A Signaling System Using Lightweight Call Sessions

Helen J. Wang, Anthony D. Joseph, Randy H. Katz
Computer Science Department
University of California at Berkeley, CA USA
{helenjw, adj, randy}@cs.berkeley.edu

Abstract—**Because of the emergence of heterogeneous access devices and diverse wired and wireless networks, and a substantial lack of support for integrating these networks, we are building a communication network and a service infrastructure that provides integrated telephony and data services across these networks. We generalize the basic call service to support communication between two or more call parties using any number of devices through any media. Call setup has already been addressed by protocols such as Session Initiation Protocol (SIP) [1]), so we focus on a missing component: scalable and fault tolerant call session maintenance and control after call setup. In this paper, we present a new signaling protocol that offers scalability, high availability, robustness, and flexibility in creating new call processing services. The protocol is compatible with existing call setup protocols, but provides lightweight call session management using a completely decentralized, soft state group control protocol. We show that this approach simplifies the implementation of the basic call service, including multi-device calling and service handoff between diverse access networks. The design of our protocol follows the principle of separation of control (signaling information) from data. The signaling protocol has been implemented in ICEBERG, an IP-based core network testbed with access to heterogeneous networks.**

Keywords— **Lightweight call session, signaling system, soft state, basic call service, service handoff, data path, scalability, availability, fault tolerance, composability**

I. INTRODUCTION

One of the principal goals of the ICEBERG Project [7][8] is to provide support for integrating services from heterogeneous networks (e.g., Public Switched Telephone Network, digital cellular networks, pager networks, IP-based networks, etc.). Since such service-level support is currently lacking in many environments, we aim to enable *Potentially Any Network Services* (PANS): the capability to access services from any device via any network (such as accessing e-mails via a cellular phone). We use the Internet as the rendezvous point. Diverse networks are treated as access networks, and the Internet (or other IP-based network) is the core switch that routes information (e.g., telephone calls, or data), integrates diverse networks and devices, and provides Internet- and telephony-like services. A key advantage of using an IP-based core is the easy, rapid, and low cost deployment of network-based services, as evidenced by the explosive growth of web services and wide adoption of E-commerce. ICEBERG leverages this characteristic to provide creative and customizable integrated communication services.

In the context of the ICEBERG project, we have designed and implemented a signaling system that is a collection of communicating network elements handling call setup, routing, and call control. This subsystem of ICEBERG is the focus of our paper. We identify the following requirements and challenges in designing the signaling system:

- **Scalability**: The signaling protocol must scale to a large user base in the wide-area, as well as a large number of call parties within a call.
- **Fault tolerance**: The signaling protocol must tolerate failures of signaling components.
- **Easy service creation**: The signaling protocol must allow easy creation of new call processing features.
- **Network/device independence**: The design of the signaling protocol must be independent of particular access networks and devices. Independence will make it easy to add new networks and new access devices to ICEBERG and will simplify their participation in the signaling protocol.
- **Performance**: Call setup latency and other call control latency must be kept low.

As call setup has already been addressed by protocols such as the Session Initiation Protocol (SIP) [1]), we address the issues associated with maintaining a call session that scales to a large number of call parties, and with adapting to call session dynamics such as protocol agent failure and recovery or call session membership changes. These issues have not been addressed in the literature.

Our signaling protocol is based upon the idea of a *lightweight call session*. Light Weight Sessions (LWS) [4] [5] are "thin" application level protocols built on top of IP Multicast and its powerful group concept emphasizing a lightweight, loosely-coupled, decentralized communication model. We apply this concept here: in a call session, protocol agents that perform signaling on behalf of communicating devices form a call setup and control communication group. A call session is loosely coupled because each participant does not need to know the identities and locations of other participants in the call session, but only needs to know the group identity of the session. The group identity serves as a level of indirection, allowing the call session to adapt to membership changes gracefully. The "lightweightness" comes from the use of a soft state signaling protocol during the call session, where protocol agents maintain periodically refreshed state. Lightweight call sessions provide a robust distributed system, because with soft state, error recovery is designed into the protocol and does not need to be treated as a special case [9]. Maintaining dynamic call sessions, and carrying out call control functions in a fault tolerant fashion are the main contributions of this paper.

