the later process can be derived from the description of the former, and hence is not explicitly described. Moreover, due to space limitations, we only describe the GetNearest and GetLocation query types in detail. The GetNum query is the simplest of all queries and just involves querying the base station, while the GetAll query is a simple extension of the GetLocation query. In the following discussion, we assume that the ID of the querying agent is q. We also assume that for any agent with ID i, C_i denotes the clusterhead of the cluster that agent i is in and B_i denotes the base station of the network that agent i is in.

4.3.1. *GetLocation*

When clusterhead C_q receives a GetLocation(id, S, G) query from agent q, it checks if agent id is in its cluster. If the agent is in its cluster, it sends a query reply to agent q. If agent id is not in agent q's cluster, then C_q forwards the query to the base station B_q. On receiving this query, B_q checks if agent id is in its network. If the agent is in the network, B_q sends a reply containing either the location of the clusterhead C_{id}, if the query is coarse or the exact location of the agent, which it obtains from C_{id}, if the query is fine. In the case that agent id is not in the local network (i.e. $B_{id} \neq B_q$), B_q finds out B_{id} from the registry, and forwards the query to B_{id}. B_{id} processes the query as explained above and sends the result to B_q. B_q sends the query result to C_q, which forwards it to agent q.

4.3.2. *GetNearest*

When C_q receives a GetNearest(C, L, S, G) query, it checks if there are class C agents in its cluster. If there are such agents, C_q finds the agent that is geographically closest to location L and sends a reply to agent q. If there are no class C agents in the cluster, C_q forwards the query to B_q.

Let's first see how B_q handles local queries. If the query is coarse, B_q just returns the location of the clusterhead, whose cluster contains class C agents and whose location is geographically closest to location L. However, if the query is fine, B_q finds the answer by using the branch and bound technique [21]. The intuition behind this technique is to query only those clusters that contain class C agents whose locations could be closest to location L. These clusters are found by first obtaining a set of clusters that contain class C agents and then looking at the minimum and maximum distances of the MBRs of the clusters in this set, from L. Clusters whose minimum distances are greater than the maximum distance of the cluster that has the least minimum distance, are discarded. B_q queries the clusterheads of the remaining clusters and waits for a certain time period to hear from them. When B_q hears from all the clusterheads (before the end of the time period) or at the end of the time period, B_q computes the agent that is closest to location L based on the information obtained in the query replies and sends the reply to C_q. Note that although the MBR of a cluster does not accurately represent the cluster boundary, it is preferred over other complex methods like the convex hull due to its low computational complexity.

B_q handles global queries similarly, by first looking up the registry to find the networks that contain class C agents and then applying the above branch and bound technique at the network level. Note that by design, this query returns the approximate geographically closest agent. This design achieves lower communication cost by trading off accuracy.

4.4. *Analysis*

In this section, we present a back-of-the-envelope analysis of the communication cost incurred by MLDS in a single sensor network. The communication cost incurred by MLDS across multiple sensor networks is primarily the single sensor network communication cost, since the IP network communication cost is relatively low due to its higher bandwidth. We also compare the communication cost incurred by MLDS with that incurred by a centralized approach, called the Central Directory (CD) approach. In CD, location information is stored in a central directory that is maintained outside the sensor network, at a base station. Thus, unlike MLDS, all location information and queries are sent to the base station in CD.

4.4.1. *Location Update Cost*

The cost of a single location update within a sensor network in MLDS is approximated by $c(1 + ph)$, where c is the 1-hop communication cost, p is the probability that the agent moved to a new cluster and h is the average number of hops to the base station. The first part of the equation is the cost of updating the clusterhead, which is just c, since MLDS has one hop clusters. The second part of the equation, cph, is the cost of updating the base station when the agent moves to a new cluster. From this equation we see that the lower the probability of an agent changing clusters, the lower is the communication cost. In contrast, the location update cost in CD is ch, since all location updates are sent to the base station in CD. Thus, the ratio of the location update costs incurred in MLDS and CD, is $\frac{1+ph}{h}$ which converges to p as $h \to \infty$. Therefore, MLDS may have a significantly lower location update cost than CD, especially when agents move locally (within a cluster) most of the time. Locality of movement is a common feature of many location based service applications. For example, a sensor network deployed in a building in order to track people, will most probably have a hierarchical structure, with sensors on each floor forming a different cluster. Since people located on one floor generally tend to move about more

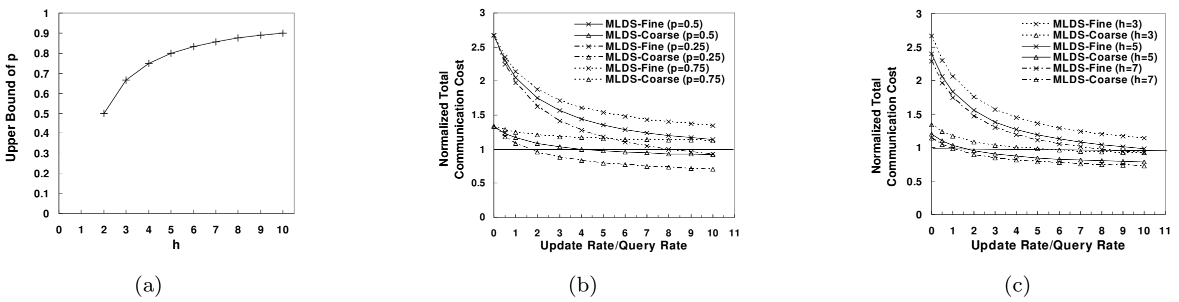


Fig. 2. Analysis of MLDS' communication cost, relative to that of CD. (a) Maximum value of p, for a given h, at which MLDS' location update cost is lower than that of CD; (b)-(c) Relative effect of update and query rate on communication cost of MLDS and CD for (b)$h = 3$, and varying p and (c)$p = 0.5$. and varying h.

frequently on that floor, the application will display locality of movement.

