

SCalable Object-tracking through Unattended Techniques (SCOUT) *

Satish Kumar

USC Information Sciences Institute
kkumar@isi.edu

Cengiz Alaettinoglu

Packet Design Inc.
cengiz@packetdesign.com

Deborah Estrin

UCLA and USC Information Sciences Institute
destrin@cs.ucla.edu

Abstract

A scalable object location service can enable users to search for various objects in an environment where many small, networked devices are attached to objects. We investigate two hierarchical, self-configuring or unattended approaches for an efficient object location service. Each approach has its advantages and disadvantages based on the anticipated load. The first approach, SCOUT-AGG, is based on aggregation of object names. The second approach, SCOUT-MAP, is based on indirection, where information about an object is stored at the locator sensor for the object. The relative efficiency of SCOUT-AGG and SCOUT-MAP can be characterized by the query to mobility update rate of the system. SCOUT-AGG performs better for low query to update rate but its performance deteriorates in general relative to SCOUT-MAP as the query to update rate increases. The rate of performance deterioration depends on query specificity (i.e., queries for a specific object or for any object of a particular type). SCOUT-MAP generally exhibits better load balancing than SCOUT-AGG for various scenarios. We support the above results through simple analytical modeling and simulation.

1. Introduction

Computer networks are rapidly becoming ubiquitous with recent advances in technology. Wireless technologies such as Bluetooth[4] enable the networking of small low-cost devices such as PDAs and domestic appliances. Networks of sensors with sensing, actuation, signal processing and wireless communications in the same module are rapidly becoming available for deployment [10, 6]. With such developments, it is possible to envision our homes and work-places equipped with hundreds of thousands of small, networked devices which can co-ordinate to perform various tasks.

We expect a key application in such an environment to

*This work was supported by DARPA under contract number DABT63-96-C-0054, as part of the VINT project. We are grateful to Ashish Goel and the anonymous reviewers for their comments.

be an *object location service*. Users may desire a service to locate various shared objects such as projectors and cameras or more common items such as coffee-mugs and keys. A key challenge is then the ability to locate a particular object in a scalable fashion among hundreds of thousands of monitored objects.

In this paper, we examine the object location problem by assuming an office environment where people and objects are tagged with unique pre-programmed IDs. Sensors that monitor the location of these tagged objects (through periodic RF interrogation signals) also communicate with each other to form a network. An user who wishes to find the location of some person in the building or a projector for his presentation simply queries this network for such information. These queries need to be efficiently directed to the appropriate sensors. Since the communication range of passive tags[3] can be small, an object tracking network in a building may require thousands of sensors. In such large sensor networks, techniques based on manual configuration of individual sensors may be impractical. Centralized solutions[1] do not scale well as the number and mobility of objects or the query rate increases. Hence self-configuring or *unattended* distributed techniques are required.

Relation to Service discovery: We build and improve upon some of the ideas proposed for service discovery in IP networks. One of our solutions for object tracking is similar to the Service Discovery Service (SDS)[18] proposal that uses an hierarchy of SDS servers and aggregation of service/object names. As we later describe in this paper, such aggregation based solutions have limitations for some scenarios. Another hierarchical solution, the Globe[13] location service uses a hierarchy of location servers but forwarding pointers are stored for each object from the root to the leaf location server monitoring the object thus causing high load (especially under mobility) and state requirements at higher level servers.

Most other service discovery techniques such as SLP[8], INS[21] and Jini[20] were designed for enterprise networks. These approaches focus on efficiently answering complex queries based on matching various attributes that makes

scaling to large networks with larger number of objects more difficult. However, a large network can be partitioned into *regions* that each run SLP, INS or Jini, thus allowing complex queries to be answered efficiently within those regions. A leader can then be chosen for each region (that stores a complete *view* of the particular region) to participate in the hierarchical self-organization and query direction techniques presented in this paper.

In this paper, we investigate the performance for two simple query types: locating an unique object (e.g., where is Joe?) and locating an object of a particular type (e.g., any projector). We do not address more complex queries such as “find me an available conference room that is closest to an available copy machine so that attendees can quickly make copies of some presentation documents”. It may be possible to resolve these queries by breaking down into the simpler queries we support.

We next describe possible approaches for a scalable object location service.

Approaches: Simple distributed solutions such as flooding queries or distributing information about all objects to all the sensors in the network do not scale well as the number of sensors or objects in the network increase. A hierarchical solution is needed to achieve better scaling.

We examine two hierarchical approaches[19] to tackle the scaling problem. One is to use an hierarchy along with some form of aggregation so that distant sensors have a less detailed view. The other is to use indirection where each object is mapped to a *home* or *locator* sensor and that sensor stores information about the object. Both schemes have advantages and disadvantages over the other based on the anticipated load. In this paper, we focus on studying the trade-offs associated with these approaches and identify the conditions under which one approach is better than the other.

We refer to the first aggregation-based approach as *SCOUT-AGG*. In this approach, sensors organize themselves into an hierarchy and higher level sensors summarize and distribute concise information about objects sensed by their child sensors. Remote sensors then use the summary information to direct queries towards sensors with more detailed information about a particular object till the relevant sensor is reached. Such an approach forms the basis of various hierarchical routing schemes for the Internet and was also proposed recently for service discovery in SDS[18].

However, aggregation based approaches suffer from inefficiencies when applied to the object tracking context. In this approach, a sensor forwards a query to all the relevant child sensors whose summaries indicate that the object may be in their sub-tree. Since, aggregation usually leads to loss of information, a parent sensor does not know for certain if the object is located in a particular child branch or not.