Another benefit of this approach is that multi-device communication becomes a first-class service, rather than a service that requires special implementation support from the signaling protocol. Services requiring endpoint changes, such as call forwarding, can be implemented using the normal multi-device call operations, like inviting a new device into the call session and disconnecting a device from the call. A novel service, called *service handoff*, allows users to switch among their communication devices in the middle of a call. Service handoff is a special case of multi-device communication. In the middle of a phone call, a participant can cause another communication device (instead of a new user) to join the conversation. Then the participant terminates the call on the current device (forcing the current device to leave the conversation), completing the service handoff. Service handoff enables *service mobility*, the ability to retain access to services as users move from one network to another.

Our lightweight call session management solves only the control signaling portion of a communication session. Establishing the data transport (or the data path) between heterogeneous devices is another challenge. Our approach, like that of many other signaling systems, is to separate call control from call data. In particular, we isolate data path creation and manipulation from the rest of the signaling system. We view the data path as a sequence of data transformation operators (e.g., an operator for converting from GSM-encoded audio to PCM-encoded audio, or a text-to-speech operator) and transport connectors (e.g., unicast, directed broadcast, or multicast connectors)[14]. The data path creation process is the process of composing these operators and connectors, and therefore is the key enabler and simplifier for any-to-any communication. While our signaling uses group communication, data paths may rely upon pairwise communications between devices or group communication among a number of devices.

The rest of the paper is organized as follows. First, we provide an ICEBERG system overview in Section II. The signaling system design is discussed in Section III. In Section IV, we show how the protocol simplifies the implementation of multi-device communications. Then we present our implementation in Section V, and performance results in Section VI. We give an in-depth discussion on our relationship with SIP and a comparison with H.323 in Section VII. We discuss our future work in Section VIII, and conclude in Section IX.

II. ICEBERG Architecture Overview

Our architecture is designed for service providers who offer customizable integrated communication services on top of the Internet. We call these service providers *ICEBERG Service Providers* (iSP). Similar to the way the Internet Service Providers (ISP) provide Internet services through the use of Points of Presence (PoPs) at different geographic locations[1], iSPs consist of ICEBERG Points of Presence (iPOPs). Each iPOP consists of ICEBERG Access Points (IAPs) which bridge between the Internet and the access networks (e.g., PSTN, GSM or pager networks), Call Agents (CAs) which perform call signaling, a Data Path Factory which establishes data flow, a Preference Registry (PR) which stores and processes user preferences, a Person Activity Tracker (PAT) which tracks user activities, and a Name Mapping Service which resolves user names in different networks. The ICEBERG network can be viewed as an overlay network of iPOPs on top of the Internet. iPOPs must scale to a large population and a large number of calls, be highly available and be resilient to failures. This leads us to employ a cluster computing platform to carry out most of the iPOP functions. This section describes in detail the cluster computing platform and each of the iPOP components.

A. Leverage Cluster Computing Platforms

Each iPOP requires processing scalability to a large call volume, 24×7 availability through fault masking, and cost-effectiveness. Clusters of commodity PCs interconnected by a high-speed System Area Network (SAN), acting as a single large-scale computer [12] (assuming no network partitions within a cluster), are especially well-suited to meeting these challenges[13]. Cluster computing platforms like the Ninja Base [15] and Active Service Platform (AS1) [9] provide an easy service development environment for service developers, and mask away from them the problems of cluster management which include load-balancing, availability, and failure management. Although both intend to be a general cluster computing platform that can support any cluster-based services, Ninja Base targets long running services with infinite lifetime (such as web servers) while AS1 targets services with finite session lifetimes (like video-conferencing). For ICEBERG, we have both kinds of services: a call served by a Call Agent is a session-based service; the Preference Registry, Name Mapping Service, Data Path Factory and Person Activity Tracker are long running servers. We describe briefly the underlying mechanism of both platforms and how we leverage them.

In a Ninja *Base*, each node houses an execution environment into which mobile code can be pushed. Services have many instances within the cluster, but clients are shielded from this by using a service *Redirector Stub* to interact with the cluster. The Redirector Stub for a service is dynamically generated at runtime and contains the embedded logic to select from a set of nodes within the cluster. Within the Base, all nodes maintain a replicated registry of all service instances; each node periodically sends out a multicast beacon carrying its list of local services to all other nodes in the Base, and listens for the multicast beacons from other nodes. The Base also maintains persistent service states. As a result, the failure of a service instance on one node automatically triggers another service instance to take over for the failed one. To the outside world, a Base acts like a non-distributed, robust, highly-available, high-performance service.