The maximum values of p at which MLDS' location update cost is lower than that of CD, for different values of h, is shown in Figure 2(a). As seen in the figure, MLDS can achieve lower location update cost than CD in sensor networks with diameters as low as 2 hops (if $p \leq 0.5$). For a network size of 5 hops, MLDS achieves lower update cost as long as $p \leq 0.8$. Thus, MLDS can achieve a lower location update cost than CD in a reasonably small network even if an agent moves between clusters 80% of the time.

4.4.2. *Total Cost*

In this section, we compare the total communication cost of MLDS with that of CD, when GetLocation queries are issued by an agent in the network. The total communication cost consists of the location update cost and the GetLocation query cost. The cost of a query in CD is $2ch$. Whereas the worst case cost of a GetLocation query in MLDS is $2c(1 + h)$, if the query is coarse and $4c(1 + h)$ if the query is fine. Note, we only consider the worst case query cost in MLDS. The query cost is much lower ($2c$) when the query is answered by the clusterhead. Thus, the worst case cost of coarse queries in MLDS is close to the query cost in CD. The cost of fine queries is low, when answered locally by the clusterhead, but high otherwise. Given the query costs, the location update costs (from the previous subsection), the location update rate R_U and the query rate R_Q, the worst case total communication cost of MLDS is $c[(1 + ph)R_U + 4(1 + h)R_Q]$ when only fine queries are issued and $c[(1 + ph)R_U + 2(1 + h)R_Q]$ when only coarse queries are issued. The total communication cost of CD, on the other hand, is $ch(R_U + 2R_Q)$.

Figure 2(b) shows the worst case total communication cost of MLDS, normalized by the total communication cost of CD, for varying ratios of location update rate (R_U) and query rate (R_Q) and different values of p, when $h = 3$. From the figure, we see that as expected, the communication cost of MLDS is lower when coarse queries are issued than when fine queries are issued. Moreover, we see that when mobile agents move locally most of the time ($p \leq 0.5$), MLDS performs better than CD when the update rate is greater than the query rate. For example, when the mobile agents change clusters 25% of the time and only coarse queries are issued, MLDS performs better than CD when the update rate is about 1.5 times the query rate. This is expected since the cost of coarse queries in MLDS is similar to the query cost in CD while the location update cost in MLDS is lower than that in CD, especially when agents display locality of movement. When fine queries are issued, MLDS' total communication cost becomes lower than that of CD only when the update rate is considerably higher than the query rate. This is expected since the cost of fine queries in MLDS is higher than the query cost in CD. Finally, the ratio of R_U/R_Q at which MLDS achieves a lower communication cost than CD, decreases as p reduces. Thus, MLDS achieves higher savings for smaller values of p. MLDS is therefore most suitable for applications that (1) have a high entity mobility rate and relatively low query rate, (2) can tolerate coarse query results and (3) have locality of movement.

Figure 2(c) shows how the normalized total communication cost of MLDS varies with R_U/R_Q, for different values of h, when $p = 0.5$. From the figure, we see that as h increases, MLDS achieves a lower communication cost than CD, at lower values of R_U/R_Q. MLDS is thus more scalable than CD.

4.5. *Discussion*

In this section, we discuss how MLDS already handles or can be extended to handle some practical issues in wireless sensor networks.

4.5.1. *Handling Node Failures*

MLDS relies on the clustering process to handle clusterhead failures. The clustering algorithm detects clusterhead failures and selects new clusterheads and forms new clusters if required. However, the location information maintained by a clusterhead is lost when it fails. This may affect performance for a maximum period of ΔT. Failure of non-clusterhead nodes do not affect the clusters but may affect network connectivity. MLDS assumes that the failure of such nodes is handled by a lower routing layer that maintains routes between clusterheads and the base station.

4.5.2. *Handling Different Query and Update Rates*

MLDS is currently optimized for systems that have a higher location update rate. However, it can be extended to dynamically adapt to the query and update rates such that it performs well under all conditions. This can be done by using a push-pull strategy that dynamically adjusts the storage-location and granularity of location information, based on the query and update load. For example, when the query rate at the base station is high, and the majority of the queries are fine queries, selective location information can be maintained at fine granularity (instead of coarse granularity) at the base station. Whereas, if the query rate at a clusterhead is high, corresponding location information can be pulled by the clusterhead (from the base station) such that most of the queries are answered locally, thereby reducing query cost. We leave the details and evaluation of this approach as future work.

4.5.3. *Energy Efficiency*

Further energy efficiency can be achieved via node sleep scheduling and load balancing. We have not considered node sleep schedules in our current design but this can be easily incorporated by maintaining a backbone of active nodes consisting of clusterheads and nodes connecting the clusterheads to the base station. The rest of the nodes in the network can maintain a sleep schedule without significantly affecting the performance. Furthermore, the clustering algorithm can be modified to rotate clusterheads for load balancing and for uniform energy usage among nodes. The location information maintained by the old clusterhead can be transfered to the new clusterhead, to prevent performance degradation during the change.

4.5.4. *Aggregation*

We have not incorporated in-network aggregation of location update messages in our design. Aggregation would reduce the communication cost but introduce delays in the data collection process, thus affecting the data freshness. Therefore, aggregation can be incorporated as an application-dependent tradeoff.

5. Implementation

We have implemented and integrated MLDS with Agimone, a unified middleware that integrates sensor and IP networks. In this section, we first give an overview of Agimone and then describe the implementation details of MLDS.

