Disseminating Active Map Information to Mobile Hosts

Bill N. Schilit*
Computer Science Department
Columbia University
New York, NY 10027

Marvin M. Theimer
Xerox Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA 94304

Abstract

Mobile computing differs from desk-top computing because of the dynamic nature of system state: as users move, the sets of stationary and mobile objects they control and the types of information they wish to access change. Navigating a mobile environment can be aided by *active maps* that describe the location and characteristics of objects within some region as they change over time. We describe an *active map service* that keeps clients informed of changes in their environment. The primary issue driving our design is the question of scale: an active map service must be able to handle updates and queries over sufficiently large regions of space to satisfy clients' interests and must be able to handle peak loads that can occur when everyone in a region is moving around, for example, to attend a meeting. Our solution detects sets of clients that wish to receive the same active map information and then dynamically assigns multicast groups to them. To guarantee clients that they and their communication links will not be overloaded the active map provides guaranteed limits on bandwidth use.

1 Introduction

In recent years an extended form of computing has emerged, called *mobile distributed computing*, in which users interact with many different mobile and stationary computers over the course of the day. Computation no longer occurs at a single location in a single environment, but rather spans a multitude of situations covering the office, meeting room, home, airport, hotel, plane, etc.

A significant aspect of this emerging mode of computing is the frequently changing execution environment to which users and long-running applications are exposed. As users move about, the sets of mobile and stationary objects they interact with may change, producing a highly dynamic execution environment in which location is important. Location information is necessary for users and applications that want to query and interact with nearby devices

[1]This work was supported by Xerox. Portions were also paid for by ARPA under contract DABT63-91-C-0027.

*Currently visiting Xerox Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto, CA 94304

and services. Such information also allows stationary clients to track moving objects. In general, location information enables software to adapt according to its location of use, the collection of nearby people and objects, as well as the changes to those objects over time. We use the term *context-aware computing* to describe software exhibiting these general capabilities. Articles about this general topic include [4], [11], [12], [13], and [14].

This paper focuses on the communication issues we have encountered in disseminating location-based information to interested clients. We introduce the notion of an *active map service* (AMS) that publishes location information for objects in a region. We shall use the term *located-object* to refer to the entities managed by an active map service. Located-objects are descriptions of anything that has an associated physical location. Examples of located-objects include persons, printers, terminals, and workstations, as well as location-dependent services, such as information agents associated with particular parts of a store or building.

A key issue that we have faced in building an active map service is that of scale. A variety of issues must be dealt with to avoid overloading the AMS, its clients, and the communication facilities that join them:

- Large meetings of mobile users may cause considerable traffic because context-aware clients at the meeting as well as remote applications may need to be informed of the frequent occupant changes. High message volume may also occur in high traffic locations such as building lobbies.
- Low bandwidth links (usually wireless) limit the amount of information that can be sent to some clients.
- Mobile hosts have significantly stricter resource budgets and hence cannot do as much work as other hosts. For example, concern for power conservation may limit the activity that battery-powered hosts are willing to perform on an ongoing basis.

Unfortunately simply partitioning the AMS into many servers, each managing a small area, is only a limited help because clients frequently desire to know about information covering an entire region whose size is independent of what might be most suitable for the AMS. This is illustrated by the common ways we have observed clients using location information:

- Many users may wish to track a particular located-object as it moves around a region. Examples include tracking a co-worker you wish to talk to and tracking the office coffee cart in order to be made aware when either is nearby.
- More often users wish to track located-objects with a specified set of attributes in a particular region. An example is tracking all members of a workgroup.
- Users may want to find the located-object nearest to a specified location (usually their own) that meets a specified set of constraints. Examples include finding the nearest printer and finding the nearest system administrator.
- Finally, users frequently wish to monitor activity at a particular location. A typical example is keeping track of available display devices at one's current location.

Of the four examples, the first three represent regional queries that will require the AMS to cover a region whose size is likely to be that of a building or small campus, irrespective of how the AMS' implementation does so.

An architecture that addresses these issues is the primary focus of this paper. The worst-case usage scenario that our architecture will be designed to handle is the "meeting scenario" in which several hundred people, all using context-aware applications, converge on an auditorium over a period of several minutes. To illustrate some of the issues involved, consider how location information about this scenario might be disseminated to the people involved: the simplest—a naive unicast approach—would result in n updates going to each of n clients, which quickly gets out of hand. Waiting for quiescence and batching the updates would reduce the message count to n, but would sacrifice timeliness. Using broadcast or multicast [3] for the updates would reduce the aggregate message traffic even further, but would likely *increase* the traffic seen by some clients, since not all clients are interested in exactly the same information. This increase may be problematic for clients residing on small hosts (as PDAs) or that are connected via low-bandwidth links.

Each of the approaches exhibits different trade-offs of server, network, and client loads. To solve the problems of controlling and balancing component loads we use a combination of three techniques: detection of sets of clients that all wish to receive the same information; dynamic assignment of client sets to multicast groups; and update suppression to guarantee clients that their communication links will not be overloaded. The next section describes context-aware computing, active maps, and our system model in greater detail. Section 3 presents our design for disseminating active map information, Section 4 describes the performance of a testbed implementation we have done, Section 5 describes related work, and Section 6 summarizes our work and presents our conclusions.

2 Context-Aware Computing and Active Maps

2.1 Context-Aware Computing

Context-aware computing is the ability of a mobile user's applications to discover and react to changes in the environment they are situated in. In our system mobile users run software that is constantly monitoring, or *subscribing* to information about the world around them. This monitoring is done for a variety of reasons, including:

- To provide a display of "interesting" located-objects.
- To keep a record of located-objects and persons one has encountered, for use by applications such as "activity-based information retrieval" which uses the context at the time the data was stored to assist in retrieval [7].
- To detect location-specific information, for example electronic messages left for the user or for public perusal.
- To keep a look out for nearby devices that can be used opportunistically by applications, such as additional display terminals in a room.

- To detect nearby people, located-objects, or services that are relevant to reminders or actions set to be triggered by their presence.

Further descriptions of a variety of location-based applications can be found in [4] and [12]. We describe one particularly illustrative application here in more detail: the **locations** program. This program displays located-objects for a region as a textual list or as a map marked with descriptive icons. "Interesting located-objects" usually means persons who have chosen to make their present location publicly available (within, say, the confines of their office building or organization), although any set of objects for which location information is available—such as printers and copiers—can also be displayed[1]. When a mobile object, such as a person, moves, the associated information is updated on the **locations** display.

One important aspect of the **locations** program is that in crowded meeting rooms or large regions filtering must be done in order to obtain a manageable amount of information to display. Thus, the information that clients are actually interested in is a subset of all the location information that might be available to them. Where and how filtering is done will turn out to be an important issue for controlling the behavior of our system.

Users of our system employ and interact with a heterogeneous and changing set of hosts. In addition to using stationary hosts and I/O devices, they may also carry wireless PDAs and notebook computers. Consequently, context-aware applications must be able to run on both small mobile hosts as well as the more traditional (and more powerful) stationary ones. Furthermore, they must be prepared to deal with a variety of different network media, from relatively slow wireless links to very high-speed LAN links.