AS1 differs from the Ninja Base approach in that AS1 service clients send keep-alive service requests instead of using redirector stubs. A timeout on service requests indicates the end of the service lifetime. Service requests contain the current session state, and service agents are entirely soft state.

We implemented iPOPs on Ninja Bases to which we added support for the keep-alive service request from AS1. For the rest of the paper, iPOP refers to an ICEBERG-capable Ninja Base.

B. ICEBERG Access Point

An ICEBERG Access Point (IAP) is a gateway that interconnects a wired or wireless access network with the rest of the ICEBERG network. It consists of both hardware and software for transcoding between the signaling protocols and the data formats used by the access network and those used by the ICEBERG network.

Running the IAP on a cluster platform provides few benefits as it includes a network-specific gateway, which may contain hard state for each call (a function of the signaling protocol used by the access networks). This state cannot be replicated, thus the failure of an IAP causes call drops (as a comparison, the failure of a Base Station Controller in the GSM network would cause all calls handled by it to be dropped). Thus, it is the gateway's responsibility to provide the appropriate level of fault tolerance. Because of this, although an IAP belongs to an ICEBERG Point

[1] ISPs generally locate PoPs such that users can make a local call and gain Internet access.

of Presence conceptually, it is considered to be a component running outside the iPOP Base.

C. Call Agent

A *Call Agent* is a service agent that performs call setup and control for a caller's device. There is one Call Agent per device per caller. An iPOP spawns a Call Agent on a node in the cluster whenever a call request (for either an inbound or outbound call) arrives at the iPOP. The Call Agent is kept alive by the periodic call requests from its client (the device or its IAP). The Call Agent is terminated when the call requests cease after the device hangs up.

We choose not to build Call Agents as part of the IAPs for two reasons. First, call setup and control within the ICEBERG network is the same for all access networks. It is not desirable to duplicate this functionality in the IAPs of each access network. Second, since IAP failures cause call drops, we want to minimize the amount of logic in IAPs, and therefore make them less failure-prone. The benefit of this decoupling is that IAPs are the only access network dependent components, which greatly simplifies the expansion of ICEBERG to new networks and devices.

D. Data Path Factory

A *data path* is an abstraction of a data flow – a set of *transcoding operators* strung together using *connectors*[11]. An operator is a unit of computation on some data, and a connector is an abstraction of the Application Data Unit (ADU) transport mechanism between two operators (e.g., RTP connector, UDP connector, etc.). For instance, a series of codec operators followed by a speech-to-text operator is a path. The concept of *data path* is powerful for enabling communication among heterogeneous end-devices because of its flexible service composability.

In our architecture, Call Agents make requests to a *Data Path Factory* to create data paths for communication between the end devices. The Call Agents provide the Data Path Factory with endpoint information, such as the endpoint's address (e.g., IP address and port number for a desktop phone, the phone number and cellular gateway address for a cell phone) and transport protocol (e.g., RTP). The use of Data Path Factory isolates the data path creation and instantiation process from the rest of the system, and cleanly separates data from control.

E. Preference Registry

ICEBERG provides customizable communication using a *Preference Registry*, a distributed service that stores and processes user preferences. The registry takes as input the caller ID, callee ID, time of day, and dynamic user information (e.g., current location or current user behavior), and returns the callee's preferred endpoint (e.g., cell-phone number or e-mail address). During call setup, the Preference registry is queried for a callee's preferred endpoint. Users convey their preferences (such as when they want to be called on what device by whom) using a tool that generates scripts (e.g., Perl or Tcl) and stores them in a Preference Registry.

F. Person Activity Tracker

The *Person Activity Tracker* (PAT) tracks a user's current activity (e.g., "Alice is on her cell phone with Bob, who is a VIP") and other dynamic events (such as a user's current location, or the call status of a device – busy, on-hold, etc.). The PAT allows users to control privacy policies, such as which information is tracked and to whom the information can be released. The information maintained in the PAT is used by the preference registry as an additional input and hints in processing a user's profile. This information is stored as soft state. Some default user behavior is assumed if the PAT does not provide any information.