5.1. *Agimone*

Agimone [9] combines two mobile agent middlewares called Agilla [5] and Limone [4]. Agilla is optimized for resource-constrained sensor networks and is implemented in nesC on the TinyOS platform. Limone is designed for more powerful nodes (e.g. PDAs, stargates and laptops) connected by IP networks and is implemented in Java on standard Java Virtual Machines (JVMs). In Agimone, creation and deployment of mobile agents within a sensor network is done using Agilla, while migration of mobile agents across sensor networks via an IP network, is done using Limone. Agilla provides primitives for an agent to move and clone itself across sensor nodes while carrying its code and state, effectively reprogramming the network. To facilitate inter-agent coordination within a sensor network, Agilla maintains a local tuple space and neighbor list on each sensor node. Multiple agents can communicate and coordinate through local or remote access to tuple spaces. In Agimone, the base stations communicate through Limone tuple spaces maintained at the base stations. Specific Limone agents called AgimoneAgents that reside at the base stations provide an interface between Agilla and Limone and enable the migration of Agilla agents across an IP network. Agimone maintains a central registry for the registration and discovery of sensor networks over the IP network.

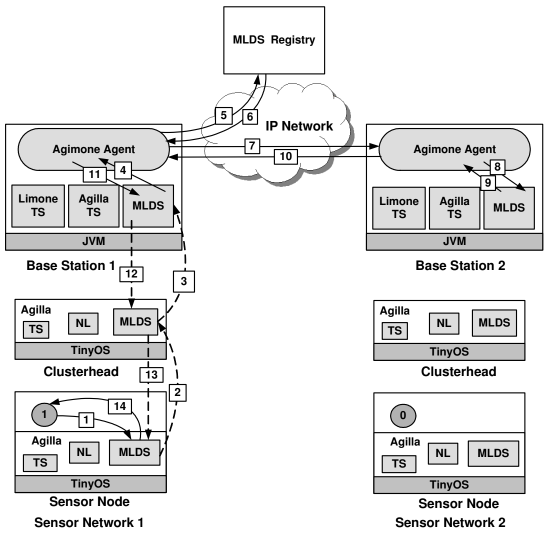
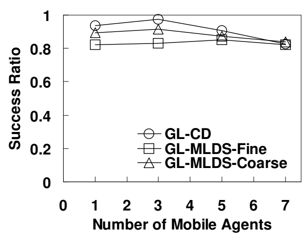


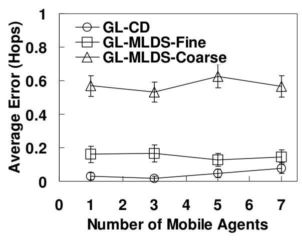
Fig. 3. Interaction between MLDS and Agimone modules when the **GetLocation(0, "global", "coarse")** query is issued by agent 1.(TS: Tuple Space, NL: Neighbor List)

5.2. *Integration of MLDS with Agimone*

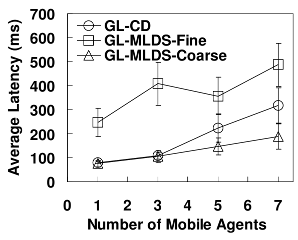
MLDS is integrated with the Agimone modules that run on the sensor nodes and base stations. It is implemented in nesC on the sensor nodes and in Java on the base station. MLDS also extends the Limone registry to serve as the registry for its location directories. Figure 3 shows the interaction between the MLDS and Agimone modules at different tiers of the hierarchy when the **GetLocation(0, "global", "coarse")** query is issued by an agent with ID 1. Agent 1 is in sensor network 1 while agent 0 is in sensor network 2, in the figure. Note that the agents are Agilla agents. Steps 1-3 in the figure show the query message being propagated up the hierarchy to the base station. Once it reaches the MLDS module at the base station, control



(a) Success Ratio



(b) Average Error



(c) Avg. Query Latency

Fig. 4. Performance of local GetLocation queries.

is transfered to the AgimoneAgent (step 4), since agent 0 is not found in sensor network 1. The AgimoneAgent then queries the registry to find out which network agent 0 is in (steps 5-6). Once it finds that out, it sends the query to the AgimoneAgent at the base station in sensor network 2 (step 7). The AgimoneAgent in base station 2 queries the local MLDS module to obtain the result of the query (steps 8-9) and sends the result back to the AgimoneAgent in base station 1 (step 10). The AgimoneAgent in base station 1 then sends the query reply to the local MLDS module (step 11). After that, the query reply is forwarded down the hierarchy to agent 1 (steps 12-14).

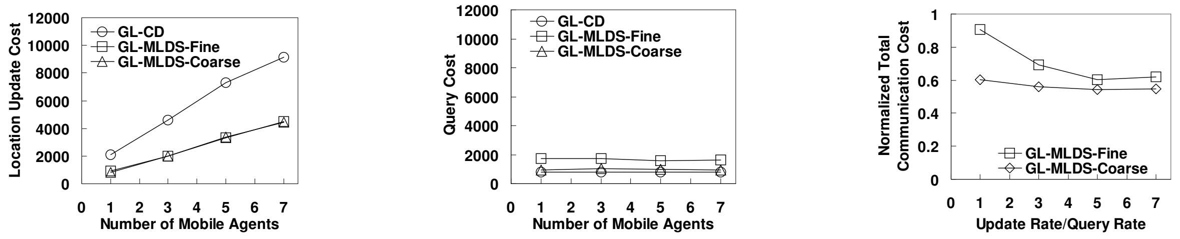
MLDS adapts Agimone's sensor-network-discovery and neighborhood-maintenance mechanisms to build and maintain its hierarchical structure. The upper two tiers of the hierarchy are formed via the sensor-network-discovery process in which the base stations register themselves with the registry. The lower two tiers of the hierarchy that lie within individual sensor networks are formed via a simple clustering algorithm. MLDS leverages Agimone's neighborhood-maintenance process to achieve clustering at minimum communication cost. Agimone maintains neighborhood information at each node through a periodic beaconing process. Each node periodically broadcasts *beacon messages* containing its ID and hop count to the base station. The hop count information is used for routing messages to the base station. MLDS integrates the clustering process with Agimone's neighborhood-maintenance process by requiring a node to include its clusterhead ID in its beacon messages.