Mobile users also expect to be able to monitor regions other than the one immediately surrounding them and expect to be monitorable themselves by remote parties. For example, a person visiting a remote site might wish to have their present location be known to an agent process running back at their home site so that interested parties can determine where they are. Similarly, that person might wish to monitor various activities occurring at their home site, such as the presence or absence of a colleague whom they wish to call by phone.

2.2 Active Maps

The AMS represents located-objects as tuples of key-value attribute pairs. Any kind of information that a client wishes to make publicly available can be stored with a located-object registration; however, every located-object must include a location attribute that describes its current physical location.

An active map consists of a hierarchy of locations with a containment relation; for example, rooms are contained in buildings and buildings are contained in a region. Figure 1. shows an

[1]The privacy issues surrounding who may gain access to someone else's location information are beyond the scope of this paper and are addressed in another paper[12]. Our work is based on the observation that many people are willing to make at least a limited version of location information available to co-workers within the confines of their workplace.

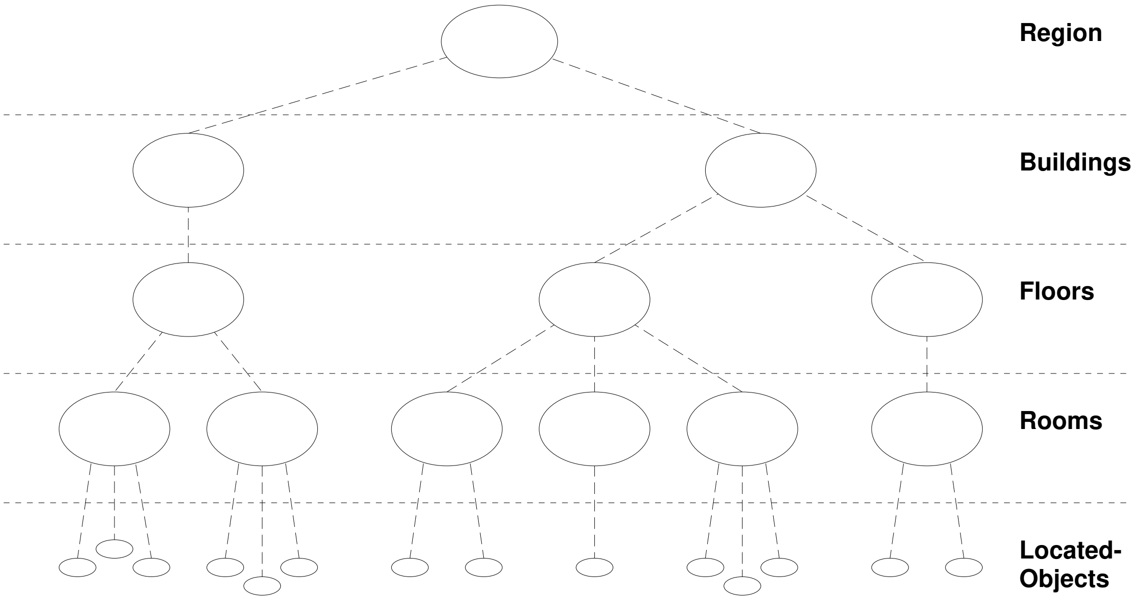

Figure 1: Example Active Map Containment Hierarchy.

example hierarchy. An AMS *region* is the set of locations maintained by a single active map server. The detailed structure (and depth) of any region's location containment hierarchy is region-specific.

The focus of our work so far has been on providing useful location information for AMS regions that cover localized, administrative entities, such as buildings or small campuses. This is because we believe the located-objects most useful for context-aware computing are close at hand, either co-located or requiring a short time to get to. Context-aware applications are more likely to monitor the contents of the room and building they are situated in than the contents of a building in the next town.

Because of the emphasis on proximate information, we do not address the question of how to manage information over larger regions, such as cities or countries. Instead, we assume that clients will accept the burden of interacting with any and all AMS regions they are interested in. An important consequence of this approach is that we must be able to scale the implementation of an AMS service to cover regions the size of buildings and small campuses so that "regional" queries can be implemented in an efficient fashion.

Clients of the active map service "publish" information about objects at a particular location and/or submit queries to obtain information about other published located-objects. Queries are matched against the attribute keys and values of located-objects stored in the AMS. Standing queries, called "subscriptions," can also be specified; the AMS will send information as it changes over time to subscribing clients.

Receiving updates in response to subscriptions is the common way clients interact with the AMS. To characterize this form of interaction more precisely, we define the AMS as a set of "publications": $P = \langle p_1, p_2, \ldots, p_m \rangle$. This is the set of located-objects for which the AMS maintains information. Any client, i, can specify interest in a dynamically changing subset of P's publications by submitting a subscription query, S_i. The set of all such subscription queries is denoted by S. All members of S are run against P whenever the state of any member of P changes. We assume that at any given time, t, at most one member of P will change its state[2]. If a member of P changes at time t then the set of subscription queries matching the object is denoted the *update query set*, $u_q(t)$. The associated set of clients who should receive notification of the change is denoted the *update client set*, $u_c(t)$.

To illustrate how the AMS works we describe two examples. In the first, mobile users continually monitor their current locations for various reasons, such as those outlined in the previous section. In our system the personal information for each user is managed by a *user agent* process[3]. Mobile users employ the user agents to monitor the user's whereabouts by various means and publish that information in the AMS. The user agent for a user U, who leaves location $l1$ and enters location $l2$ might perform the following AMS operations:

- cancel publication of U's presence at $l1$,
- cancel subscription that notifies of changes to any objects located at $l1$,
- publish U's presence at $l2$,
- subscribe to be notified of changes to any objects located at $l2$. This subscription returns the current set of objects at $l2$ as its return value.

If any other clients of the AMS were monitoring either location $l1$ or location $l2$ then these clients would be notified of user U's movement.

The second example is the implementation of the **locations** program, described earlier. This program issues subscriptions to the AMS in order to be notified of changes to relevant located-objects anywhere in a specified region. The region might be the entire AMS region or a smaller sub-region if the AMS' location hierarchy contains smaller internal regions (such as the floors of a building). To obtain information about only people the program would restrict its subscription query to match located-objects that have, for example, the key-value attribute pair `(key = "type", value = "person")`.

2.3 System Model

The context-aware applications, active map service, and examples we have described require several things from the underlying system infrastructure. In particular, we assume that suitable location sensing facilities are available to track the locations of mobile objects and that suitable databases are available that describe the locations of stationary objects of interest. The system deployed in our laboratory uses several different technologies to detect the locations of mobile objects (mostly people):

- Active badges [13], are attached to mobile objects and monitored by sensors embedded in the ceilings of rooms and corridors.

[2]This assumption holds in practice because the AMS receives information about changing objects via RPC and processes each RPC in turn.

[3]See [12] for more information on how user agents fit into our system design.