G. Name Mapping Service

The idea of *Personal Mobility* is to make the person[2], and not the communication device, be the communication endpoint[3]. The concept is becoming more important as people acquire more and more end devices. By using a single identity for an individual, ICEBERG provides a level of indirection to the desired communication endpoint. Personal Mobility is a key mechanism for managing communication across heterogeneous devices from diverse networks.

In ICEBERG, we associate each user with an ICEBERG Unique ID (iUID). The structure of iUIDs is still an open problem in ICEBERG. For now, an iUID takes the form of an e-mail address[4]. The *Name Mapping Service* maintains the mapping between the users' various communication endpoints and their iUIDs.

III. THE ICEBERG SIGNALING SYSTEM

A signaling system is a collection of network elements that use a communication protocol to effect call setup, routing, and control. In this section, we present the ICEBERG signaling system.

Before describing our signaling system, we define the basic call services to be supported.

The *basic call service* (or simply a "call") in ICEBERG refers to the communication between two or more call parties[5] using any number of communication devices via any kind of media (e.g., voice, text or video). Each call party can use either one device or multiple coordinated devices[6] (e.g., using an audio tool and a video tool together for a call). E-mail, telephone calls, and video conferencing are all examples of the basic call service. The basic call service provided by the ICEBERG architecture is not a special service that must be implemented as an extension to the ICEBERG signaling. We call the basic set of ICEBERG services *first-class services*. Current telecommunication systems are designed to support the basic call service of two parties using two homogeneous devices; additional services incur additional expenses to users. While many perceive two-party calls as the common case, it is our belief that not only user inclination, but also infrastructure support determine the common case communication model. In ICEBERG, we designed our architecture to allow multi-device communication and communication endpoint changes to become common scenarios. We

[2] It is possible to name roles (e.g., department chair, graduate admission) as well as individuals.

[3] The concept originally comes from Personal Communication Services [18] and has also been identified by the Mobile People project [16].

[4] The problem with iUIDs based upon e-mail addresses is that a person may belong to several administrative domains, already has multiple e-mail addresses, and may change addresses.

[5] A call party can be either a human user or an automated service such as voice mail.

[6] If a call party is using multiple devices in an uncoordinated way, the call party is merely making multiple calls at the same time.

also demonstrate that, in the Internet environment, it is simple to support this very general communication form as the basic service.

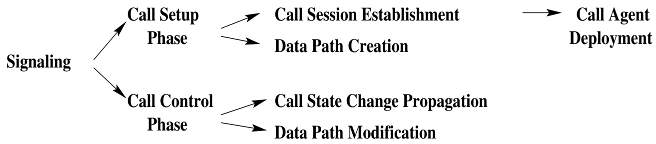

Fig. 1. Signaling Phases and Function Decomposition

Figure 1 shows the different phases of ICEBERG signaling during the lifetime of a call, and the corresponding signaling function decomposition at each phase. The *call setup phase* instantiates and installs protocol agents to service a call (Call Session Establishment), and establishes data flow among the communicating devices (Data Path Creation). A collection of protocol agents (Call Agents), their respective protocol and protocol instances form a *call session*. The *call control phase* changes the behavior of an ongoing call by altering and propagating the call states in a call session (Call State Change Propagation) and modifying the data path in a corresponding manner (Data Path Modification).

A call session can be highly dynamic. New Call Agents may join the call session (possibly simultaneously) to serve new participating devices; existing Call Agents may leave the call session because the devices they are serving hang up; and Call Agents themselves may be restarted upon failure at different locations with new identities. Thus, the signaling protocol must adapt to call session membership changes and ensure the reliable propagation of call state changes to all involved Call Agents. These requirements are best satisfied by using a shared communication channel to which all participating Call Agents subscribe. This channel hides the identities and locations of the Call Agents, and is responsible for delivering call state change messages to all involved Call Agents.

In ICEBERG, a *call session* represents this shared group communication channel. A Call Agent does not need to know the identity or location of all other Call Agents in the session but only the call session address. Not only does this improve the ability of the system to adapt to new situations, but it also yields scalability to a large number of call participants since a call session relieves the call agents from explicitly tracking session membership of a highly dynamic session by leveraging the level of indirection and rendezvous mechanisms provided by the network layer.