Hence, the query also needs to be forwarded up the hierarchy (till the root) to reach all other relevant branches to ensure a response. In addition, queries may need to be forwarded by a sensor to many child sensors if the object summaries of many child sensors indicate that they may have the object (can degenerate to flooding in the worst case)¹.

We refer to the second indirection-based approach as *SCOUT-MAP*. In this approach, sensors use a hash function to map an object name to a sensor address (similar to the name-resolution scheme in Landmark routing [16]) that becomes the locator sensor for the object. The address of the sensor monitoring the object and the object location is stored at the locator sensor for the object. A querier performs the same algorithmic mapping to the object name and derives the address of the locator sensor. The locator sensor is then contacted for the object location or for the address of the sensor monitoring the object (if this sensor needs to be contacted to obtain more information about the object such as its status).

Other indirection approaches such as DNS[15] and X.500[9] do not work well for dynamic data and are difficult to self-configure. Mobile IP[17] also uses a home agent to keep track of mobile hosts. However, hosts know the *home address* of the mobile host from DNS and use this address to contact the home agent and obtain the current location of the mobile host. In our context, we need a scalable mechanism to find the home agent for a particular object.

Approaches based on indirection too suffer from inefficiencies relative to aggregation based schemes when either the mobility of objects or network dynamics exceeds a certain threshold. As objects move in *SCOUT-MAP*, the locator sensor needs to be updated while in *SCOUT-AGG*, the advertised summaries at higher levels may not change each time the object moves. Excessive rate of network dynamics too can make indirection based schemes more inefficient since many objects may need to be re-mapped to different locator sensors after each topology change.

Summary of findings: The relative efficiency of the *SCOUT-MAP* and *SCOUT-AGG* schemes can be characterized by the ratio of the query rate to the mobility update rate of the system. At high query-to-update rates, *SCOUT-MAP* performs better than *SCOUT-AGG* since the overhead of mobility updates is offset by the efficiency in answering queries. At low query-to-update rates, the performance of *SCOUT-AGG* is better than *SCOUT-MAP* due to its more efficient mobility update procedure. At very low query-to-update rates, flooding is better than both *SCOUT-AGG* and *SCOUT-MAP* since the overhead of object mobility is min-

¹An optimization is to maintain query state at sensors so that queries are forwarded up the hierarchy only if the object is not available in the sensor's sub-tree. However, maintaining such per-query state limits the scaling of the system as the query rate grows.

imal for flooding (no mobility updates are required).

The query-to-update ratio at which SCOUT-MAP is better than SCOUT-AGG also depends on the type of queries. This cross-over ratio is lower for *specific* queries i.e., queries for unique objects (e.g., a person or a book), than for *non-specific* queries i.e., queries for any object of a certain type (e.g., any projector). A specific query is answered more efficiently in SCOUT-MAP by simply contacting the locator sensor for the object whereas the same query may need to travel to many branches of the hierarchy in SCOUT-AGG. In the case of non-specific queries, the overhead of SCOUT-AGG is smaller than that for a specific query since the probability of finding an object of a certain sub-type close to the querier sensor is higher.

The load balance across sensors at different hierarchical levels in SCOUT-AGG can be poor since many queries often need to go to higher levels to ensure a correct response. SCOUT-MAP uses hashing to uniformly distribute the locator sensor for objects throughout the network and thus the query processing load is more evenly distributed among the sensors.

Document Organization: We begin in Section 2 with a description of the SCOUT-AGG and SCOUT-MAP schemes. We compare the above two schemes through some simple analysis and simulation in Section 3 and conclude in Section 4.

2. Description of schemes

We first describe the key components of the SCOUT-AGG and SCOUT-MAP schemes assuming an underlying hierarchy of sensors that adapts to network dynamics. We later describe automatic hierarchy construction which is a common component for both the schemes in Section 2.3.

2.1. SCOUT-AGG scheme

We first describe a simple naming schema for objects and rules for aggregating object names. We then describe query processing rules followed by mechanisms to handle object mobility and network dynamics.

2.1.1. Naming and Aggregation We assume a simple attribute-value pair based naming schema for objects. All objects have an *object-name* attribute. The object-name attribute has an hierarchical value based on the defined object class hierarchy. For example, the *object-name* of a projector is *conf.equipment.projector.146* where 146 is an unique projector ID. The complete description of a projector under this naming schema could be *[object-name = conf.equipment.projector.146, location = (10, 20, 10), status = busy]*.

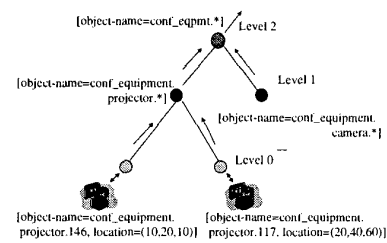


Figure 1. Aggregation in SCOUT-AGG

We use the defined object class hierarchy to perform aggregation. For example, the level 1 sensor in figure 1 that receives information about projectors from its child sensors with names *[object-name = conf.equipment.projector.146, location = (10, 20, 10)]* and *[object-name = conf.equipment.projector.117, location = (20, 40, 60)]* aggregates them to *[object-name = conf.equipment.projector.*]*. As can be observed, other attributes of the projector object advertised by the level 0 sensors are dropped by the level 1 parent sensor. Note that the level 1 sensor can advertise the full name of the projector if only one projector is reported by its child sensors.

The above aggregation is repeated at successively higher levels till the object-name consists of one of the root object types in the object class hierarchy. That is, at level 2, the aggregated name becomes *[object-name = conf.equipment.*]*. Higher level sensors cannot further aggregate this name since *conf.equipment* is a root object type.