The clustering process is a continuous process that starts when the base station (which is a clusterhead by default) broadcasts a beacon with the clusterhead ID set to its own ID. Nodes receiving this beacon message decide whether they should become clusterheads or join the base station's cluster. These nodes in turn include their clusterhead IDs in their beacon messages. These beacon messages trigger the clustering process at neighboring nodes and so on. The clustering process at a sensor node is thus initially triggered by beacon messages. On receiving a beacon message, a node decides whether it should become a clusterhead or whether it should join the cluster of a neighboring clusterhead. If the beacon message is from a cluster member, the node becomes a clusterhead. Whereas if the beacon message is from a clusterhead, the node joins the cluster of the clusterhead. If the node hears from more than one clusterhead, it joins the cluster of the clusterhead that has the least cost, where cost is determined by the cluster size (number of cluster members; larger clusters have less cost) and the clusterhead's hop count to the base station. A node may change its decision up until it announces its decision to its neighbors via a beacon message. After that, the node can change its decision only if there is a change in its local topology. In a special case, a clusterhead may change its decision and decide to join a neighboring cluster if it finds that it does not have any cluster members after a certain time interval (3 beacon periods). Local topology changes that affect cluster formation include failure of clusterheads or changes in connectivity between clusterheads and cluster members. When either of this happens, affected cluster members stop receiving beacon messages from their clusterheads. These cluster members then either join the cluster of a neighboring clusterhead or become clusterheads in case they do not have any neighboring clusterheads. The clustering process thus dynamically adapts the clusters to changes in the network topology. Clusterheads notify the base station about the initial cluster formation as well as about any changes in the clusters by sending their IDs and the MBRs of their clusters. Cluster information is also sent to the base station periodically at a low frequency to handle message loss.

6. Experimental Results

We evaluate MLDS via two sets of experiments. The first set of experiments compares MLDS' performance to the centralized directory (CD) approach within a single sensor network. Recall that in CD, location information is stored only at the base station. All location information and queries are thus sent to the base station in CD. The second set of experiments evaluate MLDS' ability to keep track of mobile agents across multiple sensor networks. We evaluate only the performance of the **GetLocation** and **GetNearest** queries in our experiments. Since the GetNum query is the same in both MLDS and CD and the GetAll query is just an extension of the GetLocation query, we do not evaluate them. In both experiments, sensor nodes are arranged in a grid. Multi-hop communication between the nodes is achieved by using a filter at the nodes that accepts packets only from neighboring nodes on the grid.

We use the following four metrics to evaluate query performance in our experiments. (1) **Success Ratio**: the ra-



(a) Location Update Cost (b) Query Cost (c) Communication Cost

Fig. 5. Communication Cost of local GetLocation queries.

tio of the number of queries that returned the accurate result and the total number of queries issued. Network query results are considered accurate if they contain the correct network name; coarse query results are considered accurate if they contain the correct cluster information and fine query results are considered accurate if they contain the correct agent location. (2) **Average Error**: the average error among all queries for which a query result is received, in term of hops. Fine query error is computed as the number of hops between the location returned in the query result and the actual location of the agent. Coarse and network query error is computed as the number of hops the agent is from the clusterhead and from the base station, respectively. (3) **Communication Cost** includes *Location Update Cost* and *Query Cost*. Location Update Cost is the total number of location information messages sent per experiment while Query Cost is the total number of query messages and query result messages sent per experiment. (4) **Average Query Latency**: the average query latency among all queries for which a query result is received. Query latency is the time interval between the issuance of a query and the arrival of the query result, at the querying node. We present 90% confidence intervals for both average error and query latency.

6.1. *Single Sensor Network*

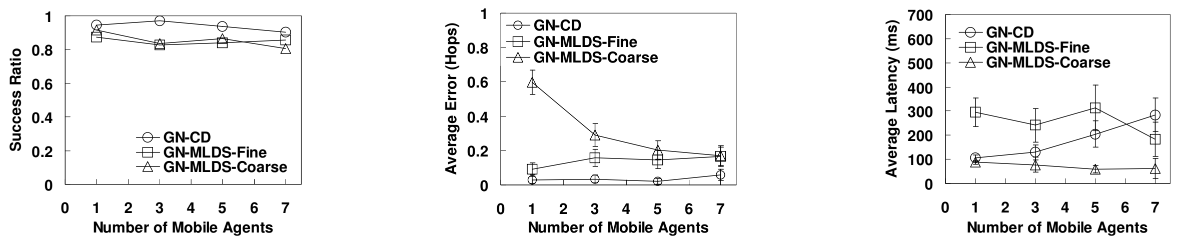
This set of experiments is carried out on a testbed of 24 tmotes, arranged in a 6×4 grid, with a PC as the base station. The PC is connected to a corner tmote, which serves as the gateway node. All tmotes are connected to a PC via USB in order to collect trace data. In each experiment, we deploy one stationary agent two hops from the base station, and n ($1 \leq n \leq 7$) mobile agents. The mobile agents are programmed to follow a random movement pattern over the sensor network at a speed of 1 hop every 5s. 200 queries were issued in each experiment, at the rate of 0.2 queries/s by the stationary agent. Note that by varying the number of mobile agents from 1 to 7 in the experiments, we vary the total location update rate from 0.2 updates/s to 1.4 updates/s and hence evaluate the performance of MLDS under varying network workloads.

