A Multicast User Directory Service for Synchronous Rendezvous

Eve M. Schooler
Computer Science Department
California Institute of Technology
Pasadena, CA 91125

August 26, 1996
Technical Report CS-TR-96-18

In partial fulfillment of the Requirements
for the degree of
Master of Science

Contents

List of Figures

Chapter 1

Introduction

Teleconferencing capabilities are becoming increasingly accessible to the average user of the Internet and are enabling users who are distributed across the network to collaborate *synchronously*, i.e., in real-time. The pervasiveness of this technology relies on the maturation of an infrastructure that more fully supports synchronous rendezvous. Toward this end, we have focused on one facet of the infrastructure, user location.

We have created a user directory service that dynamically distributes and tracks the location of users in the network. It employs *multicast*, an efficient group data distribution technique, as the basis for scalable communication. Its design was motivated by an interest in telecollaboration over the Internet and the specific problem of allowing users to find other users in order to convene multimedia sessions.

This chapter presents the context for this work and the origins of the multicast user directory idea. We describe the process of explicit-invitation session orchestration, and its need for user location assistance. Multicast addressing and multicast communication are defined, and the benefits of incorporating them in a user registry service are outlined.

1.1 Session Orchestration

Our interest in the user directory problem stems not only from the burgeoning field of computer-supported cooperative work, but also from our field testing of `mmcc`[1], a software tool that performs *session orchestration*, i.e.,

[1] `mmcc` is an acronym for multimedia conference control tool.

initiates multiperson networked collaborations [21]. Each `mmcc` acts on behalf of the user who runs it. When a user requests the initiation of a *session* – a conference consisting of a variety of underlying media or applications – that user's `mmcc` sends invitations to peer `mmcc`s, each of which acts on behalf of its user. Each recipient `mmcc` conveys the invitation to the local user, then forwards the user's response back to the *initiator*.

Although this protocol is quite straightforward, the task of finding the address for remote users in the Internet is less so. Early conferencing applications avoided this issue as they were designed for operation within the local area network or for usage among a known collection of users [27][2][1]. Frequently, the set and locale of potential participants were known a priori. However, with the migration of conferencing services out of the local-area and into the wide-area, finding remote users over the general Internet becomes problematic not only because of the tremendous numbers of potential users, but also because of increased user mobility.

1.2 A Multicast User Directory Architecture

We augmented `mmcc`'s original session coordination functions with a user directory function; the directory service relies on a multicast-based protocol both to distribute and to track user location.

To register locale, each user makes periodic announcements to a *multicast* address. User directory announcements are *scoped*, meaning they are distributed within a bounded region. As a result, scoped announcements allow a user to determine the effective network *distance* of the sender of an announcement. In addition, each user listens for announcements from others, building up a personal directory of the reachability of remote users.

By combining the session orchestration function with the user directory service, we can accomplish two tasks that had previously eluded *explicit-invitation* teleconferencing applications, i.e., applications that issue a user invitation request rather than those that allow users to request to join. First, by using multicast messages, user location can be determined dynamically, which provides a bootstrapping mechanism for call initiation. There is no longer any need either to hardcode user addresses or to know potential collaborators' addresses in advance; remote user addresses will be discovered during the listen phase of the algorithm. Second, if we can determine the approximate distance between pairs of users, then we can extrapolate the maximum distance or network diameter needed to encompass a group of users. This *group scope* value can be used during session orchestration to bound data distribution among session members. In scenarios where teleconferences may include audio, video and groupware data, a more precise scope for data distribution can result in significant bandwidth savings.[2] In addition, a more accurate scope for the multicast addresses used to send the real-time data allows better address re-use of these same multicast addresses in other parts of the network.

1.3 Multicast and the MBone

Why base the user directory architecture on a multicast protocol? Presently, the most successful large-scale experiments in telecollaboration have occurred within the Multicast Backbone (MBone), a multicast-capable portion of the Internet[4]. The MBone relies on Internet Protocol (IP) multicast at the internetworking layer to propagate data efficiently to a group of destinations[10].

A multicast address is distinguished from its *unicast* counterpart in several ways:

- A multicast address is selected from a range in the Internet address space, `224.x.x.x`, whereas a unicast address can reside anywhere else in the address space (from `0.0.0.0-255.255.255.255`). Here, addresses are indicated in dot notation, where each byte of the 32-bit address is separated by a dot and is represented by its decimal value.

- A multicast address is dynamically assigned on an as-needed basis, whereas a unicast address is statically mapped to a particular end system. Thus, multicast addresses are a limited resource to be shared among all group sessions.

- Each multicast address represents a group of end systems which appear to the network as a single group address, but each unicast address denotes a single host end system.

To distribute data to a group, an application simply sends its data to the group multicast address. The advantage of IP multicast is that routers, not applications, handle the N-way distribution of data to the members of a multicast group. The routers ensure only one copy of a message ever

[2]Savings are notable in those cases where routers disseminate data as far as the scope allows, even if no participants exist at that distance.

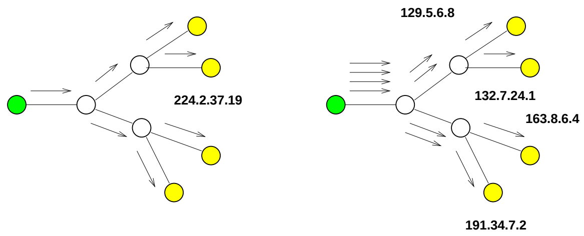

Figure 1.1: Multicast vs. Unicast Distribution Trees.

traverses a physical network link, as compared with potentially N copies if applications themselves were to distribute the message to all intended destinations, as displayed in Figure 1.1. As data propagates from a source to its destination(s), copies of the data are made only at the branching points – the routers – in the multicast distribution tree.

As a result, a new class of applications has emerged that exploits multicast's improved efficiency. The reduced overhead of communicating to groups with large membership means that applications should rethink the manner in which data is both distributed and cached. Re-examination is needed of traditional questions that trade off *pushing* data into the network, i.e., distributing data without an explicit request for it, versus *pulling* data from the network, i.e., only distributing data in response to an explicit request.

1.4 Overview

In this thesis, we present a specification for the user directory algorithm, and describe the tradeoffs regarding addressing and naming that accompanied its design. We discuss the strengths of using a multicast approach (resiliency, scoping, mobility), as well as issues of scale that arise with a fully distributed architecture. We highlight several implementation hurdles, characterize the operation of the software prototype, and offer early models to evaluate the utility of the user directory.

The initial user directory architecture is a fully distributed, multicast-based approach that periodically pushes data into the network. We describe extensions that couple it with data-pull approaches to create a hybrid architecture for improved scalability. We summarize ongoing work in this area, then ask several key unanswered questions about using the multicast archetype for tackling more general problems of resource discovery and registration in the wide-area.

Chapter 2

Rendezvous Models

User-to-user rendezvous can be categorized in a variety of fashions; explicit vs. implicit, synchronous vs. asynchronous, point-to-point vs. multipoint.

- *Synchronous vs. asynchronous* interactions. Interactions in real-time are synchronous, whereas non-real-time interactios are asynchronous.
- *Point-to-point vs. multipoint* communication. We classify two-way user interaction as point-to-point communication, and multi-way interactions as multipoint communication. Two-way is a special case of multipoint communication with two users.
- *Explicit vs. implicit* initiation models. Explicit initiation uses invitations to request user participation. An implicit join approach publishes an address where a user can "tune in" to a session without getting pre-approval to join; thus the invitation is implicit.

The user directory service was designed to facilitate synchronous multipoint interaction. In particular, it is closely coupled with an explicit-invitation model and complements existing implicit join techniques[36][38]. In this chapter, we concentrate on these last contrasting attributes, highlighting a range of user location issues that influenced the design of our directory service.

2.1 Explicit Invitations

Explicit rendezvous mechanisms are characterized by one user explicitly contacting another. Telephone calls, e-mail messaging, and Web-based queries all fall under this category. All three forms of communication raise the following key question: What addressing scheme is most appropriate for identifying locale in the network, and user locale in particular?

2.1.1 Telephone Model

Telephone addressing is a good approximation of what is needed to contact a user on the computer network. It is modeled around calling a particular location to contact a particular individual. The computer equivalent of a telephone number is the Internet host machine name, e.g., `chopin.cs.caltech.edu`. To find an individual at that host requires coupling a user login name with the host machine name, e.g., `schooler@chopin.cs.caltech.edu`. Although a computer corollary exists for telephone addressing, the simple mapping of user to computer location is insufficient for solving the larger user location problem, i.e., where might a user reside at a given time? Unlike the phone network there is no centralized administration, and as a result user addresses are not widely available through white pages services.

Another caveat is that computer-based user addresses rely on the uniqueness of user and machine identifiers (ids); although official host machine names are unique[1], user login identifiers have no such guarantees. Furthermore, a single static user-machine combination falls short when users are increasingly mobile and reachable at multiple locales.[2]

2.1.2 E-Mail Messaging

Electronic mail (e-mail) allows users to be identified with any number of addresses, by creating a level of abstraction between the address for a user and the actual location of the user's mailbox. A user receives e-mail at a main postbox, e.g., `schooler@cs.caltech.edu`, and all related e-mail addresses, e.g., `schooler@chopin.cs.caltech.edu`, forward mail to that repository.[3] Consequently, forwarding mechanisms unify a diversity of addresses, and users appear to be locatable through any of them.

[1] By an official host machine name, we mean that the host's domain name (e.g., `caltech.edu`) has been officially registered with the IANA, the Internet Administrator of Network Addresses. We assume that local system administrators are careful to assign unique host names to machines within a domain (e.g., `chopin.cs.caltech.edu`).

[2] This is differentiated from the mobile machine problem in that users, not machines, are mobile. Users migrate from machine to machine, or are engaged in computations on multiple machines at once.

[3] Hand-tuned forwarding files also help unrelated e-mail addresses forward mail properly.

Because e-mail is a form of asynchronous communication, user address resolution need not be performed in real-time. Nonetheless, e-mail has been proposed as a rendezvous mechanism for teleconferencing, e.g., an initiator sends an invitation message in e-mail requesting participation in a session, then e-mail is used to launch the needed application resources for the session. This approach leverages off the existing e-mail infrastructure. It takes advantage of the many users who are not only reachable via e-mail, but also who readily publicize their e-mail addresses, and thus can offer high user *buy-in* [3]. However, this scheme requires people to read mail often (at least to respond to an invitation in a timely fashion) and with high volumes of e-mail, a user might even have difficulty finding the invitation. More sophisticated mailers can prioritize incoming e-mail or augment a mail message's `Subject` line to tag the message as important.

Although the e-mail model supports multiple addressing mechanisms to reach a user's mailbox, it obscures the user's actual locale, since the user's e-mail address is often not the machine at which the user is actually working. Yet for teleconferencing, the actual machine location is important, because we often assume that multimedia data can be routed to that location and there will exist hardware devices to support the real-time media (e.g., integrated audio or video in the workstation). Another drawback with indirect addressing is that it may add perceivable delay for real-time services if the media needs to be routed indirectly.[4]

2.1.3 Web-based Queries

The recent phenomenon of the World Wide Web (WWW or the Web) has led to the creation of yet another user-specific address; the Web homepage. Although not often used as the point for synchronous rendezvous, it has begun to serve as a launching off point [8], in the same way that one's e-mail address serves this purpose. Because the Web homepage address often is widely distributed, the easy user buy-in argument applies here as well. Perhaps the most compelling aspect about a Web page is that its links connote associations among pages. Potentially, one could use these link relationships to derive addresses for groups of associated users. However, the homepage is location-independent; the homepage itself resides on a server machine, not on the machine where the user is currently working. Therefore,

[4]Although the control messages may be routed indirectly due to indirect addressing, the control protocol may have the know-how to compensate for this and actually route the real-time data directly between sender and receiver(s).

the homepage address has all the disadvantages of the e-mail address.