- Input activity at keyboards and mice is monitored to detect the presence of logged-in users in front of stationary workstations.
- Nano-cell-based, diffuse infrared communications [10, 2], is used to communicate with PDAs and provide room-sized cell location information.
- Nano-cell-based, radio communications [6], is used to communicate with portable notebook computers and provide roughly room-sized cell location information.

Location information for each person is synthesized from these various sources by the user agent for that person. A person's user agent may then publish location information to the AMS, subject to any privacy constraints the user has set.

In addition to the availability of location-sensing facilities, we also assume that AMS clients can be reached by packet-based communications, such as IP. This requires that applications on mobile hosts are able to use a form of "mobile IP" communications protocol [5], or employ a "mobile host-agent architecture" [10]. These approaches deliver messages to mobile hosts in different ways. For mobile IP, message delivery is a network function, whereas the host-agent architecture uses a well known user process as an intermediary to track, forward, and possibly filter packets in an appropriate manner.

In addition to assuming that packets can be routed to mobile hosts we also require efficient multicast communications. Although currently there is no standard combined mobile-multicast communications protocol, one can imagine extending the current mobile IP protocols to support multicast [1]. Throughout most of this paper we will assume that mobile-multicast is supported throughout the system. In Section 3.6 we discuss the implications of relaxing this assumption to allow the use of unicast-only wireless links.

3 Disseminating Active Map Information

3.1 Basic Unicast Design

The simplest implementation of an active map service involves using reliable unicast communications, such as remote procedure call (RPC), between the AMS and its clients. Most of the time and for most clients RPC is a satisfactory communications paradigm to use. The reason is that, for the most part, there are not that many objects moving around and subscribers of the AMS are either directly connected to high-speed networks or are not sharing their slow-speed communication link with many others. There are not many objects moving around because people are the primary movable "located-object" in the system and most of the time they are stationary. Furthermore, most of the time people are ensconced in separate offices or cubicles and hence any wirelessly connected computers they have are likely to occupy different nano-cells in isolation. Of course, there are significant numbers of workplaces for which these assumptions of mostly stationary, mostly separated, people do not hold. Similarly, not all wireless technologies employ room-sized cells for their communications.

Brief periods of increased load can be handled by simply letting the unicast design queue

AMS update messages for later transmission and by batching together multiple updates. Two benefits are gained this way: (1) the packet overhead of an update message can be amortized over more update items, and (2) if the interval between updates spans several sightings of the same moving object then fewer update items can be sent by only including the latest sighting for the object.

The disadvantages of this approach are that subscribers obtain their information in a less timely fashion and that they obtain less accurate information about the activity around them. This is a problem, for example, because a client may find out too late or not at all that someone the client wanted to know about has passed nearby. Discarding updates also requires some additional message processing by the AMS. In any case, this approach to controlling overload works only if the overload lasts for a brief period of time.

3.2 A Reliable Multicast Design

Much of the AMS load generated during overload situations is due to sending the same update message, over and over again, to many different subscribers. This duplication can be eliminated by employing multicast to distribute the update message to all interested subscribers at the same time.

We shall refer to the data structure needed inside the AMS to implement reliable communications to a particular multicast group as a *multicast channel*. The AMS multicast channels implement a standard negative acknowledgment protocol. To support this, each multicast message contains a sequence number. Receivers check to make sure they see each sequence number in turn. If a missing message is discovered, then the receiver sends a "heal" request specifying the missing sequence number. The sender then retransmits the requested message in a heal response. In order to bound the time it takes receivers to detect a missed message, whenever there is no normal message traffic for longer than the "synchronization interval," a "synchronization" message is sent. Under high load situations waiting for synchronization messages to detect misses rarely occurs since data messages perform the same function.

3.2.1 Using a Single Multicast Channel

The AMS sends out asynchronous notifications, or callback messages, that contain information about the changing context. Using a single multicast channel for all callback messages minimizes the AMS load. The information about a located-object's change-of-state can be sent once instead of once for each client whose subscription query matches the event. For network segments having multiple clients the load would also be minimized, since identical messages occur only once.

However, this reduction in server and network load comes in exchange for additional load placed on all clients and on some communication links. This is because all clients—and the communication links joining them to the AMS—now receive all updates the AMS sends out. For clients that are power conscious and/or attached to slow communications links (e.g. a PDA attached to a 19200 bps infrared link) this solution is unacceptable.

3.2.2 Multiple Multicast Channels

Instead of indiscriminately sending update information to all clients one can use multiple multicast channels in order to send out information to selective groups of clients. This requires that the AMS be able to figure out how to assign some or all updates to one or more multicast channels to be used for dissemination.

We define the general *multicast-channel assignment problem* as follows. Recall the definition of the AMS in Section as a set of publications, P, a set of subscriptions, S, and update client sets, u_c. Let $G = \langle g_1, g_2, \ldots, g_n \rangle$ be a collection of multicast channels. We desire to define a mapping, $\mathcal{M}_c(t) : u_c(t) \rightarrow G$, of $u_c(t)$ onto G, such that if the clients specified by $u_c(t)$ listen to the multicast channels specified by $\mathcal{M}_c(t)$ at time t and the AMS sends out the update information intended for $u_c(t)$ to the multicast channels specified by $\mathcal{M}_c(t)$ at time t then the following will hold:

- Every subscription client i will receive the update information it has specified with its query, S_i, and
- The overall "system overhead" of disseminating update information will be minimized.

Unfortunately, when considered in its full generality, there are a variety of system overheads to consider when choosing $\mathcal{M}_c(t)$. Furthermore, these overheads are borne by different parts of the system and are only partly comparable to each other:

- The AMS bears the cost of computing $\mathcal{M}_c(t)$, sending updates about changes to P, and disseminating $\mathcal{M}_c(t)$.
- Clients bear the cost of tracking changes to $\mathcal{M}_c(t)$, as well as filtering unwanted updates.
- Duplicate update messages put additional load on communication links.
- Notification time is affected by the use of extra messages. If the AMS sends multiple messages for the same update information then that will introduce a delay for all but the first group of recipients. This is a cost borne by the clients.

Given these different cost metrics it is difficult to define what minimum system overhead means. Worse yet, even if we restrict the problem by choosing a particular cost metric, it will likely not be easily solvable in general. For example, we might choose to minimize server and communication costs by requiring that every update be sent using exactly one message. That is, $\mathcal{M}_c(t)$ would be restricted so that every update client set, $u_c(t)$, is mapped to exactly one member of G. In this case minimizing system overhead would mean choosing $\mathcal{M}_c(t)$ such that the message filtering overhead that client subscribers experience due to receiving unwanted update messages is minimized. Unfortunately, this problem contains a solution space that is exponential in the number of publications, subscribers, and multicast channels involved.