In these experiments, we evaluate the performance of the **GetLocation** and **GetNearest** queries, at both fine and coarse granularities, in MLDS. However, only fine queries are evaluated in CD since it does not support coarse queries. We refer to the GetLocation query in CD as GL-CD, and the GetLocation fine and coarse queries in MLDS as GL-MLDS-Fine and GL-MLDS-Coarse, respectively. Similarly, the GetNearest queries are referred to as GN-CD, GN-MLDS-Fine and GN-MLDS-Coarse.

6.1.1. *GetLocation Query Results*

Figures 4 and 5 show the results obtained for the GetLocation query. From Figure 4(a) we see that the success ratio of GL-CD is higher than that of GL-MLDS-Fine, when there are fewer mobile agents in the network. GL-MLDS-Fine has a lower success ratio partly because a clusterhead retains outdated location information of an agent that has left its cluster, for a maximum time period $2\Delta T$. Interestingly, as the number of mobile agents increases, the success ratio of GL-CD decreases and approaches that of GL-MLDS-Fine. This is because as the number of mobile agents increases, the number of location information messages also increases. Since all these messages are sent to the base station in CD, there is an increased number of collisions and message loss in the network, which lowers the success ratio of GL-CD. In contrast, the success ratio of GL-MLDS-Fine remains almost constant with the increase in the number of mobile agents, due to its hierarchical architecture which limits the number of messages that are sent to the base station. The success ratio of GL-MLDS-Coarse is higher than that of GL-MLDS-Fine and only slightly lower than that of GL-CD. However, since GL-MLDS-Coarse returns an approximate location, its average error is higher than that of GL-MLDS-Fine, as shown in Figure 4(b). The query reply error of GL-MLDS-Coarse is mostly 1 hop, since MLDS constructs 1-hop clusters. Figure 4(c) displays the query latencies. As expected, GL-MLDS-Fine has the longest query latency since most queries and query results of this type take a longer path. The query latencies of GL-CD and GL-MLDS-Coarse are nearly the same when there are few mobile agents in the system. However, the query latency of GL-CD becomes higher than that of GL-MLDS-Coarse when the number of agents increases, as a result of increased network workload.

Figures 5(a), 5(b) and 5(c) show the location update cost, query cost and total communication cost incurred by the GetLocation queries, respectively. From Figure 5(a) we see that MLDS achieves about 55% savings in location update



(a) Success Ratio (b) Average Error (c) Avg. Query Latency

Fig. 6. Performance of local GetNearest queries.

cost when compared to CD. This is due to MLDS' hierarchical architecture, by virtue of which a large number of location information messages are only sent to the clusterheads and are not forwarded to the base station. Figure 5(b) shows that the query cost of fine queries is higher in MLDS than in CD. This is because GL-MLDS-Fine queries that are routed to the base station, get further routed to a clusterhead. Likewise, the query results of these queries take a longer route to reach the querying agent. Comparatively, the query cost of GL-MLDS-Coarse is much lower and is close to that of GL-CD, since these queries are routed only up to the base station.

Overall, MLDS achieves significantly lower total communication cost than CD as shown in Figure 5(c). The figure shows the total communication cost of MLDS normalized by the total communication cost of CD for varying ratios of total update rate R_U and query rate R_Q. Note that the update rate increases due to the increase in the number of mobile agents in the system. From the figure, we see that the total communication cost of MLDS is lower than that of CD even when the update rate is the same as the query rate and decreases further as the update rate becomes higher than the query rate. We also see that the normalized communication costs of both GL-MLDS-Fine and GL-MLDS-Coarse, obtained experimentally, follow the same trend as the theoretical results that are shown in Figure 2(b). The experimental results however differ from the theoretical results since the value of p is not fixed in the experiments.

6.1.2. *GetNearest Query Results*

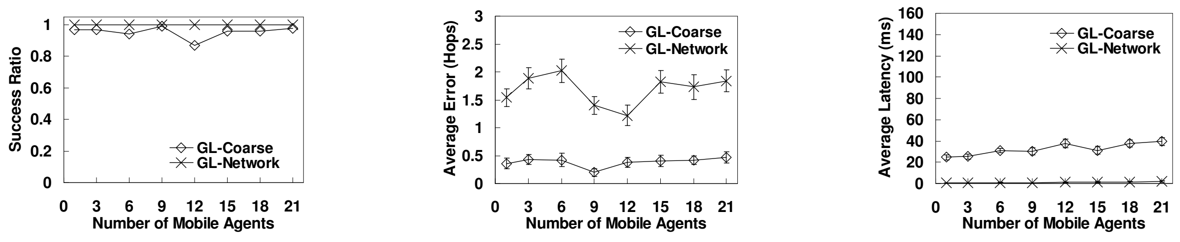
Figure 6 shows the performance of the GetNearest queries. From Figure 6(a) we see that the success ratios of GN-MLDS-Fine and GN-MLDS-Coarse are almost the same, and remain above 80%, irrespective of the number of mobile agents in the network. The average errors of GN-CD and GN-MLDS-Fine reflect the same trend as their success ratios, as shown in Figure 6(b). Interestingly, the average error of GN-MLDS-Coarse is higher than that of GN-MLDS-Fine when there are fewer mobile agents in the system, but decreases as the number of mobile agents in the system increases. This is because as the number of mobile agents increases, the probability of a mobile agent being in the same cluster as the querying agent also increases and so more queries are answered directly by the clusterhead of the cluster that the querying agent is in. Thus, with the increase in the agent density, a higher percentage of the coarse query replies contain exact agent locations, which in-turn reduces the error.