2.2 Implicit Joins

One way around the user location problem is to design conferencing systems that have implicit naming schemes. For instance, LBL's session directory tool, `sd`, provides a multicast address for users to join to participate in a session [36]. This circumvents having to know who else is part of a session or whom to contact for approval to join. Instead, participation is locally controlled by each individual joiner. The implied session policy, is to allow anyone to join who obtains the session address itself; this is akin to selecting a particular TV channel. Collaborative Web schemes also operate in this fashion[18]. When users browse on the same Web page, they become part of the same synchronous real-time session. Again, there is an implied agreement to be part of the same association and there is no explicit invitation process.

However, these approaches are commonly used to create public sessions, where the identity of other participants may not need or care to be known. Also, there is still the issue that an outside service must exist that publishes the whereabouts of the session information. Effectively the problem has become abstracted one level, since the address of the outside service must now be made known somewhere.

2.3 Goals

Ultimately, we want to build a user directory architecture that synthesizes the best attributes of each of these models: the intuitive mapping of person and equipment provided by the telephone model; the multiplicity of addresses, and thus mobility, provided by e-mail; the natural group associations readily offered through Web pages; and the simplicity offered by implicit naming models (everyone joins a single channel vs. knowing all group members' addresses).

Chapter 3

System Architecture

To experiment with an appropriate naming and addressing model for a user directory in the Internet, we have extended `mmcc` to serve not only as a session orchestration tool, but also as a user registry or locator service. In this chapter, we discuss the interplay between these two functions. We describe the rendezvous protocol, the user directory protocol, and how they leverage off each other.

3.1 Session Control: MMCC Overview

3.1.1 Rendezvous Protocol

The `mmcc` tool is an agent that acts on behalf of the user who runs it. To conduct multiparty conversations, peer `mmcc`s participate in a request-response messaging protocol[12]. When a user initiates a multimedia session, that user notifies its `mmcc`, becoming the initiator of the session. The initiator's `mmcc` sends invitation requests to the remote `mmcc`s of users invited to participate in the session.

The invitation message encapsulates a session proposal, with enough details to allow each user to know the salient characteristics of the suggested session, and consequently to accept or to decline the invitation to participate. For instance, a session description might convey who else has been invited, the topic of the session, configuration information such as which media to spawn and which media encoding schemes to use, the intended start and finish time of the conference, among other details[13].

Once a request is received by a peer `mmcc`, the associated user is notified by the graphical user interface (GUI) and asked to respond to the invitation.

The invitee's `mmcc` eventually either sends a user response or allows the request to expire. The initiator's `mmcc` is responsible for collecting these responses from the remote `mmcc`s and retransmitting the invitation until either a reply is received or its own timeout is reached.

Subsequently, the program spawns familiar MBone audio, video and groupware tools (e.g., `nv`[9], `vat`[35], `wb`[6]) that allow participants to collaborate over the network in realtime. When these tools are executed, `mmcc` dynamically selects an appropriate multicast address or addresses – separate from the user directory address – over which they share multimedia data.

3.1.2 User Directory Evolution

To locate remote collaborators, `mmcc` originally allowed a user to set up personal address books using a combination of:

- *a startup file*, read in when the program is initiated and written out when exited.
- *on-the-fly entry from the graphical user interface*, so a user can be added dynamically to the registry list, and `mmcc` subsequently can initiate a call to that user.
- *automatic updates* as new users call.[1]

The work presented in this thesis on multicast user directories augments `mmcc`'s original approaches for tracking user addresses. A main benefit of the new work is to move away from static, hand-configured approaches and to include additional automatic techniques for finding users, ultimately allowing `mmcc` to act as a dynamic user locator service. A multicast directory provides an easy method for discovering addresses of others who are able to accept calls, i.e., others who are running the session initiation program, `mmcc`, and thus are presently reachable and potentially available.

3.2 Directory Protocol Overview

3.2.1 Multicast Channel

The basis for the registry algorithm is that user location information is announced to a common multicast address that serves as a *conference bus*,

[1] Issues relating to duplicate addresses for the same user, and conflicts between old and new addresses are discussed in detail in Section 6.2.

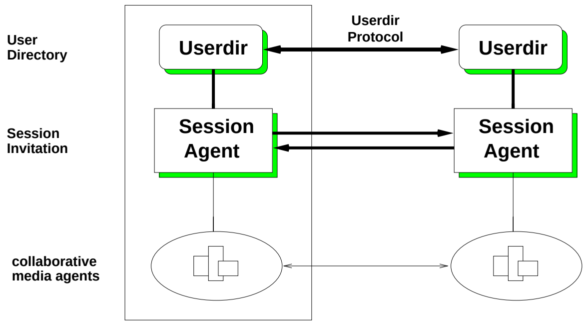

Figure 3.1: Control Flow

a shared channel for coordination[17][14]. Because multicast routing distributes data to the group of destination machines subscribing to a multicast address, we can use a multicast address to disseminate user information to groups of cooperating user registries.

For example, several existing packet audio tools are able to arbitrate the use of a local machine's audio device `/dev/audio` in this fashion. They use a locally-scoped multicast address and require that any facilities needing to share the audio device join the multicast address to participate in the arbitration process [33][35][25].

In reality, the user location announcements are sent not only to a multicast address, but also to a specific port, which in turn identifies the information as belonging to the user directory protocol. This mimics the way communication ports are used under Unix; if a program opens a connection on a well-known port then it effectively agrees to participate in the protocol that *resides* at that port. Thus, all machines subscribing to the user directory multicast address and port agree to participate in the user directory protocol.

3.2.2 Multicast Messages

The generic registry algorithm is quite simple. If enabled,[2] `mmcc` uses a fixed multicast address and port to announce the user's availability at a particular locale. The announcement message is considered a *keep-alive* message because it registers that the user is still reachable or live. `mmcc` also listens

[2]By default this capability is turned OFF.

on this address for announcements from remote users and adds those users to its directory. Note that the registry address is different from the session control address for session management, and from the address(es) used to distribute the media flows (e.g., audio, video, et cetera), as shown in Figure 3.1. Note also that the session invitation protocol is most often used in unicast mode (but nothing precludes it from sending its control messages to a multicast address), whereas the user directory protocol is multicast-based, as are the multimedia data flows.

The service is not only a technique for finding users, but for finding communities of users. In other words, the act of registration lets users see how widespread a collaborative tool is in use. In the case of `mmcc`, it learns not only a user's address, but if the user has a "phone", i.e., is running the session orchestration application, `mmcc`, at the time a rendezvous is desired.

3.3 Algorithm Refinements

3.3.1 Periodicity

The protocol distributes the latest user information via periodic announcements. The announcements are unreliable messages. They do not require acknowledged receipt. Empirical evidence shows that the probability of message receipt is high across uncongested paths and low in areas of congestion. The implication is that a user is effectively unreachable if the route leading to the user is congested. This is the desired behavior because there is no point trying to initiate a multimedia session if control messages are regularly dropped due to buffer contention in the routers or in the end systems.

The periodicity of the announcements allows `mmcc` to track changes in the status of a user's address, and in effect the status of the reachability of that user. As a result, new registrants eventually will obtain both old and new peer information. Old registrants will learn about new registrants and also which registry information has become *stale*, i.e., if no new announcement has been received to renew the validity of the old entry, then the old entry will timeout. Consequently, the user directory has potential to be quite dynamic, an advantage in highly volatile settings, such as distributed simulation.

Because periodic messages continually refresh state information, this type of protocol has come to be known as generating *soft-state*, and is the method used in emergent routing and reservation protocols [11][28]. We simply apply this multicast-based technique to the user directory problem. Besides the resiliency of a soft-state approach, we can use periodic multicast announce-

1	=	local
2-31	=	site
63	=	region
127	=	world

Table 3.1: Standard time-to-live (ttl) thresholds for scoping.

ments to track not only the staleness of an address, but also the changes in a particular user's locale.

3.3.2 Rings of Scope

We use the *announce-listen* protocol to convey user-level information as well as scoping information. A scope or *time-to-live* (ttl) value is embedded in every packet to bound propagation of that message. In the unicast world, the ttl value typically is decremented as the packet travels through each router. When the ttl becomes 0, the packet is discarded; this is intended to curtail problems with packets that might be caught in routing loops in the network. In the multicast realm, ttl's are combined with link thresholds to stem the packet propagation. If a packet's ttl value is smaller than the threshold on the desired link, then a router will not forward the message.

We look to the MBone for typical scoping granularities and emerging ttl guidelines. A *local* scope is defined as a ttl of 1 and means that a message with a ttl value of 1 is discarded by routers.[3] There is quite a bit of variability in the definition of ttl values for *site* scoping, but ttl values in the range 2 to 31 have been used and denote some sense of site or experimental testbed network. A *region* setting for a community boundary, or regional network, often takes on the ttl value of 63, whereas a *world* ttl of 127 encompasses the globe.[4] Standard ttl thresholds are summarized in Table 3.1.

Each announcement message includes at the application level the ttl value used to distribute the message at the internetworking level. In other words, although the ttl is a value that is embedded in IP headers parsed by routers, the user directory messages generated by `mmcc` also include that ttl value. The ttl value is shared among peer `mmcc`s in order for the user

[3]A ttl of 0 is also legal but is only used among processes local to a single machine, e.g., this technique has been used for audio device arbitration.

[4]Ttls above 127 are typically used for bandwidth constrained distributions.

directory to determine network distances among users. If an announcement message is received by a remote user, it implies that the embedded ttl is an appropriate scope for the announcer to reach that recipient.

Each `mmcc` makes user directory announcements at different ttl values, announcing more frequently within smaller regions (lower ttl), and less frequently within larger regions (higher ttl). Each remote `mmcc` stores the minimum ttl (as inserted by the sender) at which the announcement was first heard, so users can approximate network distances between other users in the registry.

For instance, in Figure 3.2 user `p` announces its location. The first message is sent at a ttl that does not reach the other users `r`, `s`, and `t`, as depicted by the first "ring" of scope. An announcement message with the next largest ttl reaches user `r`. An announcement with an even larger ttl reaches both `r` and `s`, and the largest ring of scope covers all users, `r`, `s`, and `t`.

Ultimately, scoping information allows a group scope to be derived for a collection of session participants. When a teleconferencing session is created, `mmcc` selects the maximum of the minimum ttl values between all group members and includes this value as part of the session description in the session invitation protocol[12]. The group scope in turn is passed to the associated media tools for proper scoping of the multicast real-time data. The tighter the approximation of the group scope, the more effective the address re-use in other parts of the network for the multicast address(es) used by the media agents.

3.3.3 Staleness

A user registry entry is only considered accurate for a finite period of time. Once an announcement message is received, a timer is set to mark when the directory entry should expire. The timer is typically associated with the minimally scoped announcement from a given remote user (see Section 6.2 for a discussion of alternate approaches). If the timer goes off yet finds a smaller scope than when the timer was set, then it does nothing; an alternate message with a smaller scope was received in the interim and a separate timer has been set.