3.2.3 Detecting Multiply-Subscribed Queries

Because the general multicast-channel assignment problem is too difficult we must resort to a special-case approach for solving a limited form of the problem. Based on our experience with a collection of location-aware applications, we first note that the AMS load is determined largely by regional or object-specific queries that are issued by large numbers of clients as well as by queries monitoring locations with frequently changing contents, for example, meeting rooms and hallways. In contrast to these "hot spots" of activity, monitoring a single located-object as it moves from one place to another in a human time-frame produces only light message traffic. Our second observation is that, in most cases, many clients will specify the same or very similar subscription queries for a particular meeting or for their regional and object-specific subscription queries.

The AMS can exploit these situations by recognizing when multiple clients are specifying the same subscription query and employing a multicast channel to service the update traffic for that query. This approach imposes no additional filtering overhead on clients and reduces server and communication system load to the extent that clients can be encouraged to employ the same subscription queries. In particular, if many clients specify the same query for their subscription to a particular location then meetings can be handled efficiently because the AMS can assign a single multicast channel to that query and can use it to send a single update message to all the clients who submitted that query. Similarly, if many clients specify the same regional subscription query then the AMS can service those subscribers via a single multicast channel that is dedicated to them.

A more detailed description of how this approach works is given next. First we observe that we require a slightly different mapping than the one we have already defined. Specifically, we define $\mathcal{M}_q(t) : u_q(t) \rightarrow G$, which maps update query sets to multicast channels (instead of the associated update client sets). This is so that we can "collapse" clients of multiply-subscribed queries together. We define $\mathcal{M}_q(t)$ by defining an equality relationship for subscription queries, such as string equality for the (ASCII) source representation of queries, and then defining $\mathcal{M}_q$ such that at all times, t, equal subscriptions map to the same member of G and unequal subscriptions map to different members of G. The size of G is assumed to be sufficiently large to allow this.

The AMS computes $u_q(t)$ by maintaining a count of the number of current subscribers who have submitted any given query and assigns multicast channels to those with more than some pre-specified minimum number of subscribers, q. In particular, the AMS maintains a record for each different query corresponding to a currently submitted client subscription[4]. This record contains a count field, a multicast channel identifier, and a list of all clients that have submitted this query. A unique unicast callback address is also maintained for each client. The count corresponds to the number of clients who are currently interested in that particular subscription query. If the count for a query is equal to or above q then the multicast channel identifier is used to send update information to all the relevant clients. If the count is less than q then update information is disseminated using the list of client callback addresses attached to the record. When the AMS receives a new subscription request that contains the q-th instance of a query it assigns a multicast channel to the

[4]Clients may withdraw subscriptions and subscriptions are also garbage collected when their submitting clients are detected to have gone away.

query, records the channel's identifier, and notifies all clients subscribing to the query to listen to the new multicast channel for update information. It does so using the list of unicast callback addresses attached to the query record.

Detecting multiply-subscribed queries will only work well if clients actually specify the same queries. To encourage this we can define a "standard" set of query templates that are parameterizable by location as well as by a standard set of located-object attribute types. When two clients substitute the same parameters, for example because they are at the same location, then identical queries will result. This provides a fairly flexible set of queries that can still be easily recognized and mapped to appropriate multicast update channels by the AMS.

Clients can be further encouraged to use standard queries by having applications offer them as easily selected options in their user interfaces. For example, the **locations** program can offer appropriate regional queries as menu options for determining which located-objects to display information about. Standard located-object types might include `printer` and `person` and common filtering criteria might include `color` vs. `black-and-white` for printers and various region-specific organization and work group names for people.

3.2.4 Detecting Recurring Update Query Sets

The previous section described how updates to clients subscribing to the same queries can be disseminated with a common multicast channel. This section presents the idea that when different queries result in updates to the same set of clients, they can also share a multicast channel. For this to happen, we must recognize recurring update query sets, $u_q(t)$; that is, sets of non-identical subscription queries that repeat across a number of update events. This will occur in situations where different groups of clients want different, but possibly overlapping, "cuts" of a large amount of location information that is available for a particular location or region. As an example, consider a large multi-organization conference that is taking place. Many members of each organization may want to see only the location information pertaining to their own organization and, perhaps, a few "sister" organizations, rather than information about all attendees of the conference. This will be especially true for clients sitting on small hosts connected via slow communications links. Regional queries can be similarly specialized.

A side benefit of recognizing recurring query sets is that it allows the AMS to use a simpler form of query equality for recognizing multiply-subscribed individual queries. Two queries whose source form does not match, but which specify the same logical query, will always appear together at the level of query sets. For example a query `(key = "type", value = "person")` and a second query having `(key = "type", value = "person")` *and* `(key = "department", value = "legal")` are not identical but match the same located-objects when all people at the query location are from the "legal" department.

To detect recurring update query sets the AMS must keep a history of the update query sets that it has computed in the past. When a new update event occurs (at time t), the AMS computes its update query set, $u_q(t)$, and then searches its history of update query sets to see if the same query set has occurred before. If it has occurred several times before then it is a good candidate for having a multicast channel assigned to it so that updates to that set can be handled with a single update message. More precisely, the AMS keeps a list, H, of maximum size m, of previously computed query sets. Each list element contains the representation for a previously computed query set, a count of the number of times the AMS has encountered this particular query set, and a multicast channel identifier. Just as with individual query records, if the count for a query set is equal to or above the pre-specified value, q, then the multicast channel identifier will designate a multicast channel that can be used to disseminate update information. If the count is less than q then the multicast channel identifier will be `nil`.

When no multicast channel has been assigned to an update query set, update information must be disseminated separately to the subscribers of each query in the query set. This is done as follows: for each query in the query set send out update information using either the multicast channel or the list of unicast addresses that is associated with that query. We shall refer to this means of update as the *multiple-messages-per-query algorithm* in the rest of this section.

The exact steps the AMS performs for an update event occurring at time t are given below:

1. Compute $u_q(t)$.

2. Search for an exact match of $u_q(t)$ in the history list H. An exact match occurs when each query in $u_q(t)$ is equal to exactly one query in a candidate query set, H_i. Matching can be done efficiently by using a hash scheme to reduce the number of potentially equal query sets in H to one or a few candidates. Equality matching of query sets is done using representations that have been sorted into a canonical order.

3. If no exact match is found then $u_q(t)$ is added to the front of H with a count of 1 and a `nil` multicast channel identifier. If this causes the size of H to exceed m then the least recently used query set is discarded from the list. We maintain a doubly linked list of query sets in which an item is moved to the front of the list on each use. This makes the query sets sorted in the order of their last use, and the item at the rear of the list is the one we discard. Update information is sent to clients using the multiple-messages-per-query algorithm.

4. If an exact match is found—suppose it is element H_i—then the following steps are performed:

   a) Increment the count of H_i.

   b) If H_i's multicast channel identifier is not `nil` then use the indicated multicast channel to send out update information.

   c) If H_i's multicast channel identifier is `nil` and the count field is less than q then send out update information using the multiple-messages-per-query algorithm.

   d) If H_i's multicast identifier is `nil` and the count field is exactly q then perform the following steps:

      – Assign an unused multicast channel to the query set and assign the channel's identifier to the multicast channel field of the query set's record.