Another requirement is that new Call Agents in a session must be able to obtain the up-to-date session state. New participating Call Agents do not have the session membership information, and therefore do not know where to obtain the complete session state. One approach is to use a centralized session state manager that maintains the current session state, but this approach has several disadvantages. First, a single state manager is a single point of failure in the system. Second, it creates a bottleneck in the control path since all state updates must flow through it. Finally since the state of the system is managed by a single entity, that entity must also implement rigorous error detection and recovery which incurs protocol complexity and implementation overhead. Our approach to this problem is to maintain the call state as soft state [31]. Soft state is operationally defined as state that is periodically refreshed by update messages. Protocol actions are triggered by new messages and timer expirations. Because session state is sent to the call session periodically (see Section III-B for details), newcomers to a session will automatically obtain the current session state through these update messages.

The *call session* abstraction of the shared group communication channel, combined with a soft state protocol for call state management, maps perfectly onto the *Light Weight Session* (LWS) architecture [4]. LWS is described as a lightweight (soft state), loosely-coupled, decentralized (anonymous Call Agents) communication model. The concept of a *call session* is analogous to the *group* concept in IP multicast [19]. We use IP multicast groups as the mechanism for our call session implementation and identify call sessions by multicast addresses.

Using one multicast address per call session raises two scalability concerns: multicast address scarcity and scaling routing state to a large number of small multicast groups. The first problem, due to the flat IPv4 multicast address space, will vanish with the deployment of IPv6 [6], MASC [33], or new IP multicast models like Simple Multicast [23] and Express Multicast [22]. For the second problem, the multicast research community has been actively seeking solutions. Multicast state aggregation [27] reduces router forwarding state by combining entries with matching outgoing interface sets. More aggressive state reduction can be achieved by leaky aggregation [28] allowing low bandwidth sessions to flow down links without receivers. Specifically targeting large numbers of small groups, EndSystem Multicast [24] builds an overlay control structure among group members at end hosts in a way that is robust to dynamic membership changes and network conditions. Similarly, Distributed Core Multicast (DCM) [26] uses an overlay of distributed core routers at the edge of the backbone maintaining multicast forwarding states, eliminating state information in the backbone routers. REUNITE [25] uses a unicast address and port number to identify a multicast group (eliminating the multicast address scarcity problem), and it uses unicast between branching points in the multicast distribution tree, eliminating the forwarding state in the non-branching point routers.

These research efforts convinced us that the group communication service model (communication among anonymous group members) provided by IP multicast is viable for deployment either in the improved form of current IP multicast or through new multicast service infrastructures as forms of application layer overlay networks.

In the next few sections, we walk through a call session establishment scenario (Section III-A), present our call session maintenance and control protocol (Section III-B), and describe our experiences with Data Path Manipulations (Section III-C). Finally, we discuss fault detection and recovery.

A. Call Session Establishment

In this section, we first explain how we locate ICEBERG service components, such as ICEBERG Access Points (IAPs), ICEBERG Point of Presences (iPOPs), Preference Registries and Name Mapping Servers. Then, we walk through the steps that are required to establish a new call session in Section III-A.2.

A.1 Service Location

ICEBERG service components are located primarily using the *Service Discovery Service* (SDS) [20] which is a directory-style, wide-area service location service. SDS provides a scalable and fault-tolerant infrastructure for secure service location. ICEBERG service components, such as IAPs and iPOPs, advertise their services to the SDS. These services can be searched and queried by other components based upon attributes such as location, service type, and service-specific values. SDS also provides wide-area support by having SDS servers arrange themselves into a multi-level hierarchy, where summary service information is propagated to parents and service queries are automatically routed between the servers. Given these capabilities, SDS satisfies ICEBERG's requirements for service location. In ICEBERG, locating an IAP from devices such as cell phones or 2-way pagers requires special support, as these devices are not capable of directly accessing SDS servers. In such cases, a bootstrap IAP is used as an intermediary. A cell phone or 2-way pager first dials a sequence of digits to access the bootstrap IAP which then uses the SDS to locate an appropriate[7] IAP for handling other requests from the device.