The timer duration can be set in a number of ways. It can be a fixed value that is configured into the system. Or, it can be inferred from observation of the network and dynamically adjusted over time. The timer also may be provided by the remote user who embeds an expiration time in the

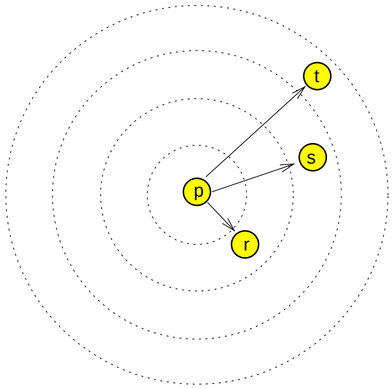

Figure 3.2: Periodic Announcements with Expanding Scope.

message itself and thus indicates when the receiver should expect the next announcement message from that user.

If the timer expires, the directory marks the entry as no longer valid. We use gradations of staleness before considering the entry entirely outdated. Typically data aging occurs as a function of the timer associated with the minimum ttl entry; the first timer expiration is considered delay in the network or suggests user mobility; a second timer potential unreachability; and a third timer expiration that the user may no longer be participating in the registry algorithm, at which point the user is *retired* from the registry; in Sections 4.1.6 and 6.2 a more precise definition of entry retirement is given. The timer interval is doubled with each expiration.

3.3.4 Message Content

Table 3.2 displays the format of an announcement message. Following the trend of HTTP, SDP, and other text-based protocols [29] [13] [30], the directory announcement messages are sent as text for easy portability and parsing. The generic contents of the announcements include four attributes: the user alias (`u`), user login id (`l`), host machine name (`h`), and host machine address (`a`). These constitute the required elements of an announcement message. The purpose of having multiple forms of user ids is that some fields are more easily parsed by machine whereas others are intended for other users. In addition, by providing alternate forms of the same information,

```
u="Eve Schooler"
l=schooler
h=chopin.cs.caltech.edu
a=129.5.6.7
t=31
d=5
m=a
```

Table 3.2: Announcement Message

heuristics can be used to help deduce user mobility as discussed in Section 5.3.

The optional fields include: a ttl field (`t`) that indicates the scope of the announcement message and that changes as the scope metric is increased or decreased; the periodicity of or the delta (`d`) between announcements at the given ttl in seconds, which in effect indicates an expiration time for the validity of the entry; and finally, a message type (`m`) that specifies whether the message is an `Announcement` (`a`) or a `Bye` (`b`) message – the `Bye` may be sent when a user explicitly ends participation in the registry protocol.[5]

[5] In the event that an optional fields is omitted, the default values are `t=127`, `d` is set to whatever the local registry's periodicity is for the given ttl, and `m=a`.

Chapter 4

Algorithm Specification

4.1 Pseudocode

The algorithm used for use location is entirely asynchronous. After setup, the algorithm consists of an announce phase and a listen phase. Announcement events are timer-driven (messages are sent at the completion of a timeout period), whereas listen events are entirely message-driven (invoked on the receipt of a message). These two types of actions operate independently from each other, and may interleave in an arbitrary fashion.

4.1.1 Initialization

The setup process `userdir_init` establishes a multicast channel on which announcement messages will be sent to and received from other `mmcc` users, i.e., users who are actively running the application and with whom one might potentially conference.[1] Each user makes announcements to the multicast channel at a series of scopes (ttls), each with an associated timer value. The initial (default) settings for ⟨ttl,timer⟩ pairs for local, site, region, and world are stored in the variable pairs ⟨`local_ttl`, `local_timer`⟩, ⟨`site_ttl`, `site_timer`⟩, ⟨`region_ttl`, `region_timer`⟩, ⟨`world_ttl`, `world_timer`⟩, and have the values ⟨1,5⟩, ⟨31,130⟩, ⟨63,530⟩, ⟨127,2100⟩ respectively. The ttl is in hops[2], and the timer is in seconds. Timers are deliberately slowed from a periodicity of seconds, to minutes, to tens of minutes, to hours, as the scope is increased from local to world distribution.

[1] The user directory's multicast address is presently 224.2.127.254.

[2] The number of routers or gateways through which a message can maximally traverse.

Next the initialization routine creates a template message that consists of the four required announcement fields to which it easily can append the optional fields – in particular the changing ttl and delta fields. The registry sets different timer events for the different ttls, and provides `userdir_announce` as the routine to call when the announcement-related timers expire. Note that `set_ttl_timer` stores the next timer interval and the ttl value as handles to be passed to the callback routine, `userdir_announce`, so that they can be used to update the template message as needed. They are actually bundled by `set_ttl_timer` into a single `Handle` data structure, which is a pointer for easy manipulation and casting. Finally, `userdir_listen` is established as the callback routine that is triggered when an incoming message is received. A `receive_message` event is set when incoming data is detected on the multicast user directory channel.

```
int
userdir_init(void)
{
    /* channel establishment: for xmit and receive */
    init_multicast_channel();

    /* create generic announcement */
    create_template_message();

    set_ttl_timer(local_timer, local_ttl, userdir_announce);
    set_ttl_timer(site_timer, site_ttl, userdir_announce);
    set_ttl_timer(region_timer, region_ttl, userdir_announce);
    set_ttl_timer(world_timer, world_ttl, userdir_announce);

    set_callback(receive_message, userdir_listen);
}
```

4.1.2 Announce Phase

When an announcement timer expires (there is one timer per ttl scope), the routine `userdir_announce` is called. It calculates an effective announcement periodicity for the given ttl using `calc_next_ttl_timer`. If the suggested message fields differ from the previous announcement, the appropriate fields of the message template are updated; the `set_template_message` routine marks the message as an `Announce` message to be sent with a scope handle->ttl and with a periodicity of next_timer seconds. Then, the local user sends the message to the multicast channel. Messages are sent by send_message using UDP datagram sockets as the basis for communication. No acknowledgement is expected and message delivery is not guaranteed. Finally, the announcement timer is reset for that particular ttl to go off again next_timer seconds in the future.

```
void
userdir_announce(Handle handle)
{
    /* calculate next timer value */
    next_timer = calc_next_ttl_timer(handle->timer,
        handle->ttl);

    /* update template msg, if needed */
    set_template_message(Announce,
        handle->ttl, handle->timer);

    /* send the message */
    send_message();

    /* reset timer */
    set_ttl_timer(next_timer, handle->ttl, userdir_announce);

}
```

4.1.3 Bye Message

When the user wishes to exit mmcc or simply to stop sending announcement messages, userdir_bye provides a mechanism to notify the other participating registries of this action. This is in lieu of allowing remote registries to use timeouts to infer that the user is no longer active.

All pending ttl-related timer events are reset in reset_ttl_timers. Then, the message is tagged as a Bye (vs. Announce) message and is sent to the largest possible scope, so it reaches all participating registries. No delta or periodicity interval is appended to the message, since the Bye message is meant to cause the affiliated user entry to be removed immediately from the recipients' user directories.

```
void
userdir_bye(void)
{
    /* turn off all ttl-related timers */
    reset_ttl_timers();

    /* update template msg */
    set_template_message(Bye, world_ttl, 0);

    /* send the message */
    send_message();
}
```

4.1.4 Timer Calculation for Announcement Periodicity

The timer for announcement periodicity is calculated in `calc_next_ttl_timer` and is a function of the announcement scope or ttl. If the registry policy is to have fixed timers for announcement messages, as is indicated by the global flag `Fixed_TTL_Timers`, then the next expiration time is strictly a function of the timer associated with the given ttl.

However, if the registry policy is to support adaptive timers, then the periodicity calculation takes several factors into account. First, it determines the `percentage` of the registry being used for announcements at the given ttl. Second, given the average message size, `Avg_Msg_Size`, and the maximum cumulative bandwidth, `Max_Data_Rate`, to be used by the user directory protocol (across all sites), it calculates the periodicity `interval`. However, because all ttl scopes have a minimum timer value, which can be found by calling `min_ttl_timer`, `next_timer` is set to the minimum of `interval` and the minimum timer value, `min_interval`.

To avoid timers getting synchronized across participating registries[7], `unsynch_timers` performs the final adjustment to the `next_timer` interval, by randomly setting the timer to an amount from the uniform distribution on the interval $[0.5T, 1.5T]$, where T is `next_timer`.

```
Timer
calc_next_ttl_timer(Timer *timer, TTL *ttl)
{

    int next_timer;
```

```
    int total_entires, ttl_entries;
    float percentage;
    int interval, min_interval;

    /* if ttl timers are fixed */
    if (ttl_timer_policy) == Fixed_TTL_Timers) {
        next_timer = *timer;
    }

    /* if ttl timers are adaptive */
    else {
        /* determine total users in registry */
        total_entries = get_registry(num_entries, Total);

        /* determine total users in registry with given ttl*/
        ttl_entries = get_registry(num_entries, TTL, *ttl);

        /* percentage of registry used for this ttl */
        percentage = (float) ttl_entries/total_entries;

        /* periodicity interval */
        interval = Avg_Msg_Size/(percentage * Max_Data_Rate);

        /* meet minimum threshold */
        min_interval = min_ttl_timer(*ttl);
        next_timer = (interval < min_interval ?
                min_interval : interval);
    }

    /* make sure timers don't get synchronized */
    unsynch_timer(&next_timer);

    return next_timer;
}
```

4.1.5 Listen Phase

The `userdir_listen` routine performs a non-blocking receive on the multicast channel. It is called when a message arrives. First, it determines if this is a `Bye` or `Announce` message. If the former, then it immediately retires the entry associated with the user information in the message with `retire_entry`, which simultaneously notifies the GUI and turns off any timers affiliated with the entry.

Otherwise, the routine tests if this is a new or old user entry. It determines this by searching for an entry with similar user information (excluding comparison with the information in the optional message fields). If no entry is found, a new entry is added both to the registry and added to the GUI using `create_new_entry`. Any fields which have not been specified in the announcement message, such as the ttl, are given default values.

If the announcement message's user information is identical to a previous entry, then `reset_expire_timers` is used to reset any outstanding entry expiration timers, and update the entry's ttl value if the ttl value in the announcement message is less. However, if the user information is *related* but not identical (e.g., only the user alias or user login differ, or only the machine name or address differ, but are "near" to the previous entry) then the new entry is associated with the old entry.

Finally, the routine calculates when to expect the next announcement message using `calc_next_expire_timer` and then it calls `set_expire_timer` to set a timer to detect when the user entry has not been renewed; the callback routine `userdir_expire` is called after an interval of time that is a function of the values `userp->ttl` and `next_timer`, as well as the number of previous timeouts, as elaborated in Section 4.1.6 below.