- For each query in the query set send out both the update information as well as the identifier of the newly assigned multicast group using the multiple-messages-per-query algorithm. Clients are expected to henceforth listen to the designated multicast channel for all future update information[5].

Given the ability to detect both multiply-subscribed queries and recurring update query sets one might wonder how important each scheme is to load reduction. We observe that in any system that is large enough to have a diverse clientele most of the multicast traffic is likely to go over channels belonging to the recurring query set scheme. However, tracking multiply-subscribed queries allows the query set scheme to avoid sending large numbers of unicast update messages when a query set does not end up using a multicast channel for update dissemination. Indeed, if only one scheme could be used then tracking multiply-subscribed queries would be preferable to tracking recurring update query sets because of this.

3.3 A Bandwidth-Limited Approach

For clients sitting on low bandwidth communication links or for hosts with limited computational resources available for "ancillary" applications (such as the **locations** program) the question of how to limit AMS update traffic is extremely important. If a client waits until a communication channel is flooded with updates before taking any action, then it will be difficult to send a message adjusting to a "smaller" subscription. Unfortunately, it is not easy to infer the amount of traffic that any given AMS subscription query will generate since circumstances can dynamically change.

The AMS can service such clients best by allowing them to specify their desired bandwidth limits as part of their subscriptions. The AMS guarantees never to send more traffic to a client than that client's bandwidth limit specifies, even if it means not sending all the update information that the client's subscription query has matched. In order to alert clients when they are missing update information because of bandwidth limitations we must add an additional count field to update messages that indicates how many updates are still pending. Clients receiving an update message with a non-zero count field know that they have received the latest available location information for any object that is included in the update message, but they may be unaware of the latest changes to any object *not* mentioned in the update message. It is up to each client to decide how to deal with partial update information.

The effect of bandwidth-limited subscription on our multicast update schemes is as follows: instead of using a single multicast channel for each multiply-subscribed query or recurring update query set we now use $b + 1$ multicast channels, where b is the number of different bandwidth limits that have been specified for the relevant individual query or any of the queries in the relevant recurring query set. Each of the $b + 1$ multicast channels is used to

[5]Note that because we use reliable multicast, clients are guaranteed to find out about changes in the set of multicast channels they should listen to, even if they miss the relevant multicast message. However, care must be taken to ensure that when a client detects a missed multicast message that contained such information the client asks for all relevant messages that might have been sent out already via the new multicast channel.

send a different "band" of update traffic for a query or query set. For example, if bandwidth limits b_1 and b_2 have been specified for a query then we employ three multicast channels, m_1, m_2, and m_3. The first b_1 bytes of update traffic in any given second will be sent via m_1. If any traffic is still available for sending during that second then $b_2 - b_1$ bytes of it are sent via m_2. Any remaining data is sent via m_3. Clients desiring bandwidth limit b_1 are told to listen only to m_1. Clients desiring limit b_2 are told to listen to both m_1 and m_2. Clients with no bandwidth limit are told to listen to all three multicast channels.

By using different multicast channels in the fashion described, the AMS only has to send out an update message once instead of b times. The price paid for this approach is that clients must be told more often of new multicast addresses to listen to; in particular, whenever a new client shows up with a different bandwidth limit than has already been seen for their subscription query. Since information about all but the first multicast channel to assign to a query or query set can be disseminated using an already assigned multicast channel, this overhead is not really a problem.

3.4 Availability

Our design relies on a centralized active map server, implying that the AMS will stop functioning whenever the server crashes. This is not a problem in practice for two reasons:

- The active map server is host-independent and hence can be run on any available server machine. A distributed watch-dog facility can monitor and restart the server if it fails.
- A well-known multicast channel can be used to allow a newly restarting server to obtain current location information for all located-objects currently in its region. All clients of the AMS are expected to be listening to this channel address and will re-register their location and re-submit their subscriptions with the AMS whenever they receive a Server-Restart message.

The result of this approach is that the active map server does not stay down for very long, assuming that the system makes two or more machines available as potential server hosts.

We have not addressed the issue of partitioning an AMS region. One can imagine AMS clients starting up new instances of the server in their partition when they notice its absence and having servers watch for each other (e.g. using the well-known multicast channel mentioned above) and perform re-integration whenever partitions heal. However, this represents future work.

One might be tempted to avoid the availability issues of a centralized service altogether by trying to implement the AMS as a distributed communications protocol for exchanging information between changing location-based objects and interested clients. Located-objects could multicast their update information and clients could multicast queries to find out what they need to know about the current state of some part of the system. Unfortunately such an approach will, in general, impose more filtering load on the system's located-object managers and clients because there is no centralized "traffic router" that has extensive knowledge about who is interested (and more importantly, *not* interested) in what. Given the need to avoid overloading weak clients and slow communication links we consider the disadvantages of a totally decentralized design to outweigh its advantages, especially given the ability of our centralized server to quickly recover from failure.

3.5 Local-Scope Versus Internet Multicast Groups

The assumption that applications can use large numbers of multicast groups is currently not reasonable in a general Internet setting: dynamic global address assignment is difficult and significant numbers of Internet-wide multicast groups would overload the routing tables. Fortunately, most AMS clients are likely to reside on hosts local to the AMS' region. Therefore by limiting the range of messages sent over a multicast group—for example, to a campus—the same group can be "recycled" and used from one campus to the next.

Currently we limit the range of IP multicast datagrams by setting their time-to-live (TTL) parameter: multicast datagrams are forwarded from one subnet to another only if their TTL is above a certain threshold. A small drawback with this approach is that AMS regions must correspond to the TTL regions defined by the multicast network routers that will be used (or that the network routers have their TTL region definitions be appropriately tuned).

Because some clients may reside on remote hosts an AMS cannot rely solely on local-scope multicast groups to disseminate heavily subscribed updates. In many cases, simply unicasting the relevant updates to the one or two remote clients interested in them is sufficient (in addition to disseminating them via local-scope multicasts).

If each AMS obtains one or a few Internet multicast group addresses for itself then these can be used to alleviate load in cases where a significant number of remote clients exist and are interested in heavily subscribed updates. This works well to the extent that fewer Internet multicast groups are needed than the AMS has obtained for itself. As the demand for additional Internet multicast groups increases, the AMS can control its own load by reusing each Internet multicast group for multiple multiply-subscribed queries or recurring query sets and forcing remote clients to assume additional filtering overhead. Note that the use of Internet multicast groups can be kept completely separate from the AMS' handling of local clients via local multicast groups (as long as clients' "remoteness" can be readily determined). Hence the problems of dealing with remote clients need not affect the local system.

3.6 Dealing With Unicast Wireless Links

Many wireless communication technologies—such as current cellular telephone systems—do not offer multicast support. An agent-based implementation for routing packets to mobile hosts also makes the provision of multicast difficult since agents communicate with their mobile hosts in an essentially unicast-based fashion.