2.1.2. Query processing A sensor on receiving a query from a user checks if the queried object is monitored locally. Otherwise, the sensor attaches its ID to the query and forwards the query to its parent. When the query travels up the hierarchy to a parent sensor that does not have any local information about the object, the parent attaches its ID to the query and forwards the query down the hierarchy to the child sensors whose aggregates cover the object name. For example, a search for *conf.equipment.projector.117* would be forwarded by a parent sensor to all the child sensors that advertise an aggregate object-name of *conf.equipment.projector.** or *conf.equipment.**. The parent sensor further adds its ID to the query and forwards the query up the hierarchy to ensure correctness. (since the parent cannot deduce from the aggregates whether the queried object is present in any of the branches from its child sensors). Thus, many queries travel to the root sensor to get forwarded along all the relevant branches of the hierarchy. When a query comes down the hierarchy and a sensor does not have local information

about the object, the sensor adds its ID to the query and forwards the query to its child sensors (if any) that advertised relevant aggregates.

The sensor that has the requested information, sends back a response to the querier following the reverse path of the recorded route in the query packet. Note that the route recorded in the query is on the order of $O(\text{number of hierarchical levels})$ which just increases logarithmically with the number of sensors in the network.

The querier sensor may get multiple responses for *non-specific* queries i.e., a query for any object of a certain sub-type (e.g., a query for any projector). A higher level sensor (level > 1) that receives a non-specific query may pessimistically forward the query to many of its child sensors following the advertised aggregates. Hence, many sensors that monitor an object of the sub-type may be reached that then generate responses back to the querier.

2.1.3. Handling mobility of objects When an object moves from one sensor to another, both the sensors generate an update message to their parent indicating the change. The parent sensors recompute their aggregates on receiving the updates from their child sensors. If the aggregate list changes, the parent sensors further send an update message to their parents. This process continues till a higher level sensor whose aggregate list remains unchanged is reached (which may be the root in the worst case).

The update message from the sensor that no longer sees the object is delayed for a short while so that the update message from the sensor that now sees the object may reach the higher level sensors first. This is to reduce the number of update messages triggered in the hierarchy (i.e., to prevent the case where an object is summarized as not being available up the hierarchy and then the object is summarized as being available shortly thereafter when it is monitored by an adjacent sensor).

2.1.4. Handling network dynamics When a higher level sensor loses one of its children or gains a new child, the higher level sensor recomputes its aggregates. If the aggregate list changes, an update message is sent to its parent sensor. This process repeats till the sensor whose aggregate list remains unchanged is reached. Whenever the parent of a sensor changes, the child sensor sends an update message with its aggregates to the new parent.

2.2. SCOUT-MAP scheme

We next describe the various components of the SCOUT-MAP scheme. We begin with a description of the hierarchical addressing and routing scheme (based on Landmark routing [16]) used in SCOUT-MAP followed by a description of the process by which objects are mapped to their lo-

cator sensors. We then describe query processing and rules to handle mobility of objects as well as network dynamics.

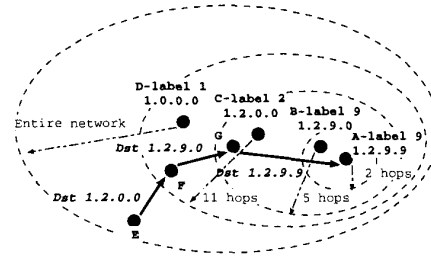


Figure 2. Query direction in SCOUT-MAP

2.2.1. Hierarchical addressing and routing We now describe the main features of the hierarchical addressing and routing component that is used by sensors to route messages among themselves. As before, we assume an underlying hierarchy of sensors with a single sensor at the root of the hierarchy. The above is illustrated in figure 2 where D is the root sensor, C is the child of D, B is the child of C and A is the child of B. For reasons of clarity, we do not show the remaining sensors in this network except for sensors E, F and G that are used for later descriptions.

Each sensor in the hierarchy is associated with a positive integer *label*. The root sensor assigns itself a label of 1 while at lower levels, parent sensors assign different integer labels to each of their child sensors. For example, the root sensor D assigns a label of 1 to itself and assigns label 2 to its child C while C assigns label 9 to its child B and B assigns label 9 to its child A. The address of a sensor at a particular level is defined to be the concatenation of its label with the labels of all its ancestors in the hierarchy. For example, sensor C concatenates its label 2 with the label of its ancestor D i.e., 1 to obtain 1.2 as its address. In this fashion, the addresses of sensors D, C, B and A are 1, 1.2, 1.2.9 and 1.2.9.9 respectively. To make all addresses be of same length, we simply pad the remaining fields of shorter addresses with 0's. For example, we represent the root sensor D's address as 1.0.0.0, C's address as 1.2.0.0 and B's address as 1.2.9.0.

Sensors at a particular level are associated with a *radius*. The radius specifies the number of physical hops that a sensor's advertisements will travel. A sensor's advertisements carries its hierarchical address, the sensor's level in the hierarchy and the hierarchical addresses of its children. Sensors at higher levels have larger radii than those at lower levels. In figure 2, A, B and C at levels 0, 1 and 2 have radii of 2, 5 and 11 hops respectively. Sensor D that is the root sensor has a radius of infinity i.e., its advertisements are flooded throughout the network. A sensor is said to be in the *vicinity* of another sensor if its distance to that sensor is less than

or equal to the radius of the higher level sensor. For example, all sensors within 2 hops of A are said to be in the vicinity of A and thus have routing information to reach A as well as information about the hierarchical level and the addresses of the children of A. Note that a parent and child sensor are always in each other's vicinities and thus have routing information to reach each other (e.g., D and C are in each other's vicinities).