A.2 Basic Signaling in Call Session Establishment

Call session establishment creates *Call Agents* to service the communication devices that are involved in a call. Together, these call agents form a call session. The communication in this phase is pairwise and can use SIP's three phase handshake session initiation messaging (i.e., INVITE, ACK, and ACK) [1]. Group communication takes place after the session is established as explained in Section III-B. The call session establishment described here, as well as SIP, does not consider component failures. In [29], we propose a fault tolerant call setup protocol.

Figure 2 depicts a scenario where Alice calls Bob, then Bob invites Carol to join in the conversation. We explain the call session establishment steps below.

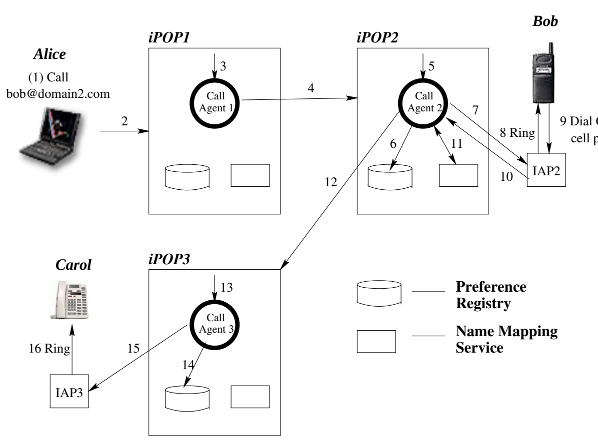

Fig. 2. Creation of a lightweight Session

1. On her wireless LAN connected laptop, Alice uses an IP phone program to call Bob using his iUID, "bob@domain2.com".
2. Since the laptop is part of the Internet, the phone program is a simple IAP that speaks the ICEBERG signaling protocol, and the IAP does not need to have access-network specific protocol translation functions. The phone program (or the IAP) first obtains a globally unique multicast address identifying the call session to be created (through the allocation mechanism described in [33]). It then uses SDS to locate an iPOP (iPOP1) for Alice, then sends the call request to it.
3. iPOP1 performs admission control[8] on the call request. If iPOP1 admits the call, it instantiates a Call Agent (Call Agent 1) to service Alice's laptop for signaling.
4. Call Agent 1 uses the SDS to locate an iPOP for Bob (iPOP2), and then issues a receive-call request to iPOP2 along with the multicast address for the call session.
5. iPOP2 performs the admission control. If it admits the call, then the iPOP deploys a Call Agent (Call Agent 2).
6. Call Agent 2 uses the SDS to locate the Preference Registry that contains Bob's preferences, looks up Bob's call receiving preferences in the Preference Registry, and obtains Bob's preferred communication endpoint for this call. In this example, it is his cell phone.
7. Call Agent 2 uses the SDS to locate an IAP with a gateway between the cellular network and the ICEBERG network (i.e., IAP2). Then, Call Agent 2 asks the IAP2 to ring Bob's cell phone.
8. IAP2 rings Bob's cell phone and Bob answers the call. Next, Call Agent 2 requests the Data Path Factory on iPOP2 (not shown in the figure) to create the data path between Alice's laptop and Bob's cell phone. Alice and Bob are in conversation from this point on.
9. After some time, Bob invites Carol to join the conversation by dialing "*i", indicating the invitation of another call party, and then dialing her cell phone number.
10. IAP2 understands "*i" as an invitation from the cell phone during a call, and relays the invitation to Call Agent 2.
11. After using the SDS to locate a Name Mapping Server, Call Agent 2 looks up the iUID from the dialed phone number passed from the IAP2, and obtains "carol@domain3.com".
12. Call Agent 2 uses the SDS to locate an iPOP for Carol (iPOP3), then issues a receive-call request to iPOP3 along with the multicast address for the call session.

Steps 13-16 are the same as steps 5-8, except that they involve setting up the call from Bob to Carol, instead of from Alice to Bob.

In Figure 2, the highlighted Call Agents form a call session, and they communicate over a multicast channel (see Section III-B for more information). The control information exchanged within the call session is used to create the data paths between the communication devices (as explained in Section III-C).