A hybrid scheme can help somewhat for systems that do not support multicast semantics across their wireless communications links: the AMS still employs multicast to reduce its own load; however, communications are directed at agent processes on LAN-connected hosts that convert the multicast traffic to unicast and forward it to the actual client applications running on mobile hosts. Unfortunately this does not alleviate the load problems that occur when the same update message gets sent separately to many clients residing in the same slow, wireless communications cell.

This problem can be partly alleviated by moving the relevant client applications partly or wholly to stationary hosts that are directly connected to a multicast-capable network. If a suitable local host is available then this may not be difficult and may even simplify matters: complicated filtering needed to cull a small, manageable amount of information from the update traffic generated during overload situations can be done on a faster stationary machine instead of on a slower, resource and power-conscious, portable machine. The final information results, which will presumably be much smaller in size than the original update traffic, can then be sent to application "front-ends" that reside on users' portable machines.

In some cases however, a suitable local host may not be available. While a mobile user is in his "home" AMS region he will likely have access to a trusted workstation or server machine on which to safely run his applications. If the user visits a remote region he may not have access to a suitable local machine there and would be forced to run his applications remotely on a machine in his home region. Depending on the applications and networks involved, the additional delays this would impose may or may not be an issue. In general, we conclude that, while unicast wireless links can be partially dealt with—especially when clients have access to local trusted hosts—the availability of multicast on wireless links is far preferable as a basis for an AMS system.

4 An Implementation

In our laboratory we currently have two implementations of the AMS running: an older version of the system has been in production use by about 40 people for several years and a new version exists for use in a simulated testbed environment. The latter was built in order to explore the ideas presented in this paper and all the results presented in this section were obtained from it.

The testbed system we built was designed to tell us two different things: how well the active map server we built works and what kinds of loads might occur under various different user movement scenarios. In particular, we were interested in characterizing the behavior of the system under "normal" load circumstances as well as when a meeting of many people took place.

Figure 2 shows some micro-benchmark numbers for our AMS testbed. These measurements were made on a Sun SparcStation-2 client communicating using Sun RPC over an Ethernet to a SparcStation-2 server machine running the active map server. The line marked "query match" shows the time required to match a simple query against a collection of AMS objects with a single object being successfully matched and returned to the query client. The slope shows that each additional object matched contributed about 66 microseconds to the service time. The line marked "query return" shows the cost of returning objects whose size is

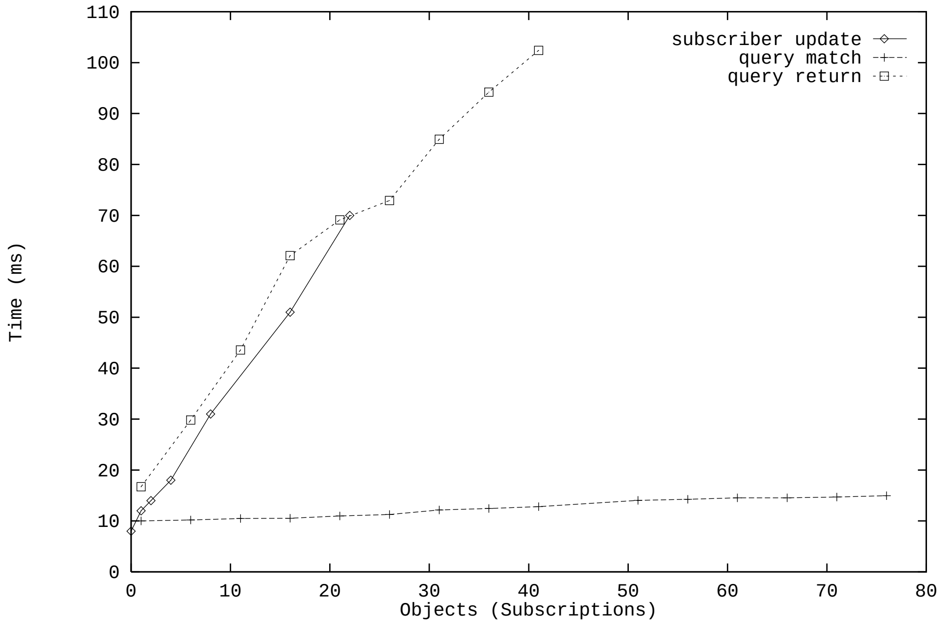

Figure 2: Micro-benchmark numbers for an active map server running on a SUN Sparcstation-2.

around 400-500 bytes to the client[6]. The slope of the line indicates about 2.5 milliseconds are needed for each additional object returned. The line marked "subscriber update" shows how AMS service time for unicast subscriber updates depends on the number of messages that must be sent to disseminate information about an update event.

To see how our system might behave under different usage scenarios we built an artificial workload generator, `SimMob`, for producing sets of sightings that can then be played back to our active map server. The workload generator is a discrete event simulation of user movement consisting of multiple "people" moving from vertex to vertex on a graph representation for a building. The building we simulated consists of multiple copies of the floor plan for our laboratory. Each person has a transition probability matrix (called a "mode") and a rate of transition. Activities, such as a meeting, that occurs during the workload simulation can change the mode and rate for one or a group of people. Our workload generator knows about paths and distances: the main simulation loop chooses a destination based on the users mode and takes this destination and expands it to the sequence of steps (i.e., sightings) along the shortest path. These steps are then emitted at times based on the step's distance and a global travel-rate. Once a move is complete, the workload generator chooses a Poisson-distributed random sleep time (based on move-rate) for the person and leaves that person stationary for that length of time before repeating the entire process again.

[6]The reasons why located-objects in our system are so large are twofold: descriptions consist of several attributes and we use the relatively inefficient Sun XDR facilities to encode object descriptions. Although one could significantly reduce the size of object descriptions it would not qualitatively change the results we obtain. Indeed, by using a large object size we obtain conservative bounds for how well our system should perform; using smaller object sizes would only improve things.

Using `SimMob` we are able to build artificial sightings of hundreds of users in hundreds of locations. To test the AMS we designed two types of workloads, "meeting" and "normal." The meeting workload has people start in individual locations and at some point converge on a single location over a specified period of time, typically about 3 to 5 minutes. The normal workload has users spend most of the time in their offices and every once in a while venture out to another location.

To see how large a normal work community might be handled by the AMS we ran a normal workload against a unicast version of the AMS with 1000 users in the system. As input to the workload generator we used an 80% probability of staying in the office location which resulted in 12% of the community being mobile at any given time, on average. This translated into mobile users being sighted an average of every 8.9 seconds, resulting in a computed average of 600 moves per minute overall. The generated workload was executed on 18 SparcStation hosts each with between 55-56 client processes, along with hosts for the server and a monitor program. Running the workload produced updates at the expected rate, with encounters with other people occurring on about 40% of the moves. No regional queries were included in this workload; users monitored only their own locations. For the unicast design we observed an average delay of 23ms in receiving updates—about twice the unloaded case of 11ms—implying that the AMS should easily be able to handle in excess of a thousand users under "normal work" conditions.