Like `set_ttl_timer`, the routine `set_expire_timer` stores the next timer interval, the pointer to the user information `userp`, and the number of timeouts (0 initially) as handles to be passed to the callback routine, in this case `userdir_expire`. Again these variables are actually bundled into a single `Handle` data structure for easy processing in `userdir_expire`.

```
static void userdir_listen(void)
{
    Entry *userp, *old_userp;
    int next_timer;

    /* receive userdir announcements */
    userp = recv_and_parse_message();

    /* if Bye message, immediately expire */
    if (is_bye_message(userp))
```

```
        retire_entry(userp);

    /* if already know remote user */
    else if (old_userp = find_user(userp)) {

        /* if the user info is identical */
        if (is_identical_user(old_userp, userp)) {

            /* turn off prev timer and update ttl */
         if (old_userp->ttl >= userp->ttl) {
                reset_expire_timer(userp);
                old_userp->ttl = userp->ttl;
            }

        /* add new info for same user */
        else create_new_info(old_userp, userp);
    }

    /* otherwise, add new entry to registry */
    else {

        /* create new entry and notify GUI */
        create_new_entry(userp);

    }

    /* calculate next timer value */
    next_timer = calc_next_expire_timer(userp);

    /* establish timer to detect expired entry */
    set_expire_timer(next_timer, userp, 0, userdir_expire);
}
```

4.1.6 Expiration of Registry Entries

Registry entries are aged over time using a series of timeout events. If the number of timeouts has exceeded the maximum (the default is presently 4), then the entry is retired by the routine `retire_entry`. When an entry is retired, it is essentially marked as inactive, an attribute depicted in the GUI as an entry that is only displayed upon request. A retired entry is only deleted from the registry when space limitations are a concern, i.e., `Max_Entries` has been exceeded. This limit is checked in both `create_new_info` and `create_new_entry`, which will delete a least recently used entry before making an addition.

If the number of timeouts remains below the maximum, `Max_Entry_Timeouts`, then the entry is aged based on the number of timeouts that have expired thusfar. Finally, a new entry expiration timer is set. The timeout interval is doubled with each expiration.

```
void
userdir_expire(Handle handle)
{
    /* last timeout? */
    if (handle->timeouts > Max_Entry_Timeouts) {
        retire_entry(handle->userp);
        return;
    }

    /* increase number of timeouts */
    handle->timeouts++;

    /* age the entry */
    age_the_entry(handle->userp, handle->timeouts);

    /* double the timeout interval and reset timer */
    set_expire_timer(2*handle->timer, handle->userp,
        handle->timeouts, userdir_expire);
}
```

4.1.7 Timer Calculation for Entry Expiration

The base timer calculation for entry expiration occurs in `calc_next_entry_timer`. This routine determines the initial expiration timer value for a directory entry by trying the following strategies in the following order:

1. Make the timer a function of the embedded delta value in the announcement message from the sender, if available.

2. Base the timer on the observed periodicity of the sender, if available. This relies on the registry to collect statistics in `reset_expire_timer` when an announcement is renewed and in `retire_entry` when the announcement ages from active to inactive.

3. Use the timer correlated with the local announcement message for the given scope. It is better to use a timer based on an adaptive value, but because the default settings are the minimum values for the timers, using a fixed timer value is sufficient albeit conservative.

```
Timer
calc_next_expire_timer(Entry *userp)
{
    int next_timer;

    /* use the sender-provided interval */
    if (userp->delta)
        next_timer = userp->delta;

    /* use the observed interval */
    else if (userp->observed)
        next_timer = userp->observed;

    /* otherwise use the local user's announcement timer */
    else next_timer = local_timer(userp->ttl);

}
```

4.2 Shared Knowledge: Who knows what when?

Given the basic outline of the algorithm, we now can reason about shared knowledge among user directories. If we let a user be represented abstractly as a process, the central question becomes: What does each process, p, know about each other process, q?

We assume for now that processes communicate using announcements, (active messaging vs. active querying), and multicast-style addressing (receivers are not likely to be known by senders). The topology need not be static, meaning the number of processes may not remain constant, and the network channels may come and go. Each process p acts on behalf of one user.

In this section we introduce a generic messaging model between processes, then refine it to include user-related content. From the model, we assert what a process can and cannot infer from its registry, and we show how the registry assists with proper session scoping.

4.2.1 Generic Messaging

In the generic case, where p knows nothing about message content, if p receives a message from q, then it can infer:

- the *existence* of a remote process q. Existence is equated with reachability and is represented by a unique network address.
- the time since q last confirmed its existence.
- over time, a history of the regularity of message receipt.

4.2.2 Knowledge Gained: User-related Content

When message content is tailored for a user directory service, p receives additional knowledge with each message. Messages now include user location information, and potentially scope and periodicity details as well. Process p gains additional knowledge of process q:

- a mapping between the network address of q and the user login id, user alias, host machine name contained in the announcement message.
- a minimum scope or distance from q to p, based on the receipt of announcements from q. A first approximation of the minimum distance from p to q, inferred from the known distance from q to p.
- a time period, Δ, within which to expect the next keep-alive message at the minimum scope.

4.2.3 Knowledge Obscured: Lack of Message Receipt

If p receives no message from q after Δ, it could be attributed to any of the following:

- user q has slowed the periodicity of its announcements at the minimal scope.

- user q has changed locations, and the scope and accompanying periodicity have both changed.
- user q is not making announcements.
- unreachability, e.g., network failure or machine failure.

Because of the nature of the announce-listen protocol, in that information is pushed into the network, rather than pulled or requested by recipients, a process making announcements cannot infer which other processes are actually listening, nor which processes have actually stored the sender's user information in their local registries.

4.2.4 Process State

A process p defines the following registry sets:

- $p.registry$ is the set of q from whom p has received recent announcements, whereas $p.stale$ is the set of q from whom p has not received recent announcements, but once did.
- $p.reachable[scope]$ is the set of those q that p assumes are reachable within scope $scope$, because p received a message from these processes containing the ttl $scope$.
- $p.history$ is the set of all announcement messages received at p.

In addition, p stores, for each q, the value $p.minscope[q]$, the smallest scope at which it received a message from q. There also exists a boolean $p.announcing$ which indicates that p is participating in the registry protocol by sending announcement messages.

All of these can be refined further by adding a time attribute, t. For instance $p.scope_t$ would denote the set of q located within distance, $scope$, of p at time t.

To more precisely specify $p.registry$, we introduce two functions used in its calculation:

- $announce(q, t, scope, \Delta)$ is an announcement message from q at time t with scope $scope$. Optionally, the message includes a periodicity interval Δ. This provides an indication of the next point in time, $t + \Delta$, when another announcement with that scope will be sent.

- $f(q, scope, \Delta)$ calculates at p an upper bound on the expected arrival time of the next announcement message from q at the given $scope$. This calculation is based on Δ if included by q in its announcement message; if not, p approximates Δ using $p.history_q$; and if $p.history_q$ is not being maintained at p then the calculation is based on the periodicty used by p to make its own announcements at the scope $scope$. The function f returns a value that is a multiple of what it estimates is the announcement periodicity.

Thus, assuming a global clock, at time t:

$$\begin{aligned} & q \in p.reachable[scope]_t \equiv \\ & \quad (\exists\ t', \Delta :: \quad announce(q, t', scope, \Delta) \in\ p.history_t \\ & \qquad\qquad\qquad \wedge\ (t - t' \leq f(q, scope, \Delta))) \end{aligned}$$

The registry can be defined in terms of the reachable processes as follows:

$$\begin{aligned} & q \in p.registry_t \equiv \\ & \quad (\exists\ scope :: q \in p.reachable[scope]_t) \end{aligned}$$

In other words, $p.registry$ is the union of the sets of reachable processes at all scopes.

4.2.5 Session Scope

For a group session initiated by p:

- $session.members$ is the set of all processes q (including p) participating in the group session.
- $session.scope_p$ is the appropriate $scope$ value that allows p to reach all session members. The value $session.scope$ is the scope that permits any process in the session to reach all other members.

Thus $session.scope_p$ is specified as the maximum value of all session members' minimum $scope$ needed to reach the initiator p:

$$\begin{aligned} & session.scope_p = \\ & \quad (\mathbf{max}\ q : q \in session.members : p.minscope[q]) \end{aligned}$$

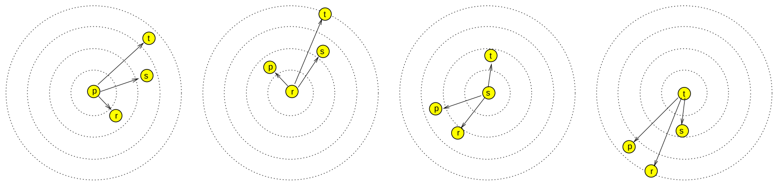

Figure 4.1: Session Scope.

Furthermore, we can derive *session.scope* by taking the maximum of the $session.scope_q$ for all q belonging to *session.members*. In Figure 4.1, $session.scope_q$ is calculated by taking the maximum of $session.scope_p$, $session.scope_r$, $session.scope_s$, and $session.scope_t$.

$$session.scope =$$
$$(\mathbf{max}\ q : q \in session.members : session.scope_q)$$

4.2.6 Changing Topologies

In the event users migrate or replicate themselves within the network, p can determine from *p.registry*:

- multiple locations for the user associated with q, and can order the probability a multi-homed user is at q, versus some other process locale, r. This is accomplished by maintaining *p.history* (or $p.history_q$ and $p.history_r$) and tracking how often a remote user is found during session initiation at q versus r.

- the existence of a proxy registry at q, when multiple users reside there simultaneously. A proxy registry is noted when multiple announcements are continually made from the same address.

4.2.7 Exceptions

Other scenarios that impact process knowledge:

- some q may not make announcements, but may only listen for them (e.g., firewalls, forwarding groups).
- some q may only make announcements, but not listen for them.
- p may hear indirectly about a process q. This may occur out-of-band, for instance at session initiation if a request message from initiator r includes a list of invitees. Even if q does not participate in the registry algorithm, p can infer information learned indirectly through process r; q's existence and user-related information, but not scope, or timeout validity. However, the scope of q can be infered from the group scope of the proposed session, $session.scope_r$.
- there may exist *stable* knowledge in the system, if some entries are always cached and never expire.
- `Bye` messages increase knowledge, i.e., the registry knows for certain that a user has chosen to stop participating in the registry. This is a more precise method than using timeouts, which eventually will lead to the same conclusion but take longer.

4.3 Probabilistic Model

4.3.1 Objective Function

Although we can assert what processes can and cannot learn based on the announce-listen nature of the user directory protocol, ultimately we want to ask how good is the registry. Our objective function tries to minimize the number of occurences of two scenarios:

- **false negative**: $q.announcing$ is true, but q does not appear in $p.registry$. The cost of dropping an active process from the registry is $cost_\alpha$. For simplicity, we will assume that annoucements made by q will reach p at some scope.
- **false positive**: q is a member of $p.registry$, but it is not issuing announcement messages. The cost of keeping a nonparticipating process in the registry is $cost_\beta$.

However, the severity of error is not the same for these two cases. While a false positive may be misleading about a user's reachability, a false negative makes synchronous rendezvous entirely impossible. A false positive provides enough useful information that it might be possible to track down the user at some later point (if the user returns to that location, or the network repairs itself enough for q's announcement messages to be received by p, or through a forwarding entity as in the case of a forwarding group). Thus $cost_\alpha \geq cost_\beta$.

We would like to minimize the number of errors in the registry, weighted by the severity of the errors. Therefore, our objective function is the following:

$$
\begin{aligned}
&Objective function = \\
&\quad (\#\ q :: q.announcing \wedge q \notin p.registry) * cost_\alpha + \\
&\quad (\#\ q :: \neg q.announcing \wedge q \in p.registry) * cost_\beta
\end{aligned}
$$

This function assumes full knowledge of the state of the system—an unrealistic assumption. Instead, we use probabilities to estimate the number of false positives and negatives assuming some knowledge about network reliability and congestion.