B. Call Control in a Lightweight Call Session

Followed by the call setup phase is the call control phase during which the call parties may hang up or interact with their devices in the middle of the call (e.g., invite a new call party, switch to a new device, or perform call forwarding, etc.). Maintaining dynamic call sessions and carrying out these control

[7] Appropriateness may be based on cost, quality of service, etc.

[8] Admission control is a separate problem from signaling. Any admission control mechanism can be plugged into this signaling process.

functions in a fault tolerant fashion are the main contributions of this paper. In this section, we describe the control protocol for call session maintenance and call control.

Call control involves altering and propagating call states in an established call session, and creating or modifying the data path accordingly. The call state of a call session includes call party identities, the involved communication devices and their status (e.g., busy or on hold), and data path endpoint information for all the data streams in the session (e.g., IP addresses and port numbers of the data stream end points, location of transformation agents, etc.). In this section, we focus on call state information exchange between the Call Agents.

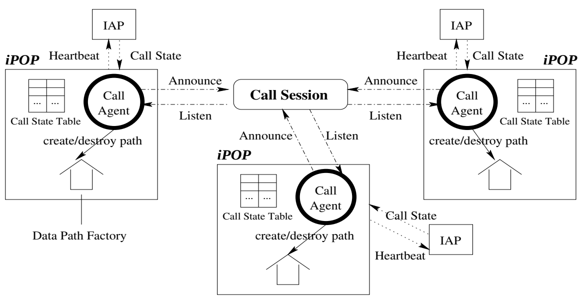

Fig. 3. Signaling in the Call Session

Figure 3 illustrates the soft state protocol for propagating call state changes. Each IAP periodically sends the call request to the iPOP. The call request[9] contains the call state of one endpoint, and serves as a keep-alive heartbeat for the serving Call Agent. The call state is one of the rows in Table I. The call agent, in turn, periodically announces the call state obtained from the call request to the call session. The Call Agent also listens to the multicast channel and receives call states of the other endpoints. Hence, each Call Agent maintains the complete call state of a call session: the entire Table I, each entry of which is uniquely identified by “iUID” and “Device” (the primary key of the table). The reliability of call state propagation is ensured simply by periodic retransmission over the lifetime of the call session. When a Call Agent receives a new call state (i.e., if its key can-

iUID	Device	Status	Path Endpoint Info
alice@domain1.com	laptop	busy	123.3.4.5/9876
bob@domain2.com	cell phone	busy	gw: 10.3.2.2/66 time slot: 5
...	...	...	...

TABLE I
CALL STATE IN A CALL SESSION

not be found in the call state table), it knows that a new endpoint just joined the call session, and this Call Agent needs to establish the data path with the new device through the Data Path Factory (more in Section III-C).

Call state changes are made by the IAPs through periodic call requests with modified call state, and then the modified state is propagated to the call session through the Call Agent’s periodic announcements. When a Call Agent receives a modified call state (e.g.,the data path information has changed), the associated data path will be modified accordingly.

When a device hangs up, its IAP sends a “leave” message to its iPOP. Then its Call Agent forwards the “leave” message to the call session, stops announcing its call states to the call session, instructs the Data Path Factory to tear down the data paths, and finally destroys itself. If either “leave” message is lost, a timeout will occur either on the call request or on the call state, and the same action will take place. When there is only one Call Agent left in the call session, that Call Agent will leave the call session as well since there is no one else to communicate. The Call Agent periodically sends the “hang-up” message to the IAP until no more IAP call requests on this call arrive, then the Call Agent terminates itself. When the IAP receives the “hang-up” message, it hangs up the device, and stops the periodic call request to the iPOP.

The combination of periodicity and the use of multicast is often called the *announce/listen* model in the literature, and is appropriate where *eventual consistency* rather than transactional semantics are sufficient. A desirable property of the announce/listen model is that component failures are tolerated as the normal mode of operation, rather than addressed through a separate recovery procedure [9]: recovery is enabled by simply listening to channel announcements. The announce/listen model initially appeared in IGMP [34], and was further developed and clarified in systems such as the MBone Session Announcement Protocol [35].

Despite the simplicity of the design, this scheme handles simultaneous call state changes at multiple Call Agents gracefully. In addition, no special handling is necessary for multi-device calls. In contrast, traditional telecommunication systems handle call transfer and multi-party call conferencing through the installation of new “trigger points” for these services in the basic call processing state machine, a significantly more difficult and complicated process.