The query latency of GN-MLDS-Fine is higher than that of GN-CD when there are few mobile agents in the network, as shown in Figure 6(c). However, as the number of mobile agents increases, the query latency of GN-CD increases considerably whereas the query latency of GN-MLDS-Fine decreases. The query latency of GN-MLDS-Fine becomes less than that of GN-CD when there are 7 mobile agents in the network. The increase in the query latency of GN-CD with the increase in the number of mobile agents is a result of increased network workload. The reason for the decrease in query latency of GN-MLDS-Fine with the increase in mobile agents in the network is the increase in the percentage of queries that get answered locally by the clusterhead. This same reason also causes the decrease in the query latency of GN-MLDS-Coarse as the number of mobile agents in the network increases. Thus, the GetNearest query benefits significantly from local responses, made possible by MLDS' hierarchical architecture. The benefit is not only decreasing query latency but also decreasing query cost (not shown here), with increasing agent density.

6.2. *Multiple Sensor Networks*

We now evaluate MLDS' performance across multiple sensor networks. In these experiments, mobile agents move between three sensor networks via an IP network running over 100Mbps Ethernet. The IP network is private with a single Linksys WRT54G router and an 8 port switch. The experiments are carried out on a testbed of 45 tmotes, equally divided into three sensor networks arranged in a 5×3 grid. Each sensor network has a PC connected via USB to one of its corner tmotes that serves as a base station. These PCs are connected to each other via the IP network. A fourth PC on the IP network serves as the registry.

Evaluating MLDS' performance requires comparing its results with the ground truth. The ground truth is obtained by connecting every tmote except those directly attached to a base station to the registry PC via USB. The tmotes are programmed to send trace messages identifying key events like agent movement and query activities over their USB port. The registry PC monitors these connections for incoming trace data and saves them into a file. In addition,



(a) Success Ratio (b) Average Error (c) Avg. Query Latency

Fig. 7. Performance of global GetLocation queries.

it also accepts trace messages over the IP network, which the base stations use to record trace messages generated by the tmotes they are attached to. The registry PC serves as a central aggregation point for the trace data. Each trace event is time stamped and saved for off-line analysis.

Like the single sensor network experiments, we evaluate only the performance of the GetLocation and the GetNearest queries. Both the network and coarse granularity versions of the queries are evaluated. In each of these experiments, the workload is varied by varying the number of mobile agents in the system from 1 to 21 in increments of 3. The mobile agents move 10 hops randomly in a sensor network before migrating to another sensor network selected randomly and repeating. Initially, the mobile agents are distributed evenly across the three sensor networks. The GetLocation and GetNearest queries are issued at a rate of 1 query every 5s by an external agent running on the registry PC. Note that this differs from the single network experiments where the querier is located within the sensor network. By placing the querier on the registry, the query messages only travel down the hierarchy. Scenarios where the query messages travel up the hierarchy are already evaluated in the single-network experiments. Each experiment is repeated 100 times.

6.2.1. *GetLocation Query Results*

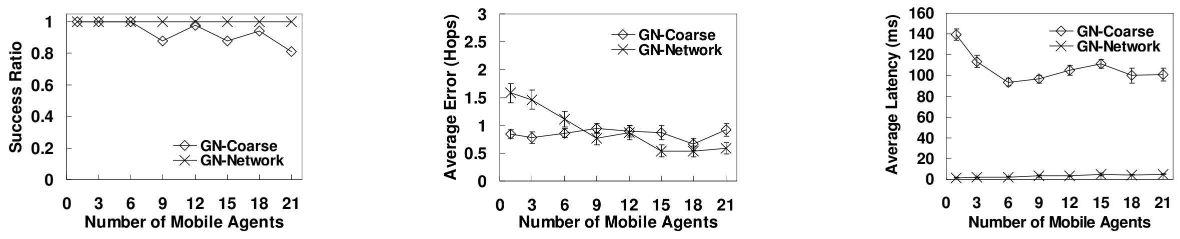
In these experiments, a querier located on the registry periodically issues a GetLocation query for a particular mobile agent (termed the *target agent*) within the sensor networks. The success ratio of the GetLocation query is shown in Figure 7(a). Both the coarse (GL-Coarse) and network (GL-Network) granularity versions of the query achieve nearly perfect success. GL-Coarse has a slightly lower success ratio because it attempts to return a more accurate location of the agent. However, it has approximately 3 times lower error, as shown in Figure 7(b). Notice that GL-Network has a higher average error variance and that GL-Coarse has an average error variance of less than one. This is because an agent may be multiple hops away from the network base station, but can be at most one hop away from its cluster head. GL-Coarse has significantly longer latency as shown in Figure 7(c). The latency of GL-Network is negligible since the querier is located on the registry and can query the registry locally to determine which network the target agent is in. For GL-Coarse, the agent must first lookup which network the agent is in, then query that network's base station to determine which cluster the agent is in. As the number of mobile agents increases, the latency also increases due to increased network congestion.