2.2.2. Mapping object names to locator sensor addresses

As described before, sensors store location and other information about a particular object at the locator sensor for the object. We next describe the process by which the locator sensor for a particular object is reached through an example. We call the message with the object information to be stored at the locator sensor as the *map* message for the object. Suppose sensor E in figure 2 wishes to store information about object o1 (o1 is a unique object ID) at the locator sensor for o1.

- Since E is within the vicinity of the root sensor D, E knows the addresses of D's children. Suppose the root sensor D has 3 children with addresses 1.1.0.0, 1.2.0.0 and 1.3.0.0. E then algorithmically maps the object name o1 to the set of addresses of D's children i.e., {1.1.0.0, 1.2.0.0, 1.3.0.0} to obtain 1.2.0.0 as the initial locator sensor address for o1.
- Since E is not within the vicinity of sensor 1.2.0.0, E simply sends the map message towards the root sensor 1.0.0.0. This map message then reaches F which is in the vicinity of 1.2.0.0.
- F then algorithmically maps the object name, o1 to the set of addresses of the child sensors of sensor 1.2.0.0 (sensor C) and obtains 1.2.9.0.
- Since F is not within the vicinity of sensor 1.2.9.0, F sends the map message towards sensor 1.2.0.0 with 1.2.9.0 as the locator sensor address. This map message then reaches G in the vicinity of 1.2.9.0.
- G algorithmically maps o1 to the set of addresses of the child sensors of sensor 1.2.9.0 (sensor B) to obtain 1.2.9.9.
- Since G is not within the vicinity of sensor 1.2.9.9, G sends the map message towards sensor 1.2.9.0 with 1.2.9.9 as the locator sensor address. The above map message then reaches some sensor in the vicinity of 1.2.9.9 that then forwards the message to sensor 1.2.9.9 (sensor A).
- The locator sensor 1.2.9.9 stores information about o1 such as the address of sensor E that monitors o1 and the location coordinates of o1. The locator sensor then sends an ack message with its address to E.

- E stores the address of the locator sensor for the object carried in the ack message. This information is used to handle network dynamics (Section 2.2.5). E re-initiates the mapping process if no ack is received after a timeout delay. This delay is increased exponentially for each subsequent re-mapping if no response is received from the locator sensor.

Note that the same algorithmic mapping function is used by all the sensors in the network so that the same locator sensor is reached consistently by all the sensors for the same object o1. The object information stored at the locator sensors are soft-state and need to be periodically refreshed (possibly at a low frequency of once every 4-5 hours). Information about an object may be stored at the locator sensor for the object (e.g., projector) and another locator sensor for the object type (e.g., conference_equipment). Hence, the locator sensor for an object type contains information about all the objects of a certain type and can be used to answer non-specific queries. However, load balance could be impacted if many queries are for the same object type or many objects belong to the same type. This problem may be alleviated somewhat by the use of multi-level mapping described below.

Multi-level mapping can be done to improve efficiency of answering queries for local objects and reduce the mobility update overhead. For example, sensor E can first store information about o1 at a close-by locator sensor. E achieves this by fixing several labels of the local locator sensor address with its own labels in the map message for o1. For example, if E (with address 1.1.2.3) would like the local locator sensor to be in its own sub-tree rooted at 1.1.0.0, E sends the map message for o1 with 1.1.0.0 as the initial locator address. This message would follow the same steps as previously described to reach the local locator sensor for the object. The local locator sensor stores the object information, sends an acknowledgment to E and generates a regular map message to the *global* locator sensor (if the local locator sensor sees the object for the first time) following the procedure described at the beginning of this section.

2.2.3. Query processing To query for an object, a sensor simply contacts the locator sensor (the local locator sensor is contacted first when multi-level mapping is used) to obtain location information or the address of the sensor monitoring the object. The querier sensor can then contact the sensor monitoring the object if more information about the object is desired.

2.2.4. Handling mobility of objects A sensor, on seeing a new object, updates the object's location at the locator sensor for the object (the local locator sensor in the case of multi-level mapping). In the case of multi-level mapping, the local locator sensor generates a map message to the

global locator sensor if this object information is seen for the first time by the local locator sensor. Similarly, a sensor that no longer sees an object or an object of a particular type updates the local and global locator sensors appropriately.

2.2.5. Network dynamics Sensors may need to re-map objects after a topology change. We consider an approach where objects are periodically re-mapped at low frequency but appropriate re-map messages are also triggered by some topology change. Mainly, if a parent sensor re-assigns addresses to its child sensors, the parent sensor floods a *Gen-map* message throughout the network with its address. For example, when C (in figure 2) at level 2 with address 1.2.0.0 assigns new addresses to its child sensors (due to addition/deletion of a child sensor), C floods a *Gen-map* message with its address 1.2.0.0 throughout the network. Object-monitoring sensors, and local locator sensors in the case of multi-level mapping, then remap objects whose locator sensors fall in the range 1.2.0.0. These mechanisms do suffer from higher overhead under high network dynamics than schemes based on aggregation as we show in Section 3.2.4. We do not address methods to determine the optimal re-map frequency for different objects based on their mobility characteristics and the frequency of network dynamics in this paper.

2.3. Automatic Hierarchy construction

We now describe the process by which sensors automatically organize themselves into a hierarchy that adapts to network dynamics. Both SCOUT-AGG and SCOUT-MAP schemes use this hierarchy for scaling. Our automatic hierarchy construction algorithm is based on the Landmark Hierarchy [16]. The previously presented algorithms for Landmark hierarchy construction were however incomplete or had excessive convergence times. We consider a bottom-up hierarchy construction algorithm since top-down[7] methods can not easily build hierarchies that reflect topology parameters like node neighbourhood densities. Some of our presented techniques for hierarchy construction are an extension to those used in MASH[2]. Other types of hierarchies such as the area hierarchy[11, 14, 5] are more difficult to self-configure and previously proposed schemes do not guarantee that all nodes in the network will be associated with some cluster or do not completely specify the mechanisms to construct a multi-level hierarchy that adapts to network dynamics.