4.3.2 Data Confidence

What are the probabilities associated with false positive and false negative errors in the user directory?

Let us consider the simple case in which a process p makes a single announcement at a fixed periodicity, T. T is considered the last time before t, the current time, that an announcement was made, and $2T \geq t > T$. We define a process to be *dead* when it stops making announcements; otherwise, a process is considered alive.

$D(t)$ is the cumulative distribution function of the probability that a process dies at time t, given it was alive at time 0. $M(t)$ is the cumulative distribution function of the probability that a message arrives at t.

For a process to appear in the registry, a message must have been *received* from that process. The probability of a false negative is the probability of a process being alive, given that it does not appear in the registry at time

t. This can be restated as the probability that a process is not dead given that it has not been heard from yet at time t:

$$Pr\{\neg Dead_t | \neg Received_t\} = \frac{Pr\{\neg Dead_t \wedge \neg Received_t\}}{Pr\{\neg Received_t\}}$$

Focusing on the numerator:

$$\begin{aligned} Pr\{\neg Dead_t \wedge \neg Received_t\} &= Pr\{\neg Dead_t\} * Pr\{\neg Received_t | \neg Dead_t\} \\ &= (1 - D(t)) * (1 - M(t)) \end{aligned}$$

By definition, $Pr\{\neg Dead_t\}$ is $(1 - D(t))$. To compute the probability $Pr\{\neg Received_t | \neg Dead_t\}$ we observe that since the process is assumed to be alive at time $t \geq T$, the announcement message has already been sent. Therefore, the only reason the announcement would not have been received would be that it was lost in the network, which is modelled by $(1 - M(t))$.

Focusing on the denominator, we must derive the probability that a message has not been received at time t. This is due to either the process being dead at announcement time T or being alive at announcement time but the message simply not arriving by time t. The denominator is:

$$\begin{aligned} Pr\{\neg Received_t\} &= Pr\{Dead_T\} + Pr\{\neg Dead_T \wedge \neg Received_t\} \\ &= D(T) + Pr\{\neg Dead_T\} * Pr\{\neg Received_t | \neg Dead_T\} \\ &= D(T) + (1 - D(T)) * (1 - M(t)) \end{aligned}$$

Thus, the probability of a false negative is:

$$Pr\{\neg Dead_t | \neg Received_t\} = \frac{[1 - D(t)] * [1 - M(t)]}{D(T) + [1 - D(T)] * [1 - M(t)]}$$

The probability of a false positive is the probability of a process being dead, given that it appears in the registry at time t. This can be restated as the probability that a process is dead given that it has been heard from at time t:

$$Pr\{Dead_t | Received_t\} = \frac{Pr\{\neg Dead_t \wedge Received_t\}}{Pr\{Received_t\}}$$

Focusing on the numerator:

$$Pr\{\neg Dead_t \wedge Received_t\} = (D(t) - D(T)) * M(t)$$

Given that a message was received at time t, the process must have been alive at time T. Therefore for the process to be dead, it must have died in the interval $[T, t]$, which is captured by $(D(t) - D(T))$. Finally, the probability that the message is successfully received is $M(t)$.

For the denominator, we can reuse the calculation for $Pr\{\neg Received_t\}$. Thus, the probability of a false positive is:

$$Pr\{Dead_t | Received_t\} = \frac{D(t) - D(T)}{1 - D(T)}$$

Chapter 5

Evaluation

There are several advantages and disadvantages to using a soft-state multicast architecture for this class of problems. The benefits include inherent resiliency to change in the network, the ability to derive group scoping, and support for multi-homed users and mobility. The drawbacks mostly relate to scaling limitations, but also encompass issues of security and privacy.

5.1 Resiliency

The main benefit of multicast announce-listen approach is its resiliency, especially since the context for this research is the Internet. The sheer number of entities connected to the network is enormous, making stability unlikely. Hosts go off-line or become portable or mobile. Nor is there stability in routing among networked objects, since the topology of the network is ever-changing and routes are sometimes transient. The distributed and dynamical nature of the system means that cooperating resource servers can never entirely know if/when there is consensus about the state of all resources[34]. Thus, we promote a design that does not require consensus. Each individual `mmcc` trusts its own catalog of information as a reflection of its connectivity to the rest of the world. If there is a communication problem with user q, the algorithm detects it within a fixed time period, $f(q, scope, \Delta)$, leading to dynamic updates and automatic recovery.

5.2 Scoping

Scoping for group rendezvous comes for free when using the registry algorithm. The registry stores not only the address of a user in the network, but also the necessary multicast scoping to create streams of data to reach that user. For a session control tool, selecting an accurate scope for data flowing between two or more users is critical. It provides a better bound for the underlying multicast routing tree, which decreases the work done in routers and avoids unnecessary distribution of data to disinterested regions of the network.[1] Additionally, a better bound on the group-wide scope allows more effective multicast address spatial re-use. For instance, the same multicast address with a ttl equal to 1 may be re-used somewhere else in the network if the resultant distribution trees will not overlap. As multicast addresses are dynamically assigned and are limited in number, efficient re-use is important.

The scoping problem is exacerbated when groups of individuals wish to collaborate. Proper scoping is needed among all potential group members. The concentric rings of scope that are used to make announcement messages allow the registry to store the minimum scope at which each site can hear from all other sites (who are making announcements). In combination with something like a session control protocol [22][14], the user registry data can yield the proper group scoping value from pairwise scoping. Stated more generally, the registry can be used to ascertain proper session scoping for group rendezvous. Because there presently is no technique to determine an accurate scope for a group of individuals, ttl estimates are often conservatively high and result in overdistribution; the multicast user directory service is an improvement over the current method of guessing group scope. It accomplishes this by storing enough information about user "distances" to supply a strong approximation for the correct group scope value.

5.3 Mobility

User mobility is also trivially provided; if the periodicity of the announcement messages is high, then user movement is quickly detected.[2] Basically, the algorithm works by detecting when a user has multiple addresses, i.e.,

[1]Data is only distributed to disinterested regions if multicast pruning is not available.

[2]It is not our intent to solve mobility at an IP level, but instead to try to support application-level mobility.

when a user is *multi-homed.* For now, let us assume that a unique user id exists for each individual in the network. Furthermore, let us assume that a user is only considered multi-homed when the same user id is associated with different machine addresses, *from which active announcements are being generated.* This is differentiated from the condition that a user is associated with multiple entries in the registry but fewer than 2 are presently *active.* The registry stores all known user locales for a user, and attempts to order these entries according to its definition of a "most likely" address; presently, the directory considers the address from which is received the most recent announcement the most likely locale at which to find the user. However, you could imagine a different heuristic being used, such as the address at which the user is most often found when making an invitation, or the address from which the remote user last called the local user. When the session control protocol wants to call a user, the registry can provide the most recently active entry among the multiple choices, or the control protocol may use all of the addresses to contact all addresses at once, since only one can return a positive response. Thus, the condition of being multi-homed is really a special case of multiple addresses being associated with a single user.

5.3.1 Multi-Homed Users

To find multi-homed users, the registry examines the contents not only of announcement messages, but also of the startup configuration file, on-the-fly input from the GUI, and session control messages from new users.

The simple case is that a user is multi-homed temporarily. Therefore, the registry transitions between storing two valid addresses for a user because the old user location is in the process of timing out.

However, the condition of being multi-homed is not always transient. The user may have multiple agents running (multiple `mmccs` for now) and these agents will continue to execute, i.e., will not timeout from inactivity. This may occur when the user starts up an agent, then moves to an alternate machine and starts another agent; the user may never explicitly stop any agent. This already happens on the MBone, where an individual tunes in to the MBone Audio channel, then moves to another machine and repeats the process; both instances of the user appear, even though we know the user can only physically reside at one place at a time.

Under these conditions, the registry takes one of two actions. If the multi-homed user does not have the same user id as the local user (e.g., the local user is `schooler@chopin.cs.caltech.edu` and the multi-homed user is `feldy@frisbee.isi.edu`, who also is announcing from the location `feldy@ultimate.myri.com`), then the registry simply makes note of the association between the entries, just as it would if the entries were not active. In other words, the registry will keep track of a set of users who are multi-homed together.

If the multi-homed user has the same user id as the local user (e.g., `schooler@liszt.isi.edu` and `schooler@chopin.cs.caltech.edu`, then the registry creates a *forwarding group* among the related entries, which suppresses redundant announcement messages. The forwarding group is maintained by its members and its protocol is subordinate to the main operation of the registry. It curtails announcements from all but one group member, but all members continue to listen on the same multicast address as the registry protocol, so any changes to the group are heard by all.

5.3.2 Forwarding Groups

There are several components to the forwarding group algorithm:

1. *When to initiate the algorithm?* If the local user detects a similarly named user located at a different address, then link the entry to those already multi-homed to it, and set a timer. The timer should be considerably longer than the natural periodicity of the announcements, or the period during which an entry can go stale. When the timer expires, if there are no other entries to which the local user is linked, then the local user is no longer multi-homed. However, if there are other entries that share the same user id, then invoke the forwarding group algorithm.

2. *Who should lead and who should follow?* The user in the group with the largest machine address becomes the leader and continues to post announcement messages. All other users with similar user ids will stop making announcements. The leader considers the forwarding group algorithm terminated when it is the only member left in the group. If while leading a user detects a new user with a higher address, leadership will be passed to the new user.

3. *How to detect when the group is leaderless?* In the same way that the local user would normally timeout to send another announcement message, the local user would timeout if no announcement were received

from the group leader. Again, this timeout is the duration of the staleness period. If leaderless, then reconstitute a leader. There is no real harm in there being multiple leaders; it only means that there will be more announcement messages than necessary.

4. *When to terminate the forwarding group algorithm?* The algorithm terminates in a variety of fashions. First, it stops when the local user decides to stop running the user agent, in this case `mmcc`. Before departing, the local user sends a `Bye` message to the global registry address, not only to avoid the potentially long timeout period in all registries, but also to speed the membership update of any forwarding group in which the user was participating. To reach all participating registries, the `Bye` message is sent with the largest scope possible. Therefore, when receiving a `Bye` message, the registry algorithm behaves similarly to when the registry retires an entry. Second, the algorithm stops when a forwarding group consists solely of the local user; at that point it is acceptable for the local user to begin announcements again.

One consideration is that the forwarding group leader may not reside at the user location having the smallest ttl (i.e., shortest network distance) relative to an initiator of a session. Fortunately, the user associated with a forwarding group can only ever exist at one of the locales. Thus the leader can forward a session invitation request to all other forwarding group members, but at most one will answer the request affirmatively (note that none of the users may be active).[3] On the other hand, the session invitation protocol may perform the task of simultaneous invitations, if it has stored multiple addresses for the intended user[12][23].

It may be the case that different users cannot "see" each other, meaning that their announcements are not reaching each other due to asymmetric routing. In the situation where these users are actually meant to be part of the same forwarding group, it is harmless for them both to be sending announcements. However, an intermediary user in the forwarding group may be able to hear the announcements from both participants. While not optimal, this situation is harmless as well.

[3] However, if a user agent such as an electronic answering machine is allowed to respond to a session invitation, then we may have the situation where multiple user locations respond simultaneously. Nonetheless, it is still the case that only one user ever responds in the affirmative, as a proxy answering service can only ever respond that the user is unavailable.