6.2.2. *GetNearest Query Results*

These experiments are the same as the GetLocation experiments except that the target agent in these experiments is a stationary agent that resides two hops away from the base station on one of the sensor networks. The querier on the registry periodically searches for the mobile agent closest to the target agent. The results of the experiments are shown in Figure 8. As the number of mobile agents increases, the success ratio of the coarse granularity version of the query (GN-Coarse) decreases due to network congestion preventing updates from propagating up the hierarchy, as shown in Figure 8(a). On the other hand, the network granularity version of the query (GN-Network) almost always succeeds since it involves at most 1 call to the registry. The average error of GN-Coarse remains roughly less than 1 hop regardless of the number of agents as shown in Figure 8(b). This is expected since the cluster members are at most 1 hop away from the cluster head. The average error of GN-Network is also close to 1, but is dependent on the size of the network. As the number of mobile agents increases, the probability of finding an agent in the same network as the target increases, decreasing the latency of GN-Coarse, as shown in Figure 8(c). The latency of GN-Network is negligible because in our experiments, an agent is always present in the target agent's network and hence the queries are always answered locally by the base station.

6.3. *Summary*

Our experimental results show that MLDS successfully keeps track of mobile agents within a single sensor network and across multiple sensor and IP networks. MLDS consistently achieves success ratios above 80% in all our experiments, at significantly lower total communication cost than the CD approach. In particular, coarse in-network-queries supported by MLDS achieve the lowest communication cost and query latency, while introducing an average error of less than 1 hop. Thus, applications that can tolerate a small amount of location error gain the most from us-



(a) Success Ratio (b) Average Error (c) Avg. Query Latency

Fig. 8. Performance of global GetNearest queries.

ing MLDS. Furthermore, MLDS' hierarchical architecture enables efficient execution (low cost and latency) of GetNearest queries, especially when the density of mobile agents is high.

7. Related Work

MLDS is related to data-centric storage (DCS) systems like GHT [20], DIFS [7], DIMENSIONS [6] and DIM [11]. GHT hashes data by name to nodes in the network and provides no index for accessing the data. DIFS leverages on GHT and maintains a hierarchical index of histograms to support multi-range queries. DIM, on the other hand, uses a locality-preserving hash function that maps a multi-attribute event to a geographic zone. It divides the network into zones and maintains a zone tree to resolve multi-dimensional range queries. DIMENSIONS hashes sensor data to nodes in the network and maintains a multi-resolution hierarchical index that enables it to efficiently answer queries by drilling down to the appropriate nodes. The key differences between the above systems and MLDS are (1) the above systems store sensor data while MLDS is specifically tailored for storing location information of mobile entities, (2) MLDS supports a broad range of flexible spatial queries which cannot be supported efficiently by the above systems, and (3) MLDS builds a distributed directory over multiple sensor networks connected by an IP network, while the above systems are designed for a single sensor network.

TSAR [3] is another in-network storage architecture, which stores sensor data at a lower tier consisting of sensor nodes and stores only meta data at a higher tier consisting of a network of proxies. Unlike the above approaches, TSAR maintains a distributed index at the proxies. TSAR differs from MLDS in that it is not tailored for storing location information nor does it support spatial queries. Moreover, unlike MLDS, TSAR does not have an in-network tier that enables the system to take advantage of data locality while resolving queries. The comb-needle approach proposed in [13] also deals with in-network storage and retrieval of data and uses an adaptive push-pull technique to achieve this. In this approach, an event is stored along a short vertical path (resembling a needle) centered at the node that detects the event, while queries are disseminated across the sensor network along paths resembling a comb. However, when the query rate becomes higher than the event rate, the process is reversed and event information is forwarded along combs while queries are disseminated along needles. This method would incur a high communication cost for spatial queries like *GetNearest* and *GetNum* and hence cannot efficiently support the range of spatial queries supported by MLDS.

As mentioned earlier, MLDS' hierarchical directory structure is similar to that used by the Internet's Domain Name System (DNS) [15]. However, DNS does not support functionalities provided by MLDS like efficient maintenance of location information of mobile entities and spatial query services. Macro mobility solutions designed for the Internet like Mobile IP [18] mostly use DNS to locate a mobile entity and hence also differ from MLDS. MLDS' design is closer to micro mobility solutions like Hierarchical Mobile IP (HMIP) [8] and MMWN [19], used in cellular networks. However, unlike these systems, MLDS is (1) specifically designed for sensor networks with the goal of minimizing communication cost without considerable loss in data accuracy and (2) provides a rich set of multi-granular spatial queries.

MLDS is also related to the protocols presented in [2] and [22], which address in-network processing of K-Nearest Neighbor (KNN) queries and are based on the branch-and-bound technique [21] that is also used in MLDS. Another related service is EASE [23], which keeps track of mobile entities within a single sensor network through in-network storage and supports multi-precision queries that fetch the location of a specified mobile entity. MLDS differs from the above protocols in the following three important ways: (1) MLDS presents an architecture for storing location information in sensor networks that enables efficient computation of the nearest-neighbor and other multi-resolution spatial queries, (2) MLDS is the first location directory service that can keep track of mobile entities across multiple sensor networks, and (3) we implemented and integrated MLDS with a mobile agent middleware and present experimental results on a physical sensor network testbed. In contrast, the above protocols are only evaluated through simulations.

8. Conclusion

We present MLDS, a Multi-resolution Location Directory Service for tiered sensor networks that enables the realization of location based services that involve wide-area tracking of mobile entities. MLDS has several salient features: (1) it is the first system that maintains location information of mobile entities across sensor and IP networks, (2) it efficiently supports a range of multi-granular spatial queries that can span multiple sensor networks and (3) it has low communication cost. We integrated MLDS with Agimone, a mobile agent middleware for sensor and IP networks, and evaluated its performance on a testbed of several tmotes. The empirical results show that MLDS successfully keeps track of mobile agents across single and multiple sensor networks at significantly lower communication cost than a centralized approach. Most importantly, MLDS enables applications to achieve the desired tradeoff between accuracy and communication cost, which is particularly useful in resource constrained sensor networks.

Acknowledgments.

This work is funded by the NSF under the ITR grant CCR-0325529 and the NeTS-NOSS grant CNS-0520220.