We next describe our algorithm for hierarchy construction. All sensors start off at the lowest level of 0 with a radius of r_0 hops. Each sensor then sends out periodic advertisements to other sensors within r_0 hops. The sensors then wait for a certain *wait time* that is proportional to their advertisement radius in order to allow advertisements from

other sensors to reach them. At the end of the wait period, a level 0 sensor starts a *promotion timer* if it has not yet chosen a parent. The promotion timer is inversely proportional to the number of other sensors from which level 0 advertisements were received. This would cause sensors located in relatively dense regions to have smaller timeout values.

When the promotion timer expires, a sensor promotes itself to level 1 and starts sending periodic level 1 advertisements within radius r_1 . In these advertisements, the newly promoted sensor lists its potential child sensors i.e., namely the level 0 sensors whose advertisements it previously received. Only the level 0 sensors that appear in this potential children list can choose the level 1 sensor to be their parent. The above ensures that parent-child relationships are established only between sensors that can see each other's advertisements and thus are able to communicate with each other. A level 0 sensor picks the closest potential parent to be its parent sensor.

After promotion to level 1, the level 1 sensors start a promotion timer for level 2 if they hear at least one other level 1 sensor. Otherwise, the hierarchy construction is completed. However, level 1 sensors do re-start the election process if they subsequently hear an advertisement from some other level 1 sensor. Note that, the radius, r_i of sensors at level i should be greater than $2 * r_{i-1}$ to ensure that level i sensors can see at least one other level i sensor if any exist.

The above process is performed recursively by sensors at each level to construct the multi-level hierarchy. A sensor re-starts the election process if it does not receive periodic advertisements from a parent sensor for a certain period. Similarly a parent sensor drops a sensor from its children and/or potential children list if it does not hear any advertisements from the sensor for a certain period.

A level i sensor may demote itself if it does not have any children or its potential children are a subset of some other level i sensor's potential children. If either of the above criteria is satisfied, the level i sensor demotes itself only if it sees another level i sensor that can be its parent after demotion. Discussion of the rules to prevent sensors from oscillating between different levels are however beyond the scope of this paper.

3. Comparison of SCOUT-AGG and SCOUT-MAP schemes

We first compare the overhead incurred by SCOUT-AGG and SCOUT-MAP through simple analysis. We then follow up the analysis with some simulation results.

3.1 Analysis

We present a simple model to analyze the relative message overhead of the SCOUT-AGG and SCOUT-MAP

schemes. Let C_a and C_m be the average number of hops that messages traverse to answer a query in SCOUT-AGG and SCOUT-MAP respectively. Let D_a and D_m be the average number of hops that mobility updates traverse in SCOUT-AGG and SCOUT-MAP respectively. If Q is the total number of queries generated per second and M is the total number of updates about objects (mobility and soft-state refresh updates) generated per second, the total message cost (per second) of SCOUT-AGG and SCOUT-MAP are $Q * C_a + M * D_a$ and $Q * C_m + M * D_m$, respectively.

The ratio of the message cost of SCOUT-MAP to SCOUT-AGG is then $R = \frac{Q * C_m + M * D_m}{Q * C_a + M * D_a}$.

The message cost of a query in SCOUT-MAP, C_m , is the path length from the querier sensor to the locator sensor to the sensor monitoring the object and back to the querier sensor. The message cost of a mobility update in SCOUT-MAP, D_m , is the path length from the sensor monitoring an object to the locator sensor for the object and back. We make a worst-case assumption that D_m is equal to C_m . We also make a best-case assumption of an unit message cost for mobility updates in SCOUT-AGG, $D_a = 1$ i.e., the aggregates stored at the parent sensor (at an average of 1 hop away) is unaffected by object motion and hence the parent need not communicate any more information up the hierarchy.

Then, the ratio of the message cost of SCOUT-MAP to SCOUT-AGG becomes $R = \frac{(Q/M+1)*C_m}{(Q/M)*C_a+1}$.

At very high query to mobility update rates i.e., $Q/M \gg 1$, the above ratio tends to $R = \frac{C_m}{C_a}$. In the common case, R may be less than 1 at higher Q/M due to the relatively higher message overhead to answer a query in SCOUT-AGG (due to false positives) than SCOUT-MAP.

At very low query-to-update rates ($Q/M \ll 1$), the ratio of message overheads, $R = C_m$. Hence, the performance of SCOUT-MAP can be significantly worse than SCOUT-AGG at low query-to-update rates.

We note that the performance of SCOUT-MAP could be improved if the possibly distant locator sensor is not updated for every object move. The above can be achieved by maintaining a forwarding pointer at a sensor that contains the address of the neighbouring sensor that an object has moved to. The locator sensor for an object is updated only when the length of the forwarding pointer is greater than (or more generally, some percentage of) the mobility update overhead, D_m for the object. In this case, the message overhead of modified SCOUT-MAP is $Q * C_m + (M/D_m) * D_m = Q * C_m + M$.

The ratio of the message overheads of modified SCOUT-MAP and SCOUT-AGG is then $R_{new} = \frac{(Q/M)*C_m+1}{(Q/M)*C_a+1}$.

At high query to mobility rate, above ratio tends to $R_{new} = \frac{C_m}{C_a}$ which is the same as before.