5.3.3 Degenerate Case: Inactive Users

Even when a user is not multi-homed, it may make sense to suppress user announcement messages under certain conditions; for example if a user no longer is actively working at a locale, even though the registry is still sending address information. We consider this a degenerate case of mobility. Although this version of the user directory does not eliminate such users from participation in the registry algorithm, a later version could do so.

Activity could be gauged by noting keyboard and mouse events, though this does not accurately assess activity if the user is simply reading a file, listening to an on-line seminar, or is logged in remotely and not using standard input to that machine. Perhaps a better description then of the desired effect is that we want to detect some combination of activity and presence; we want to curtail announcements if the user is no longer present at the locale (e.g., walks away from a machine when participating in a session), but we don't want to stop announcements under false conditions (e.g., inactivity at the keyboard, but present). A couple of alternate mechanisms to gauge presence include tracking activity via something like the Unix `ps` command, or detecting audio input to a sound program, or movement in a video program. Presently, the registry algorithm does not try to detect this degenerate case.

5.4 Firewalls and Proxies

One concern with the multicast registry architecture is the implicit assumption that all end-users are able to communicate with all other end-users. In reality, there is a large (and growing) number of users who sit behind *firewalls*, for privacy and/or security reasons. For instance, often the names of host machines internal to an organization are kept private (are not made known to the Domain Name Service[20][26]), with only an E-mail, FTP, or Web server machine publishing its address externally. While this doesn't deter outside users from contacting internal machines, the idea of announcement messages may run counter to company policy. A more serious problem is if gateway machines actively block data routed across it, unless the data originates from an internal machine, or is in response to an internal request.

In these cases, we employ a *proxy registry*. By "proxy" it is meant that a single agent acts on behalf of a collection of users. This is suggestive of the forwarding chain architecture. Like the forwarding chain, a proxy listens on the same multicast address as the user registry. Because messages are not meant to flow across the firewall, the proxy resides on the firewall or gateway machine and bridges the external and internal communities of users by participating in the announce-listen protocol with both.

Internal users only announce up to the scope of the firewall.[4] The proxy forwards these announcements but substitutes the proxy machine address as the contact address for the user (this requires that all internal users have unique user ids). External users' announcements are forwarded to internal users, after the proxy adds in additional scope for reaching its end-users.

For session control purposes, the proxy address serves as a beacon where initial messages are routed. The proxy in turn forwards incoming queries to the requested addressees, then steps out of the loop. If the firewall normally disallows incoming messages across the gateway, then the proxy helps to reverse the process interactions as always originating from an internal machine.

Unlike the forwarding chain mechanism, the proxy registry does not reduce bandwidth usage. It simply masks the actual machine address where a user resides. Thus, there is still the same number of users wishing to participate in the directory protocol. A minor extension (and optimization) is that the user directory protocol allows an announcement message to include multiple users at the same time, so that the proxy registry can announce its constituency simultaneously.[5]

5.5 Scalability Issues

5.5.1 Bandwidth Consumption

As more users participate in the registry there are more and more announcement messages produced. At what point is the bandwidth consumption of these messages unreasonable? This is partly answered by considering the overhead of each message. In the case of a user directory (versus some other resource registry), a generic announcement consists of a user login id (`schooler`), a user alias that is meant to be humanly readable (`Eve Schooler`), a machine address (`131.215.1.28`) and an accompanying host name to give domain context (`chopin.cs.caltech.edu`). If we consider the overhead of the optional fields, we must also include the announcement's ttl

[4]Note that the proxy group members can receive each others' announcement messages directly on the registry address without the help of the proxy.

[5]Because of maximum transmittion unit (MTU) limits, the proxy registry is not likely to be able to include more than 10-15 user announcements at a time.

(15), the periodicity associated with that scope (5), and the message type indicator (b).

As messages are text-based, and each character consumes 1 byte, an announcement message can be encoded with approximately 128 bytes. This calculation is based on a typical user login id being 8 characters long, a variable length user alias field that is frequently less than 32 characters, a host address of 15 bytes maximum (as there are 4 fields, each at most 3 characters wide and a dot separating them), a variable length host address that is likely to be much less than 32 bytes long (if a host address consists of 4 fields; 3 of which are approximately 8 characters long, and one a 3-character domain name, e.g., .com, with a dot separating the fields). Additionally, ttl requires 3 bytes maximum, ttl periodicity 8 bytes maximum (if time is represented in seconds and is assumed to be less than 3 years in duration), and the message type 1 byte. Finally, there are approximately 21 bytes of overhead that help with message parsing (3 per field; two characters to frame the beginning of each field, and one to frame the end). This compares with a binary message that is only 25 percent smaller.

5.5.2 Adaptive Periodicity

Bandwidth consumption is also a function of the periodicity of the announcement messages and the number of users participating in the directory protocol. This can be kept constant through adaptive periodicity. There is precedent already in RTCP and sd[24] [36]. The announcement periodicity is adjusted based on the number of other announcements heard at that ttl. These adjustments occur at each ring of scope. Thus in the aggregate, we can keep the bandwidth usage below some upper bound; for an initial estimate for bandwidth etiquette, we look to LBL's session directory tool, sd, which uses less than 1kbit/sec in the aggregate. Yet, if timer values get too large, then periodic announcements becomes less effective for mobility.

5.5.3 GUI Presentation

From a usability standpoint, the GUI must also scale. In particular, it must support filtering to reduce the numbers of entries to display. Example filters are ttl thresholds, domain name groupings, and degrees of information staleness. In addition, the GUI must provide multiple views of user entries, e.g., the lexicographic ordering of any announcement field (user id, user alias, host name, host address, ttl or periodicity).

5.5.4 Data Structures

To support GUI filters and lexicographic sorting, each user entry is dynamically allocated and is part of a binary tree. The links of the binary tree are updated as a new filter or sort function is requested, or as entries are added or expire. As a precaution, the directory has a maximum number of user entries it will track at once. If the threshold is exceeded, the oldest (least recently renewed) entry is deleted. Another technique that will clear an entry from the registry is the receipt of a `Bye` message. If the maximum has not been reached, entries are simply marked as expired, but are not removed from the data base.

5.5.5 Hierarchical Addressing

Although multicast provides a useful group distribution mechanism, it has the unfortunate property of creating a flat information tree. We would somehow like to use the ideas from forwarding chains and proxy groups to form the basis of hierarchies of information, where each level in the hierarchy uses a different multicast registry address.

The danger in resorting to a hierarchical organization is that hierarchies often need pre-configuring, something multicast architectures have handily avoided. Therefore, can we create a self configuring system? One that knows when and how to partition the registry into different levels as the numbers of users grows? Another hierarchy complication is that we might lose scoping information if we cluster users according to a hierarchical structure.

5.5.6 Multicast Routing

Although we would like to avoid producing an Internet-wide multicast spanning tree, it may be unavoidable that large communities of users will result in large multicast routing trees. This of course would depend on to what extent users remain subscribed (announcing and listening) on the directory multicast address. Will multicast routers be able to support such a large distribution group?

With routing algorithms such as Protocol Independent Mulicast (PIM), there are additional operational concerns[11]. What is the appropriateness of sparse versus dense mode PIM? What if the periodicity of the announcements at the application level are several orders of magnitude slower than the announcements used by the routing algorithms? If the frequency is low, then routers will not prune the source-based trees of dense mode and the distribution degenerates to broadcast. With sparse mode, there might be a larger number of prune messages than data itself, resulting in route reconstitution each time a low-frequency registry announcement is made.

Chapter 6

Design Considerations

6.1 Asymmetries: Scoping and Periodicity

A key contribution of the user directory architecture is to convey scope information for group sessions. Thus far, the algorithm has been described as if scoping between source and destination pairs is symmetrical. In reality, this is not guaranteed.

Thus, the minimum ttl value needed for an announcement from agent p to reach agent q may differ from the ttl value needed from q to p. As a result, an agent's registry actually stores information about scoping requirements in the direction from remote user to the local user. This value may or may not match the scoping requirements in the reverse direction. As a first approximation, the registry assumes the values match. To assemble a more accurate scoping threshold for a pair of users to be able to send data to each other entails finding the maximum of the minimum ttl needed in both directions. For a group of agents, this generalizes to the maximum of the minimum ttl needed for each agent to receive from all other agents.

As alluded to earlier, the user directory service provides the group scoping information to a separate protocol, such as a session invitation protocol[12][23]. For an agent that initiates a multiway session, it typically suffices to use the approximate group scope value in a session initiation message; remote agents can re-negotiate later if any of them find the value too small. This approach is sufficient to get the conversation started, especially if the initial session request message is sent via unicast and is itself not reliant on multicast scoping.

Additionally, if a user joins mid-session, a tool such as `mmcc` recalculates the optimal scope and shares it with any media agents (tools such as nv, vat) that also rely on it. As with session initiation, the ttl eventually needs to be large enough to encompass the new joiner, but the accuracy of the ttl can be improved iteratively (a smaller value might still reach some but not all of the group members). The implication is that the ttl of a group session can be inconsistent temporarily among members, but eventually becomes consistent.[1] [2]

In the same way that the user registry caches individual user location information, an `mmcc`-like session tool could easily cache membership information for frequently-assembled groups, and thus store the accompanying group scope value. In effect, group scope information could be precalculated or recalculated at any time for later use.

Variation may exist among the ttl choices of different agents, as well as the periodicity associated with the ttls. The choice for appropriate ttl granularities and timers is a decision local to each agent. Each agent decides how many rings of ttls are appropriate, and what should be the associated periodicities for each ring of scope. In the extreme, an agent may also opt only to listen for announcements, or only to multicast announcements, or to perform other filtering functions (e.g., only up to or beyond a certain ttl) for outgoing and incoming information. For instance, an agent at the end of a low bandwidth link may generate fewer rings of announcements at lower periodicities.

The implication of having varied (vs. fixed) periodicities for specific ttl settings is that it now makes more sense to embed a periodicity value in each announcement. The periodicity of announcements with a given ttl may vary not only between sites, but also within the same site over the lifetime of the registry, e.g., when an agent is trying to stay below a designated bandwidth consumption level. By knowing the periodicity of an announcement at the time it is sent, and knowing the past values for the periodicity of previous announcements from the same remote user, an agent can more accurately calculate the validity or staleness of information in the registry, as shown in

[1]Such an iterative procedure relies on the late-joiner knowing who else is meant to be reached, in other words knowing more about membership than the receiver-initiated model assumes. In addition, which user iteratively changes the ttl and which user detects that the ttl is insufficient needs further thought. Similarly, when users depart from a session, the scope value may need re-evaluation as well.

[2]There is the added question of whether it is worth increasing the scope value for the member who is outside the "majority" scope, i.e., an outlier. An alternate technique would be to provide a unicast tunnel to the outlier, and avoid compromising spatial re-use of multicast addresses.

Section 4.3.2.

6.2 Reachability and Data Aging

Another key goal of the multicast user directory is to convey the reachability of a collaborative community; in the case of `mmcc`, the community is the collection of users who are running the tool and have chosen to make their location known. However, there are additional degrees of reachability that the registry service attempts to convey.

As described earlier, the generic behavior is that when an announcement is heard for the first time it is added to the directory, and a timeout value is associated with it. If it is not renewed within the timeout, the data is considered stale. The discussion up until now mostly has centered around the notion that each agent's registry information is a composite of these announcements. However, the directory is actually a superset of these announcements; the other techniques for an agent to learn about user registry information include a startup configuration file `.mmccrc`, the GUI, and session control messages. We consider these to be less dynamic forms of data receipt, and refer to them collectively as *cached* entries.