To confirm the fact that things like meetings will quickly overload a unicast design we also ran several meeting workloads against our unicast version of the AMS. Figure 3 shows our results, indicating that meetings with more than about 50 participants started to produce excessive move-response times. In contrast, a single multicast channel version of the AMS seems able to support meetings of more than 500 participants.

The overall performance of a version of the AMS that incorporates the multiple multicast schemes described in this paper is determined primarily by the average number of update messages that the server ends up sending out per update event. That is, the cost of managing the data structures needed for these schemes is small compared to the cost of actually sending update messages out on the network[7]. When users employ standard queries the way that client applications try to encourage them to, the average number of messages that the AMS must send out for an update event is kept quite low. Indeed, for most update events, a single message suffices to update all regional subscriptions. The same is true for subscriptions to meeting locations. Unfortunately, only production use of our system by real users who have had a chance to write new applications will tell us how many non-standard subscription queries will occur in practice.

Figure 4 illustrates how the maximum size of meetings handled by the AMS server depends on the average number of messages that must be sent out per update event. The values were obtained by disabling the use of multicast channels for identical clients sets and generating artificial meeting workloads with a different numbers of distinct subscription queries. In this graph, the maximum meeting size is the point at which the AMS is receiving updates quicker than they can be disseminated. The graph shows the potential benefit of merging

[7]If update query set sizes were to start numbering in the many tens to hundreds then the cost of sorting them into canonical order and storing them would start to be significant. However, our whole design is predicated on the assumption that that many different queries will not match any given update event.

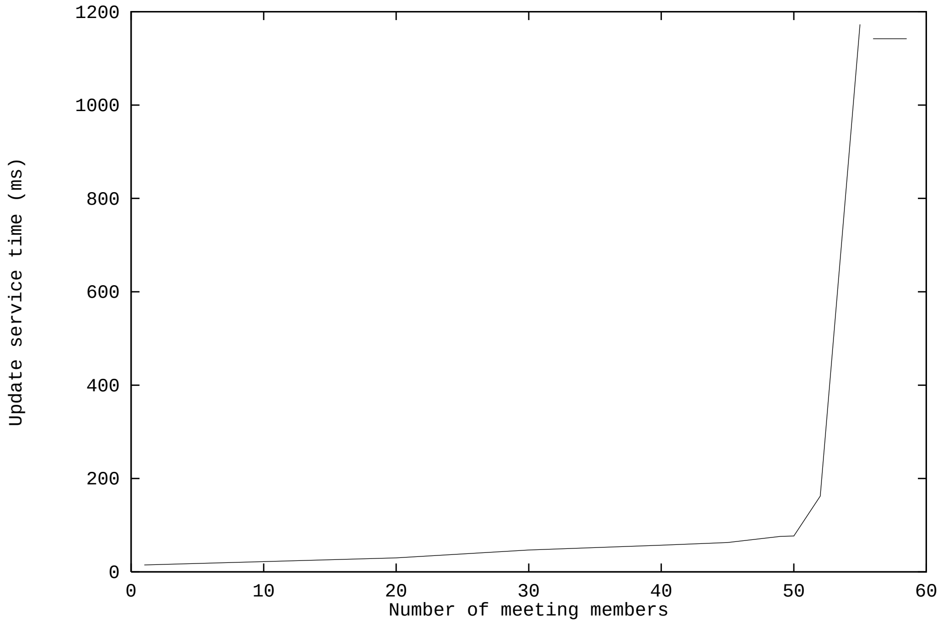

Figure 3: Average update time for clients monitoring a meeting as a function of meeting size.

client update sets into multicast channels over the use of query matching alone.

5 Related Work

As mentioned at the beginning of the paper, there have been a number of articles on the topic of location-based systems, including [4], [13], [11], [12], and [14]. [4] is perhaps the broadest and most recent overview available on the topic. The distinguishing feature of the work presented here is that it is the first that addresses the issues of scaling and overload conditions for systems covering "medium-sized" regions, such as buildings and small campuses.

Our work can take advantage of several lower-level communications technologies that have been, or are being, developed by others. These include IP-multicast[3], mobile IP[5], and mobile IP multicast[1].

A number of other systems also use a publish/subscribe style of information dissemination, most notably [9] and [8]. However, all these systems employ only broadcast or at most a few multicast groups for their implementation, resulting in extensive client-side filtering of messages. In contrast, our approach aggressively employs large numbers of multicast groups in order to keep client filtering overhead and slow communication link loads to a minimum.

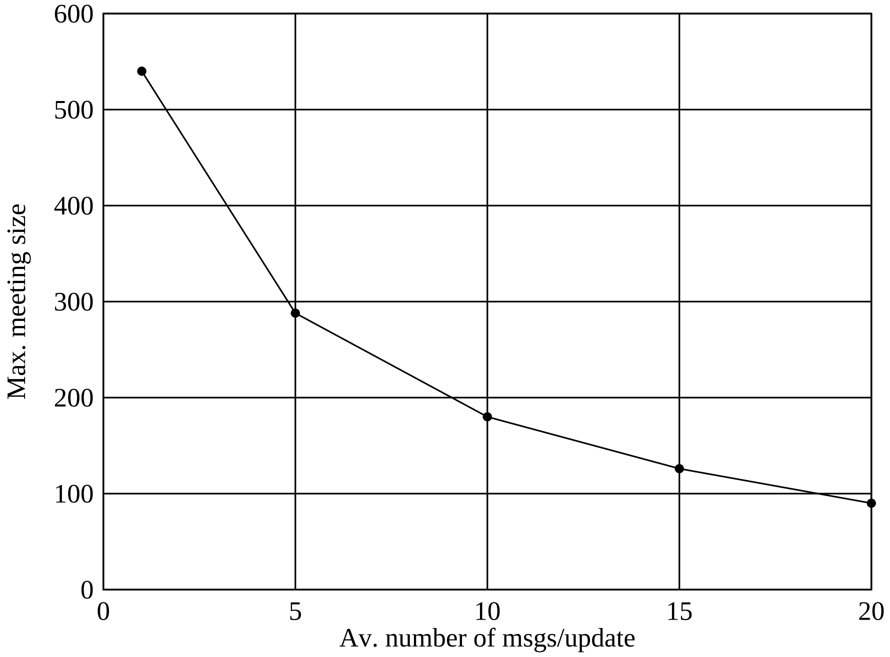

Figure 4: Maximum size of meeting that can be handled as a function of the average number of messages that must be sent out per meeting update event.

6 Summary and Conclusions

This paper has described an active map service that supports context-aware computing by providing clients with information about located-objects and how those objects change over time. We have focused on the communication issues of disseminating information from an active map server to its clients. In particular, we have addressed how to deal with various overload situations that can occur. Simple unicast callbacks to interested clients work well enough if only a few located-objects are moving at any given time and only a few clients wish to know about any given move. However, if many people are moving about in the same region and many clients are interested in their motion, then the AMS may experience overload due to the quadratic nature of the communications involved. This overload affects both the server as well as any slow communications links being used.