However, at low query-to-update rates, R_{new} tends to 1. Hence, the use of forwarding pointers seems to improve

performance of SCOUT-MAP close to SCOUT-AGG at low query-to-update rates while maintaining the same relative performance as before at high query-to-update rates. The benefits may be higher in practice. For example, if people move around only on their building floor, the length of the forwarding pointer may rarely become greater than D_m and hence very few updates may need to be generated to the locator sensor. A disadvantage to the use of forwarding pointers is the difficulty to maintain pointers under high network dynamics.

3.2. Simulation set-up

We compare the overhead incurred by the SCOUT-AGG and SCOUT-MAP schemes using the ns[12] network simulator. We mainly compare the bandwidth overhead incurred by the schemes but we also consider query loss rates for scenarios that involve sensor failures. Unless otherwise stated, our scenarios consist of connected topologies of 300 wireless sensors with a transmission range of 120 meters and 3000 objects randomly distributed in a rectangular region of 1275x1275 meters. Hence, each sensor monitors a random number of objects. Sensors self-organize themselves into an hierarchy and generate periodic refresh messages every 400 seconds (with a random jitter) for hierarchy maintenance. Each sensor then generates a mean of 10 queries per hour for randomly chosen objects. We simulate the system performance for 15000 seconds and each result is averaged over 10 simulations using different topologies and random number seeds.

For aggregation purposes, we create objects with 3-level hierarchical names in our simulations i.e., names are of the form *a.b.c* where *a* represents the object type, *b* represents the object sub-type and *c* is an unique id among all the objects of the same sub-type. We chose 10 types with an average of 10 sub-types per type for our simulations.

We assume that object names are 4 bytes and the IDs of sensors carried by the source route in SCOUT-AGG(Section 2.1.2) are each of 4 bytes. The names of aggregates (such as *a.b.** or *a.*.**) at higher level sensors in SCOUT-AGG are also 4 bytes. In SCOUT-MAP, we assume that hierarchical addresses are 8 bytes. All messages also carry a 4 byte time-stamp and sequence number that is used for loop prevention. We assume that responses are a fixed size of 50 bytes and all packets have a 20 byte header (same as IP header size). We do not model packet losses due to radio interference in our simulations.

In our simulations, queries are always directed by the schemes to the sensors monitoring the relevant object that then generate a response to the querier sensor (i.e., locator sensors in SCOUT-MAP do not directly respond to the querier sensor with information about the queried object). We perform two-level mapping of the object and the object

sub-type (e.g., a.b.c and a.b.*) in SCOUT-MAP by fixing the highest 3 levels of the initial locator sensor address to be the same as the object-monitoring sensor (i.e., a sensor that monitors an object stores information about the object at the local locator sensor for the object and the object sub-type that then store this information at the corresponding global locator sensors).

3.2.1. Performance for different types of queries Figure 3 shows the overhead of SCOUT-AGG and SCOUT-MAP as the percentage of specific queries is increased for a scenario where all the objects are static.

SCOUT-MAP performs better than SCOUT-AGG when all queries are specific. In SCOUT-MAP, specific queries are always answered by simply contacting the locator sensors for objects whereas the same query may have to travel to all the sensors that advertised the object's sub-type in SCOUT-AGG. For example, a query for a.b.c may travel to all the sensors that advertise a.b.* as an aggregate.

The performance of SCOUT-AGG is better than SCOUT-MAP when all queries are non-specific since a non-specific query can be answered more frequently by some nearby sensor than can a specific query. For example, a query for object a.b.* simply needs to travel to a branch that advertises an aggregate a.b.*. In SCOUT-MAP, at least the local locator sensor for the object sub-type still needs to be contacted to answer the non-specific query.

The overhead of SCOUT-AGG and SCOUT-MAP for specific queries is higher than the overhead for non-specific queries (much more so for SCOUT-AGG). Due to the very high overhead of SCOUT-AGG for specific queries, SCOUT-MAP out-performs SCOUT-AGG for most mixes of specific and non-specific queries.

Figure 4 shows the average number of queries and responses processed as a function of the level of the sensors in the hierarchy for specific queries. In general, SCOUT-MAP exhibits better load balancing properties than SCOUT-AGG (we saw similar results when objects were mobile and when all queries were non-specific) since SCOUT-MAP uses hashing to uniformly distribute the locator sensors for objects throughout the network. Hence, the number of queries and responses processed at a sensor is largely independent of the hierarchical level of the sensor. We do see a slight increase in load with the hierarchical level of the sensor since the higher level sensors are located in more central portions of the network and are thus in more of the paths taken by queries. In SCOUT-AGG, many queries need to travel to higher levels to reach other branches in order to ensure a correct response.

3.2.2. Scaling Figure 5 shows the variation in overhead as the number of sensors is increased. The 300, 500, 700 and 1000 sensor topologies were created by randomly distribut-

ing sensors in a rectangular grid of 1275x1275, 1550x1550, 1875x1875 and 2200x2200 meters respectively and sensors had an average degree of 8.1, 9.3, 8.9 and 9.6 respectively. The number of objects monitored per sensor is kept constant at 10 and the query rate also constant at 10 query/hour/sensor. The number of object types and sub-types were also kept constant. We also repeated the experiments with flooding as a baseline for comparison.

We find that both SCOUT-MAP and SCOUT-AGG scale well relative to flooding. We see a greater increase in overhead of SCOUT-AGG when all queries are specific (see figure 5) partly due to the greater number of objects that belong to the same sub-type distributed over many branches of the hierarchy. Similarly, this also accounts for the slightly slower rate of increase in overhead for SCOUT-AGG when all queries are non-specific (see figure 5) since the probability of locally finding an object of same sub-type is higher. As expected, flooding performs poorly as the number of sensors increases. Flooding performs worse for non-specific queries compared to specific queries since responses are generated by many sensors that monitor an object of a particular sub-type (an expanding ring search may reduce the extra responses).