The entries combine in a number of ways, and the GUI is used to relay information to the user about their relative accuracy. First, the cached registry information does not have timeout values associated with them. They are always valid entries, but become more "active" (highlighted in the GUI) if an announcement confirms them. If there is a conflict between a cached entry and a multicast announcement, the announcement will always prevail, unless the cached entry was made more recently than the announcement. If a registry entry times out, it goes through a number of stages where it appears to age; the convention for the MBone tools is to grey out un-renewed entries in the GUI and eventually to remove them. Here, we never remove a once-cached entry. Instead we introduce several gradations of GUI feedback, to differentiate between cached vs. announced entries, and between announcements that are current, timed out or unheard from cached entries for which an announcement is never received.

If the user directory only stores the minimum $\langle$ttl,timer$\rangle$ pair, then data aging is a function of the timer value of the stored ttl. However, more precise aging can occur if several $\langle$ttl,timer$\rangle$ pairs are stored, even though the minimum ttl is the one used during the session scope calculation. For example, if all ttl granularities and their accompanying periodicities are known for a specific registry entry, then the entry can be progressively aged as each timer value is reached. Furthermore, the minimum ttl value can be progressively updated. This is less severe than outright removal of a minimum ttl – instead it is gradually increased; this is useful if there has been a slight change in the user's location or network routing.

To avoid the overhead of storing all $\langle$ttl,timer$\rangle$ pairs for all entries, it still can be useful to store simply the pair associated with the smallest ttl, plus the timer associated with the largest ttl. The entry begins to age when the smaller timer interval has passed, and is considered expired in the registry when the larger interval has passed. How much data to store per entry is considered a local decision for each user registry.

6.3 Unique User Identifiers

Earlier we described that the registry allows the creation of a forwarding chain for entries that logically belong together. The registry accomplishes this through explicit mechanisms in the startup configuration file, `.mmccrc`, as well as through subsequent actions either at the command line or through the GUI. However, it is desirable to determine these associations implicitly as well.

If we separate the addressing information from the remainder of the user announcement, the user becomes identified by the alias (`Eve Schooler`) and a login id (`schooler`). Both of these tags are imperfect, since different users across the network may share the same alias or login id. Even for the same user, the login id may not be universal for all machines used by the user.

Unless we assume that announcements with identical aliases or login ids belong together, there is no way for a registry to determine implicitly when announcements are related. This approach would be unreasonable in the wide area, because of the high probability of alias and login id overlap. However, if we make use of the host name and/or address in the user announcements, we can determine when announcements with identical aliases or login ids are emanating from agents in the same vicinity as each other, i.e., in the same approximate name or address space. If so, then the entries begin a forwarding chain.

This clearly works for users who migrate from machine to machine in the local area, but is not useful in the wide area. Thus, we assume more explicit techniques will be used by a user who uses different machines across multiple address and/or name domains.

In those cases where neither implicit nor explicit techniques are useful, we can supply GUI assistance so that users are aware of similar entries. For instance, two users who share the same alias might be displayed differently than other entries, e.g., highlighted or italicized. Or, if such an entry is selected, ask for further clarification by showing the accompanying fields of information that differentiate the entries from one another.

6.4 Parameterization

Although the general algorithm is quite simple and straightforward, it is less clear how to parameterize it for maximal efficiency and utility. We introduce a series of questions raised by the specifics of the user directory algorithm, then ask several more questions relating to the present and evolving usage of the Internet.

6.4.1 Timer Values and Bandwidth Control.

- **How many tiers of ttls to supply?**

  By default, `mmcc` supports four rings of ttl announcements; a local, site, region and world scope. These correlate to ttl values of 1, 31, 63, and 127. There is nothing in the design that requires that announcements be made at any or all of these granularities; these are presently configurable.

- **What are the appropriate timer values for announcements?**

  The timer frequency for the local scope clearly should be higher than any of the remote scopes, with the frequency decreasing as the ttl increases. For each ttl, there needs to be a maximum timer (minimum frequency) of announcement messages to make the service useful. Yet, there needs to be a minimum timer (maximum frequency) to limit bandwidth consumption, as in the RTCP specification. The actual values need to be based on statistics on the frequency of mobility and the numbers of users attached to the network. For instance, for local scoping, the theoretical upper bound for the number of Ethernet users is O(1000). In practice, at Information Sciences Institute (ISI) where there are 10 Ethernet segments, it is more like O(50). At Caltech, the number is O(150).

- **What is a reasonable bandwidth to consume per scope?**

There is a close relation between bandwidth usage and announcement periodicity. Bandwidth thresholds should be based on assumptions of link capacity at various scope *distances*. Clearly, a 10 Mbps Ethernet is able to support many more announcement messages than a T1 link (1.5 Mbps), but many fewer than a 100 Mbps FDDI ring. The difficulty is that we do not want to be too conservative regarding bandwidth usage. In other words, the function should not be strictly decreasing as the number of announcements (users) increases. There is an operational cross-over point where the application merits more bandwidth if there is a larger community of users relying on it - and this may be a good argument for why a generalized (as in separate) locator service makes sense. But, it is still important to have a hard upper limit.

- **Can we recommend efficient parameter combinations?**

  As mentioned in Section 4.1.4, stability of the system depends on the existence of a function that takes as input a scope (ttl), the bandwidth available at that scope, the number of users heard at that scope and overall, and the size of announcement messages, and produces a timer frequency. The intent is that this function adaptively changes the periodicity of announcements so that the program uses no more than a certain amount of bandwidth in the aggregate. Close study of the operating function will allow us to assess under what conditions the architecture is appropriate.

6.4.2 Internet Characterization.

- **Can we estimate the effects of scaling by asking the right questions?**

  Although it is unlikely that a registry service should be based on the premise that it needs to support all 3+ billion people in the world, it is not unreasonable to ask how many telephones exist, since that is the most widely-used entity for synchronous rendezvous. Current estimates suggest about 100 million telephones exist (only some portion of which are used for human-to-human interaction). Another place to look for future estimates is IPv6, which is targetted to support on the order of 10^{12} networks and 10^{15} hosts.

- **How often do users request to rendezvous compared to how often registry announcements are made?**

This goes back to the issue of finding a balance between a centralized and a distributed service; do estimates of future network usage warrant the distribution of requests or data? If we combine a query-based registry with a data-based registry, can we use scope to allow requests to be answered by nearest neighbors?

- **How dynamic is the system and how much data hiding is necessary?**

  Back-of-the-envelope calculations need to be supplemented with collected statistics about how dynamic user addressing is, the degree to which users are multi-homed, and the pervasiveness of firewalls in the Internet.

- **How to avoid designing around artifacts of the current MBone architecture?**

  The multicast infrastructure continues to change and we need to separate the stable from the experimental aspects of its operation. Major changes on the horizon include administratively-scoped addresses, and hierarchical multicast routing.

  Configuration questions also need answering, such as how often do pairwise scope values differ, and when they do differ by how much? How often do MBone thresholds change?

Chapter 7

Related Work

7.1 User Registries

The idea of using multicast to track users in the local area is inspired by work at Cambridge University and Olivetti on active badges that track mobile users, and by efforts at University College London to associate a collection of active badges with a locally-scoped multicast address [15][16]. Influential aspects of the design were that the system did not require explicit user action to participate, that the beaconing interval was higher for mobile users than say mobile machines, and that user availability was part of the location reporting.

This effort is similar in spirit to the Internet Multicast Locator (ILM), an effort at the University of West Virginia, that has focused on using multicast to advertize GPS information (latitude, longitude, plus user information), and to map location information to a browsable electronic map. Although GPS provides a precise mapping of user to physical locale, our user registry is more interested in mapping a user to a specific network locale in order to route data flows to the proper media devices within a real-time collaboration context.

An alternate mapping scheme is provided by a software system at University of Rochester that uses advertisements to map user phone numbers to network numbers, which could be useful for network management, i.e., network debugging, but again the registry does not attempt to be used for synchronous rendezvous.

7.2 Soft-state Protocols

The main inspiration for the user directory protocol were the notions of soft-state and the announce-listen paradigm. This approach appears in several important Internet protocols, such as the Real-time Transport Control Protocol (RTCP)[24], the Resource Reservation Protocol (RSVP)[28] for quality of service in the network, and Protocol Independent Multicast (PIM)[11] that offers both dense- and sparse-mode multicast routing algorithms. All make use of multicast addresses, and reconstitute or expire state as a function of timer events.

However, the user directory algorithm is most like LBL's session directory tool, `sd`[36], and its follow-on implementation, `sdr`[38]. The distributed announcement protocol was inspired by `sd`'s session announcement technique[37]. However, our user registry tracks users, whereas the session registry tracks sessions. Thus the user registry has an even more critical scaling problem to solve. User location information has longevity, whereas session information is transient. User announcements may be multi-homed, either temporarily or more permanently, but session announcements only ever emanate from a single source. In other words, user information may be long-lived but has potential to be dynamic, whereas session information is static. User announcements are sent at multiple scopes, whereas session announcements are distributed at one scope; the session announcement scope is typically an estimate of where the furthest interested party will be located. Session announcements offer an implicit-join technique to initiate sessions, however these sessions suffer from the same problem as explicit-invitation sessions in that it is difficult to properly scope group membership. Consequently, `sd` and `sdr` often overdistribute session announcements, and thus the associated session data flows.

A useful modification to the original session directory is the idea that the specification of session descriptions[13] has been separated from the specification of the session announcement protocol[37]. We would like to devise an equivalent separation of description from announcement. This would provide a universal description of users that could be used by a variety of network appliations, and that could be embedded in user announcements with ease.

7.3 Directory Hierarchies

A fully distributed, multicast solution represents one end of the directory archicture spectrum. However, there are many related efforts that take more centralized or hierarchical approaches. Utilities such as `finger` or `zephyr` are centralized services that are machine or site specific. Although they provide up-to-date information, they both are strictly localized services. As such, they are not suited for collaborative rendezvous in the general Internet.

The `whois` service is transaction-based and centralized, but does not scale[40]. Its successor `whois++` provides a hierarchy of servers that support structured queries and an indexing capability. Although the client-server protocol has been outlined, the server's abilities to gather information are intentionally left unspecified[39]. The mechanisms behind the user directory architecture are well suited to be used toward implementing `whois++` servers.

The Domain Name Service (DNS) provides a hierarchical architecture that allows mapping machine names into machine addresses, bridging the machine-human interface problem [20]. In recent years, the DNS has been extended to store information records that pertain to additional network resources, such as mail server addresses which could be used as a first-stage mechanism for locating users (e.g., `schooler@cs.caltech.edu`) [26]. Although the DNS has not traditionally been used for user location, it is our intuition that its hierarchical framework would combine well with multicast to improve scaling properties.

7.4 Bootstrapping

It is often difficult to initialize distributed services because of complicated configuration procedures. A common solution to this bootstrapping problem is to use a shared address that binds related resources together. The Address Resolution Protocol (ARP) uses this technique in the local area[31]. It relies on a shared broadcast address to resolve Internet addresses into link-level addresses.