References

[1] R. R. Brooks, P. Ramanathan, A. Sayeed, Distributed target tracking and classification in sensor networks, in: Proceedings of the IEEE, 2002.

[2] M. Demirbas, H. Ferhatosmanoglu, Peer-to-peer spatial queries in sensor networks, in: P2P'03.

[3] P. Desnoyers, D. Ganesan, P. Shenoy, Tsar: A two tier storage architecture using interval skip graphs, in: SenSys'05.

[4] C.-L. Fok, G.-C. Roman, G. Hackmann, A lightweight coordination middleware for mobile computing, in: Coordination'04.

[5] C.-L. Fok, G.-C. Roman, C. Lu, Rapid development and flexible deployment of adaptive wireless sensor network applications, in: ICDCS'05.

[6] D. Ganesan, D. Estrin, J. Heidemann, DIMENSIONS: Why do we need a new data handling architecture for sensor networks?, in: HotNets-I'02.

[7] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, S. Shenker, DIFS: A distributed index for features in sensor networks, in: SNPA'03.

[8] E. Gustafsson, A. Jonsson, C. E. Perkins, Mobile IPv4 Regional Registration, IETF Draft, draft-ietf-mobileip-reg-tunnel-09.txt (2004).

[9] G. Hackmann, C.-L. Fok, G.-C. Roman, C. Lu, Agimone: Middleware support for seamless integration of sensor and ip networks, in: DCOSS'06.

[10] T. He, C. Huang, B. M. Blum, J. A. Stankovic, T. F. Abdelzaher, Range-free localization and its impact on large scale sensor networks, ACM Transactions on Embedded Computing System (TECS) 4 (4).

[11] X. Li, Y. J. Kim, R. Govindan, W. Hong, Multi-dimensional range queries in sensor networks, in: SenSys'03.

[12] J. Liu, J. Reich, F. Zhao, Collaborative in-network processing for target tracking, Journal of Applied Signal Processing.

[13] X. Liu, Q. Huang, Y. Zhang, Combs, needles, haystacks: Balancing push and pull for discovery in large-scale sensor networks, in: SenSys'04.

[14] L. Luo, T. Abdelzaher, T. He, J. A. Stankovic, Envirosuite: An environmentally immersive programming framework for sensor networks, TECS.

[15] P. Mockapetris, K. J. Dunlap, Development of the domain name system, ACM SIGCOMM Computer Communication Review 8 (4) (1988) 123–133.

[16] D. Moore, J. Leonard, D. Rus, S. Teller, Robust distributed network localization with noisy range measurements, in: SenSys'04.

[17] S. Pattem, S. Poduri, B. Krishnamachari, Energy-quality tradeoffs for target tracking in wireless sensor networks, in: IPSN'03.

[18] C. E. Perkins, S. R. Alpert, B. Woolf, Mobile IP: Design Principles and Practices, Addison-Wesley Longman Publishing Co., Inc, 1997.

[19] R. Ramanathan, M. Steenstrup, Hierarchically-organized, multihop mobile wireless networks for quality-of-service support, Mobile Networks and Applications 3 (1) (1998) 101–119.

[20] S. Ratnasamy, B. Karp, L. Yin, F. Yu, GHT: A geographic hash table for data-centric storage, in: WSNA'02.

[21] N. Roussopoulos, S. Kelly, F. Vincent, Nearest neighbor queries, in: SIGMOD'95.

[22] J. Winter, Y. Xu, W.-C. Lee, Energy efficient processing of k nearest neighbor queries in location-aware sensor networks, in: Mobiquitous'05.

[23] J. Xu, X. Tang, W.-C. Lee, EASE: An energy-efficient in-network storage scheme for object tracking in sensor networks, in: SECON'05.

[24] W. Zhang, G. Cao, Optimizing tree reconfiguration for mobile target tracking in sensor networks, in: Infocom'04.

Chenyang Lu is an Assistant Professor of Computer Science and Engineering at Washington University in St. Louis. He received the B.S. degree from University of Science and Technology of China in 1995, the M.S. degree from Chinese Academy of Sciences in 1997, and the Ph.D. degree from University of Virginia in 2001, all in computer science. He is author and co-author of more than 70 refereed publications, and received an NSF CAREER Award (2005) and a Best Paper Award from International Conference on Distributed Computing in Sensor Systems (DCOSS'06). His research interests include real-time embedded systems and wireless sensor networks.

Gruia-Catalin Roman is the Harold B. and Adelaide G. Welge Professor of Computer Science at Washington University in Saint Louis. Roman has an established research career with numerous published papers in a multiplicity of computer science areas. His current research involves the study of formal models, algorithms, design methods, and middleware for mobile computing and sensor networks. He has been on the faculty of the Department of Computer Science at Washington University in Saint Louis since 1976. Roman is also a software engineering consultant. Roman served as associate editor for ACM TOSEM and was the general chair for ICSE 2005.

Chien-Liang Fok is a Ph.D. student in the Computer Science and Engineering Department at Washington University in St. Louis where he also received B.S. degrees in Computer Science and Computer Engineering. He is currently working in the area of sensor networks under the guidance of Dr. Chenyang Lu and Dr. Gruia-Catalin Roman. His primary research interests include flexible software systems, middleware, and virtual machines.

Sangeeta Bhattacharya is a Ph.D. student in the Computer Science and Engineering Department at Washington University in St. Louis. She received her B.E. degree and M.S. degree in Computer Science from Bangalore University, India and Texas A & M University, respectively. She is currently working on developing services to achieve application QoS guarantees in resource constrained wireless sensor networks, under the guidance of Dr. Chenyang Lu and Dr. Gruia-Catalin Roman.