3.2.3. Performance for various object mobility and query rates We simulate object mobility by moving an object from a sensor to one of its neighbouring sensors after a randomly selected mobility pause time. We classify objects into high, moderate and low mobility types. The mobility pause time of highly mobile objects is randomly selected from 10-300 seconds while that for moderately mobile objects is selected from 300-15000 seconds. The low mobility objects do not move for the duration of the simulation. The above parameters were chosen to somewhat mimic the real world that consists of very mobile objects (e.g., people), moderately mobile objects (e.g., portable projectors) and relatively static objects (e.g., printers).

In our simulations, we vary the percentage of highly mobile objects while keeping the percentage of moderately mobile objects at 20% with the rest of the objects being low mobility type. We first vary the percentage of highly mobile objects between 10, 30 and 50 while keeping the query rate fixed at 10 query/hour/sensor. We then fix the percentage of highly mobile objects at 10 and vary the query rate between 10, 20, 30, 40 and 50 queries/hour/sensor. We compare the performance of the schemes for these scenarios with a composite query-to-update metric. The query-to-update rate for a particular scenario is the ratio of the query rate to the percentage of highly mobile objects. We used the above definition since the total mobility updates per second was largely dominated by the updates from highly mobile objects.

SCOUT-MAP is more efficient than SCOUT-AGG in answering queries but can be more inefficient in processing

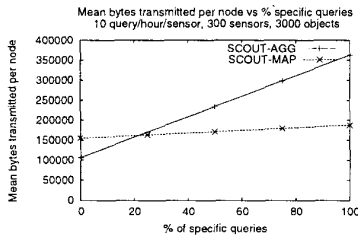


Figure 3. Overhead for various query mixes

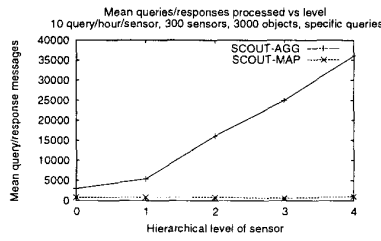


Figure 4. Load balancing properties

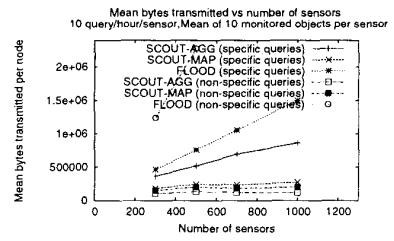


Figure 5. Scaling properties

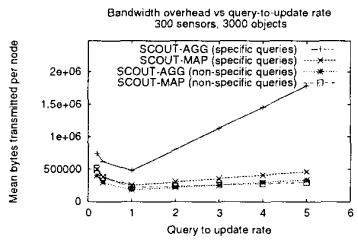


Figure 6. Overhead for various query-to-update rates

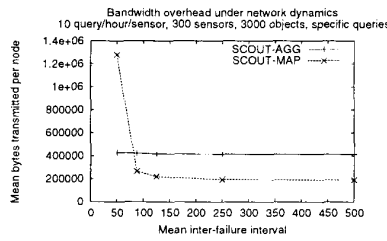


Figure 7. Bandwidth overhead under network dynamics

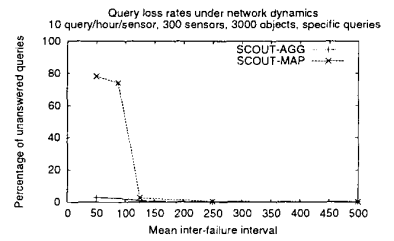


Figure 8. Loss rate under network dynamics

mobility updates. Mobility updates in SCOUT-AGG are more efficient since a mobility update need not travel further up the hierarchy once the update reaches a sensor that already monitors another object of the same sub-type or type. Mobility updates always need to travel at least to the local locator sensor of the mobile object in SCOUT-MAP.

Figure 6 shows the overhead of the schemes as the query to update rate increases. These results confirm that the relative overhead of SCOUT-MAP improves as the ratio of the query to mobility update rate is dominated by the query rate. The relative overhead of SCOUT-MAP to SCOUT-AGG also improves slower for non-specific queries compared to that for specific queries as the query-to-update rate increases (due to more efficient processing by SCOUT-AGG of non-specific queries than specific queries). We also found that this relative overhead improves slightly slower when objects of same type are clustered together due to lesser false positives from aggregation in this case (graph not included).

SCOUT-AGG can perform better when many queries are for nearby objects. A scoped search can then be performed

to efficiently reach the sensor monitoring the queried object. Since the efficiency to answer queries in SCOUT-AGG improves for this scenario, the query to update rate at which SCOUT-MAP performs better may increase compared to the case where there is no query locality. Further discussion of issues such as the impact of the object class hierarchy on SCOUT-AGG can be found in [19].

3.2.4. Performance under sensor dynamics Figures 7 and 8 compare the performance of the schemes under sensor dynamics. Starting from the beginning of the simulation, a randomly selected sensor is brought down once every *mean inter-failure interval*. The delay between two successive sensor failures is selected from 0 to 2 times the desired mean inter-failure interval. For example, a mean failure-interval of 500 seconds, is achieved by bringing a randomly selected sensor down after a delay uniformly distributed over [0,1000] seconds from the previous randomly selected sensor failure (starting from the beginning of the simulation). The down-time is uniformly distributed over [0,1000] seconds. Sensors re-organize the hierarchy when-

ever dynamics occur.