Generated workloads illustrate the extent of the potential overload problem: while our system seems capable of handling in excess of a thousand users under "normal" low-load circumstances (exemplified by people mostly sitting in their offices), meetings of more than about 50 people manage to overwhelm it if they are handled only by means of unicast communications. In contrast, when the AMS employs multiple multicast groups it is able to support meetings of more than 500 participants.

Our approach to this problem has been to exploit multicast techniques to avoid requiring the AMS to repeatedly send out the same update information message to different receivers. In order to avoid losing update information we use reliable multicast channels, which require clients to monitor sequence numbers in each multicast message and request retransmissions whenever they detect a missed sequence number.

To avoid increased client filtering overhead and added overhead on slow communications links we use multiple multicast channels so that AMS clients see only the subset of AMS updates they are interested in. Assigning multicast channels to suitable subsets of the update traffic in order to minimize system load is, in general, a very hard problem. The special-case solution we have pursued is to recognize subscription queries that have been submitted by multiple clients and sets of subscription queries that match multiple update events. This approach is based on two observations about AMS load: it is generated mostly by regional queries that are submitted by large numbers of clients and by update traffic related to a relatively small number of "hot spot" locations or objects.

To encourage the use of the same queries by many clients we have structured our applications to offer standard queries as easily invoked menu options. Structuring applications in this fashion can sharply increase the likelihood that many clients will employ the same query. The ability to tailor both the AMS and its clients is perhaps the key factor that enables our design to succeed. In essence we ensure that our load reduction schemes will work well by making sure that most client applications generate exactly the kinds of queries the schemes are designed to handle. If AMS applications were designed without keeping this in mind they might well not exhibit the regular patterns that the AMS depends on for controlling system load.

While tailoring applications to behave properly is key to our design, we feel that it is important to offer as much flexibility to clients as possible within the confines of our design. For example, because the recognition scheme used by the AMS dynamically recognizes multiply-subscribed queries, applications can change the set of standard queries they employ without having to change or even notify the AMS itself. We also do not outlaw "non-standard" queries in our design.

Another important feature of our design is its ability to strictly limit the update traffic that clients will be exposed to if they desire such a limit. By allowing clients to explicitly specify bandwidth limits for the AMS update traffic, we avoid being "blocked out" on a slow communications link from other more important application traffic and from inadvertently consuming all of a power-conscious host's resources.

Although we pursued a centralized design for the AMS we do not feel this causes an availability problem since the active map server is host-independent and hence can be restarted on any available server host. Restarting a server can be done fairly quickly by using a well-known multicast group address to solicit location and subscription information from all clients in the system. As a consequence, we have not felt the need to explore more complicated and expensive fully decentralized designs. However, future work remains to be done on dealing with partitions of an AMS region. We hope to explore an extension of our centralized design that relies on dynamically creating servers for each partition and then "merging" servers when partitions heal.

Because most clients of the AMS are expected to be local to the region it covers, the AMS uses local-scope multicast groups to reach local clients, thereby avoiding the difficulties of using large numbers of Internet multicast groups that must be globally allocated and managed; unicast is used to reach remote clients. We do not yet have experience with significant numbers of remote clients. If remote clients remain rare then use of a small pre-allocated set of Internet multicast group addresses per AMS region may suffice for dealing with them more efficiently than unicast would. However, if activities such as telecommuting make them common then our design could become problematic if AMS regions cannot obtain large numbers of easily managed Internet multicast group addresses for their private use.

Finally, we observe that our approach to disseminating information relies on the availability of multicast throughout a system, including its wireless communication links. While multicast can be added in a straight-forward manner to the experimental infrared and radio facilities that we have built in our lab, it is not readily available as part of the current commercial cell phone infrastructure and is not yet a part of any mobile IP protocol proposal. While we have described an agent-based approach that enables a system with only unicast-based wireless communications to take partial advantage of our multicast solutions, we believe that a widespread deployment of active map facilities—or facilities like them—will require the widespread introduction of wireless and mobile multicast facilities.

Acknowledgments

We thank our colleagues with whom we have discussed many of the ideas in this paper, especially Doug Terry and Steve Deering, who provided valuable insights and feedback to us in the early phases of the design. Conversations with Lixia Zhang helped us understand techniques for bandwidth limiting. A number of people worked on the location systems in our lab, including Roy Want, Mike Spreitzer, Karin Petersen, David Nichols and Phil James. The editors and referees provided plenty of valuable comments. Finally, we appreciate Mark Weiser's leadership in pursuit of the Ubiquitous Computing vision, under whose umbrella this work was pursued.

References

[1] Arup Acharya and B. R. Badrinath. Delivering multicast messages in networks with mobile hosts. In *In the 13th Intl. Conf. on Distributed Computing Systems*, pages 292–299, May 1993.

[2] Norman Adams, Rich Gold, Bill N. Schilit, Michael Tso, and Roy Want. An infrared network for mobile computers. In *Proceedings USENIX Symposium on Mobile & Location-independent Computing*, pages 41–52. USENIX Association, August 1993.

[3] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[4] Andy Harter and Andy Hopper. A distributed location system for the active office. *IEEE Network*, pages 62–70, January/February 1994.

[5] John Ioannidis, Daniel Duchamp, and Gerald Q. Maguire Jr. IP-based protocols for mobile internetworking. In *Proc. SIGCOMM '91*, pages 235–245. ACM, September 1991.

[6] Christopher A. Kantarjiev, Alan Demers, Ron Frederick, Robert T. Krivacic, and Mark Weiser. Experiences with X in a wireless environment. In *Proceedings USENIX Symposium on Mobile & Location-Independent Computing*, pages 117–128. USENIX Association, August 1993.

[7] Michael G. Lamming and William M. Newman. Activity-based information retrieval: Technology in support of personal memory. In F.H. Vogt, editor, *Personal Computers and Intelligent Systems.*, volume A-14 of *IFIP 12th World Congress. Proceedings of Information Processing 92*, pages 68–81. IFIP, Elsevier Science Publishers (North-Holland), 1992.

[8] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The information bus – an architecture for extensible distributed systems. In *Proceedings of the Fourteenth ACM Symposium on Operating System Principles*, pages 58–68, Asheville, NC, Dec 1993. SIGOPS, ACM.

[9] Steven P. Reiss. Connecting tools using message passing in the field environment. *IEEE Software*, 7(4):57–66, July 1990.

[10] Bill N. Schilit, Norman Adams, Rich Gold, Michael Tso, and Roy Want. The PARCTAB mobile computing system. In *Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, pages 34–39. IEEE, October 1993.

[11] Bill N. Schilit, Marvin M. Theimer, and Brent B. Welch. Customizing mobile application. In *Proceedings USENIX Symposium on Mobile & Location-Independent Computing*, pages 129–138. USENIX Association, August 1993.

[12] Mike Spreitzer and Marvin Theimer. Providing location information in a ubiquitous computing environment. In *Proceedings of the Fourteenth ACM Symposium on Operating System Principles*, pages 270–283, Asheville, NC, Dec 1993. SIGOPS, ACM.

[13] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, Jan 1992.

[14] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, September 1991.