For the wide area, broadcast communication is prohibitively wasteful. Instead, multicast has been used to more efficiently supply a shared address. This is the approach used by the user directory to find and to communicate with peer user directories. A similar technique is used by the service location protocol, a parallel effort which proposes a general purpose protocol that allows users in the local area to find common services, such as printers or other resources[41]. Like the user directory protocol, there is a common multicast address shared by all server location servers. A client queries a server for the location of one of these common services. Each type of resource is in turn addressable via a multicast address. The client can contact the resource directly if the multicast address for the service is known. The intent however is for the service location server to provide the mapping between well-known services and the possibly dynamic addresses assigned to them in different administrative locations, behaving very much like the RPC portmapper[42]. Again, the user directory could be integrated as one component in the service location architecture. However, the service location protocol is not meant for the wide area.

An alternate technology for service location is called *anycasting*. Like multicast, resources subscribing to an anycast address are bound together in some sense. A query to an anycast address aims to locate a host supporting a particular service. If several servers support the service, the requestor doesn't care which server responds. The semantics of the messaging are that the request is relayed via best-effort delivery to at least one, and preferably only one, of the servers. The anycast capability is implemented at the level of routers in the network, and makes special use of unicast addressing, but is distinctly not multicasting.

The value of the service location protocol and anycasting is that they allow processes to bootstrap by knowing a minimal amount in order to configure or intialize themselves on startup. The same holds for the user directory service.

7.5 Expanding-Ring Search

The `traceroute` utility employs scoped unicast messaging to obtain the route used by messages between a particular source and destination. The scope field is incrementally increased in order for the requestor to learn the communication path taken to the remote machine, a technique known as *expanding-ring search*. A similar technique has been used for multicast path discovery by the `mtrace` utility, though the trace is accomplished by incrementally walking up the multicast tree from destination to source. The technique used by `traceroute` and `mtrace` does not announce a resource but rather finds a resource, in these cases the remote machine. In contrast, the user directory algorithm uses expanding-ring search to announce resources.

Chapter 8

Status and Future Directions

A prototype multicast user directory service has been integrated with the `mmcc` session orchestration tool[1]. It supports several levels of scoping and adaptivity. Presently, we are working to solve many of the scaling and configurability issues before wider public release of the registry feature. We have begun more rigorous analysis of how to parameterize the tool for efficient operation as the number of users grows very large. And, we recognize the need to identify the scenarios under which hybrid architectures need to be considered, in particular hierarchical multicast architectures that support querying.

The user directory algorithm, like other soft-state algorithms, is particularly appealing because of its inherent simplicity. Its ability to detect change in the network accommodates application-level mobility (migration) and its use of multicast makes it highly efficient for multiway information distribution. The scoping information collected by the registry service subsequently can be used to derive the proper scope for groups of users, leading to a better bound on data distribution and better multicast address re-use in other parts of the network.

Finally, as part of an emerging collaboration architecture, a multicast user registry facilitates boot-strapping among a variety of communities. It is likely that the proposed user directory solution is useful for any collaborative tool that relies on explicit invitation for users in the wide-area, especially

[1]See the directory `ftp://ftp.isi.edu/confctrl/mmcc` for the latest software release information.

if the user community grows large. Additionally, this architecture may be used to provide location services for a variety of resources; users, databases, hardware in the network (such as transcoders or routers), replicated caches, etc. Thus, we believe there is merit in offering a range of address location services, whether to locate users or any other resources that might facilitate group rendezvous, such as public session addresses or in-the-network services (multimedia translators, mixers, reflectors).

Chapter 9

Acknowledgment

These ideas have benefited greatly from discussions with Eric Bax, Steve Casner, Mani Chandy, Deborah Estrin, Bob Felderman, Rohit Khare, Rajit Manohar, Philippe Mauran, Berna Massingill, Steve McCanne, Mika Nyström, Adam Rifkin, Paul Sivilotti, and John Thornley. Eve Schooler was supported in part by NSF grant CCR-9120008 from the Center for Research in Parallel Computation, by grant AFOSR-91-0070 from the Air Force Office of Scientific Research, ARPA grant DABT63-1-0001 as a collaborator with USC's Information Sciences Institute, and by a grant from the AAUW Educational Foundation. The views and conclusions in this document should not be interpreted as representing the official policies, either expressed or implied, of the AAUW, AFOSR, ARPA, NSF, or the U.S. government.

Bibliography

[1] S.R. Ahuja and J.R. Ensor, "Coordination and Control of Multimedia Conferencing", *IEEE Comm.*, Vol. 20, No. 5, pp. 33-43, May 1992.

[2] Arango et al., "The Touring Machine System", *Comm. of the ACM*, Vol. 36, pp. 68-77, Jan 1993.

[3] N.S. Borenstein, "Computational Mail as Network Infrastructure for Computer-Supported Cooperative Work", *Proc. ACM Conf. on CSCW*, Toronto, Canada, pp. 67-73, 1992.

[4] S. Casner, S. Deering, "First IETF Internet Audiocast", *ACM Sigcomm Comp. Comm. Review*, Vol. 22, No. 3, pp. 92-97, July 1992.

[5] M. Chandy, J. Misra, "How Processes Learn", *Distributed Computing*, Vol. 1, No. 1, pp.40-52, 1986.

[6] S. Floyd, V. Jacobson, S. McCanne, L. Zhang, C.-G. Liu, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", *ACM Sigcomm Comp. Comm. Review*, Vol. 25, No. 4, pp. 342-356, Aug 1995.

[7] S. Floyd, V. Jacobson, "The Synchronization of Periodic Routing Messages", *ACM Sigcomm Comp. Comm. Review*, Vol. 23, No. 4, pp. 33-44, Oct 1993.

[8] T.J. Frivold, R.E. Lang, M.W. Fong, "Extending WWW for Synchronous Collaboration", *Proc. 2nd International WWW Conference '94*, Oct 1994.

[9] R. Fredericks, "Experiences with real-time software video compression", Xerox PARC, July 1994.

[10] S. Deering, "Host Extensions for IP Multicasting", RFC 1054, Stanford University, Stanford, CA, 1988.

[11] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, L. Wei, "An Architecture for Wide-Area Multicast Routing", *Proc. ACM Conf. SIGCOMM*, pp.126-135, Aug 1994.

[12] M. Handley, E. Schooler, "Session Invitation Protocol (SIP)", IETF MMusic Working Group, Internet Draft, Feb 1996.

[13] M. Handley, V. Jacobson, "SDP: Session Description Protocol", IETF MMusic Working Group, Internet Draft, July 1995.

[14] M. Handley, I. Wakeman, J. Crowcroft, "CCCP: Conference Control Channel Protocol: A Scalable Base for Building Conference Control Applications", *Proc. ACM Conf. SIGCOMM*, Aug 1995.

[15] R. Want, A. Hopper, V. Falcao, J. Gibbons, "The Active Badge Location System", *ACM Transactions on Information Systems*, Vol. 10, No. 1, pp.91-102, Jan 1992.

[16] A. Harter, A. Hopper, "A Distributed Location System for the Active Office", *IEEE Network*, Vol. 8, No. 1, Jan 1994.

[17] V. Jacobson, "Multimedia conferencing on the Internet", ACM SIGCOMM '94 Tutorial, London, UK, Aug 1994.

[18] N. Mates, M. Nyström, E.M. Schooler, "The Web meets MOOs, IRC, and the MBone", Computer Science Dept., California Institute of Technology, June 1995.

[19] S. McCanne, V. Jacobson, "vic: A Flexible Framework for Packet Video", *Proceedings ACM Multimedia*, pp. 511-522, Nov 1995.

[20] P.V. Mockapetris, K. J. Dunlap, "Development of the Domain Name System", *SIGCOMM Symposium on Communications Architectures and Protocols*, pp.123-133, Stanford, CA, Aug 1988.

[21] E.M. Schooler, "Case Study: Multimedia Conference Control in a Packet-switched Teleconferencing System," *J. of Internetworking: Research and Experience*, Vol. 4, No. 2, pp. 99-120, June 1993.

[22] E.M. Schooler, "Connection Control Protocol: Specification", Technical Report, USC/Information Sciences Institute, Marina del Rey, CA, 1991.

[23] H. Schulzrinne, "Personal Mobility for Multimedia Services in the Internet", *Proc. European Workshop on Interactive Distributed Multimedia Systems and Services*, Berlin, Germany, Mar 1996.

[24] H. Schulzrinne, S. Casner, R. Fredericks, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", IETF AVT Working Group, Internet Draft, Mar 1995.

[25] H. Schulzrinne, "Voice Communication Across the Internet: A Network Voice Terminal", Dept. of Elec. and Comp. Eng., Dept of C.S., Univ. of Mass., Amherst, MA, 1992.

[26] R. Ullman, P. Mockapetris, L. Mamakos, C. Everhart, "New DNS RR definitions", RFC 1183, IETF, Oct 1990.

[27] H.M. Vin, D.C. Swinehart, P.T. Zellweger, P.V. Rangan, "Multimedia Conferencing in the Etherphone Environment", *IEEE Computer*, Vol. 24, No. 10, pp. 69-79, Oct 1991.

[28] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, "RSVP: A New Resource ReSerVation Protocol", *IEEE Net*, 1993.

[29] T. Berners-Lee, R. Fielding, H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0.", RFC 1945, IETF, May 1996.

[30] N. Borenstein, N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, IETF, Sept 1993.

[31] Plummer, D., "An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48-bit Ethernet Addresses for Transmission on Ethernet Hardware", RFC-826, Symbolics, November 1982.

[32] S. Casner, K. Seo, W. Edmond, C. Topolcic, "N-Way Conferencing with packet Video", *Proceedings Third International Workshop on Packet Video*, Morristown, New Jersey, Mar 1990; also available as Technical Report ISI/RS-90-252, USC/Information Sciences Institute, Marina del Rey, CA, April 1990.

[33] V.J. Hardman, M.A. Sasse, M. Handley, A. Watson, "Reliable Audio for Use over the Internet", *Proceedings INET'95*, Internet Society, Hawaii, July 1995.

[34] M.J. Fischer, N.A. Lynch, and M.S. Paterson, “Impossibility of Distributed Consensus with One Faulty Process”, *Journal of the ACM*, 32(2):374–382, April 1985.

[35] V., Jacobson, S. McCanne, “Visual Audio Tool”, Lawrence Berkeley Laboratory, software on-line, available at `ftp://ftp.ee.lbl.gov/conferencing/vat`.

[36] V., Jacobson, S. McCanne, “Session Directory”, Lawrence Berkeley Laboratory, software on-line, available at `ftp://ftp.ee.lbl.gov/conferencing/sd`.

[37] M. Handley, “SAP: Session Announcement Protocol (Version 1)”, draft 0.2, IETF MMusic Working Group, Internet Draft, June 1996.

[38] M. Handley, “sdr”, software on-line, available at `ftp://cs.ucl.ac.uk/mice/sdr/`.

[39] C. Weider, J. Fullton, S. Spero, “Architecture of the Whois++ Index Service”, IETF, RFC 1913, Feb 1996.

[40] K. Harrenstein, M. Stahl, E. Feinler, “NICNAME/WHOIS”, RFC 954, SRI, Oct 1985.

[41] Veizades, J., Kaplan, S., Guttman, E., “Service Location Protocol”, IETF Service Location Working Group, Internet Draft, June 1996.

[42] R. Srinivasan, “RPC: Remote Procedure Call Protocol Specification Version 2”, RFC 1831, Sun Microsystems (Aug 1995).