We plot the overhead of SCOUT-AGG and SCOUT-MAP with single level mapping as the inter-failure interval is increased for a scenario with 100% specific queries. The performance of SCOUT-MAP for multiple level mapping should be slightly worse than the shown results since sensors that monitor objects as well as the local locator sensors for these objects may need to re-map objects after a topology change. We notice that both the schemes perform well above the threshold of 125 seconds in our simulations. Before this threshold, the percentage of unanswered queries and the bandwidth overhead of SCOUT-MAP increases rapidly. This is due to the high overhead to re-map many objects in SCOUT-MAP after a topology change (especially for sensor failures at higher levels in the hierarchy) relative to the overhead incurred by the re-computation of aggregates in SCOUT-AGG.

4. Summary and Future Work

We investigated two hierarchical approaches for scalable, self-configuring object location. The first approach, SCOUT-AGG, based on aggregation of object names similar to SDS[18], performs well at lower query-to-update rates. The second approach, SCOUT-MAP, based on the name-resolution scheme in Landmark routing [16], performs better as the query-to-update rate increases. The query to update rate at which SCOUT-MAP becomes better than SCOUT-AGG is lower for specific queries compared to non-specific queries. SCOUT-AGG exhibits poor load balancing properties compared to SCOUT-MAP since many queries may need to go to higher levels in SCOUT-AGG to ensure a correct response while SCOUT-MAP uses hashing to distribute the load evenly among all the sensors. We show through simulation that both SCOUT-AGG and SCOUT-MAP scale well with an increase in the number of sensors. We also briefly presented an automatic hierarchy construction algorithm that adapts to sensor failures and is used by SCOUT-AGG and SCOUT-MAP for scaling.

In SCOUT-AGG, we do not require that all objects be searched using a single hierarchical name-space as presented in this paper. In practice, overlays of multiple hierarchies based on different name-spaces may be useful for efficient searches along various dimensions[18]. For example, a hierarchy and name-space based on geography may be useful for queries based on geographical proximity. Further analysis is the subject of future work.

Future work will investigate efficient resolution of more complex queries. We also intend to investigate the use of forwarding pointers in SCOUT-MAP and Bloom filters[18] for aggregation in SCOUT-AGG. Bloom filters can reduce the false positives due to aggregation but at the expense of possibly higher mobility update overhead.

References

- [1] A. Harter, A. Hopper, P. Steggle, A. Ward and P. Webster. The Anatomy of a Context-Aware Application. In *Proc. of the ACM/IEEE MobiCom*, Aug. 1999.
- [2] A. Rosenstein, J. Li and S.Y. Tong. MASH: The Multicasting Archie Server Hierarchy. *ACM Computer Communication Review*, 27(3), July 1997.
- [3] Automatic Identification Manufacturers Association. RFID Technology Information. <http://www.aimglobal.org/technologies/rfid/>.
- [4] Bluetooth Special Interest Group. <http://www.bluetooth.com>.
- [5] D. Coore, R. Nagpal and R. Weiss. Paradigms for Structure in an Amorphous Computer. Technical Report 1614, Massachusetts Institute of Technology Artificial Intelligence Laboratory, Oct. 1997.
- [6] D. Estrin, R. Govindan, J. Heidemann and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proc. of the ACM/IEEE MobiCom*, Aug. 1999.
- [7] David G. Thaler and China V. Ravishankar. Distributed top-down hierarchy construction. In *Proc. of the IEEE INFOCOM*, 1998.
- [8] J. V. E. Guttman, C. Perkins and M. Day. Service location protocol, version 2. *RFC-2608*, June 1999.
- [9] G. Neufeld. Descriptive names in X.500. In *Proc. of the ACM SIGCOMM, Austin, Texas*, Sept. 1989.
- [10] G. Pottie, W. Kaiser, L. Clare and H. Marcy. Wireless Integrated Network Sensors. Submitted for publication, 1998.
- [11] J. Hagouel. *Issues in Routing for Large and Dynamic Networks*. PhD thesis, Columbia University, May 1983.
- [12] Kevin Fall and Kannan Varadhan, editors. ns Notes and Documentation. The VINT Project. Available from <http://www-mash.cs.berkeley.edu/ns/>, Oct. 1998.
- [13] M. van Steen, F. Hauck, P. Homburg and A. Tanenbaum. Locating Objects in Wide-Area Systems. *IEEE Communications Magazine*, pages 104–109, Jan. 1998.
- [14] N. Shacham and J. Westcott. Future Directions in Packet Radio Architectures and Protocols. *Proc. of the IEEE*, 75(1):83–99, Jan. 1987.
- [15] P. Mockapetris. Domain names - concepts and facilities. *RFC-1034*, Nov. 1987.
- [16] Paul F. Tsuchiya. The Landmark hierarchy: A new hierarchy for routing in very large networks. In *Proc. of the ACM SIGCOMM*, 1988.
- [17] C. Perkins. Ip mobility support. *RFC-2002*, Oct. 1996.
- [18] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph and R. Katz. An Architecture for a Secure Service Discovery Service. In *Proc. of the ACM/IEEE MobiCom*, Aug. 1999.
- [19] S. Kumar, C. Alaettinoglu and D. Estrin. Scalable Object-tracking through Unattended Techniques (SCOUT). Technical Report USC CS TR00-738, University of Southern California, 2000.
- [20] Sun Microsystems. Jini technology white papers. <http://www.sun.com/jini/whitepapers>.
- [21] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan and J. Lilley. The design and implementation of an intentional naming system. In *Proc. of the 17th ACM SOSP*, Dec. 1999.