C. Data Path, a Simplifier

In this section, we explain how data paths are created and manipulated.

Data path construction establishes the flow of data between the communication endpoints of different call parties. A path consists of a sequence of operators and connectors (see Section II). The power of the path concept comes from the highly flexible composability of operators and connectors. Ideally, given two endpoint data formats and locations, a path consisting of a sequence of operators with appropriate connectors can be automatically constructed. We do not yet have a full-fledged path creation service. The current ICEBERG Data Path Factory only creates and places predetermined operators at the appropriate locations. An operator description is associated with each operator for its creation (which code to execute), placement (which host in the cluster), and destruction (what to clean up after the operator terminates). Modifying a data path involves changing the operator descriptions, and restarting or modifying the currently executing operators.

Creating paths in the wide area and maintaining them in a fault tolerant fashion are the main challenges for data path creation and maintenance. A centralized data path factory responsi-

[9]These periodic call requests are the same as the service requests in AS1, see Section II-A.

ble for instantiating, executing, and maintaining the paths is not appropriate for the wide area, because when it is out of service (from host crashes or network partitions), it orphans all the paths it maintains (with operators and connectors running on different hosts).

Distributed path creation and maintenance poses difficult questions like who owns a path, and if a path is owned by more than two entities, then who gets to create and destroy the path. Also, our signaling protocol is based on an asynchronous (soft state) anonymous (using multicast address as a level of indirection and the rendezvous) group communication model, which makes the protocol incompatible with synchronous path creation and maintenance between pairs of endpoints. Multiple owners of a single path would need to cooperate synchronously to create and destroy the path, so a path should be owned by only one entity. This is somewhat counter-intuitive since a path describing the communication between two devices does indeed involve two endpoints, and therefore, the path is perceived to be owned by these two endpoints.

Now let us present a different view for this scenario with an additional given information: an intermediate data format (which can coincide with the data format supported by either endpoint). From each endpoint's view, all it needs to do is to construct a data path to send and receive data in the intermediate data format. This essentially decomposes the communication between two devices into two paths with the output data format of one path the same as the input data format of the other, namely the intermediate data format. This way, each endpoint owns its half of the path, and it creates and destroys the data path autonomously with no need to synchronize with the other endpoint. Each half-path can be composed of operators all running on the same iPOP that is serving the endpoint, so it is reasonable to run a single Data Path Factory on each iPOP (where network partitions are rare), responsible for creating half-paths for endpoints serviced by that iPOPs. Like any Ninja service, the Data Path Factory will be highly available and fault tolerant.

Now we address how to determine the intermediate data format. Each iPOP knows what operators and connectors it has and which are running. Therefore, it can export a list of data formats it supports. During the call setup, the caller's iPOP obtains this list from the callee's iPOP. Then it selects a matching data format as the intermediate data format. We currently take the first match, but we are investigating more sophisticated algorithms that select the most cost-effective data format for all call participants.

D. Fault Detection and Recovery

We have addressed the control path and data path. In this section, we discuss the signaling system's fault detection and recovery mechanisms.

The Preference Registry, Name Mapping Service, Person Activity Tracker and Data Path Factory are long running Ninja Servers. The Ninja Base handles their fault detection and recovery through multiple service instances, automatic restart after failures, and client redirector stubs (see Section II-A).

As explained in Section II-B, the IAP does not benefit from running on Ninja Base. An IAP failure causes all calls being served to be dropped. Since IAPs periodically send call requests to iPOPs, IAP failures can be detected by the iPOPs, and are treated in the same manner as the devices hanging up the call. iPOPs also send heartbeats to IAPs for the IAPs to detect the network partitions between them. A non-transient network partition causes IAPs to use alternative iPOPs if available, or terminate the calls being served by the cut-off iPOPs.

A Call Agent lasts for the lifetime of a call, and is kept alive by periodic call requests from the IAP. Failure detection and recovery of the Call Agent is no different from Call Agent instantiation, just as in AS1[9]. The iPOP maintains a table of call request and Call Agent pairs on some nodes in the cluster. The entry for a failed Call Agent will eventually timeout and be removed. An arriving call request at the iPOP with no associated Call Agent causes a new call agent to be spawned to service the call request. Each call request contains the up-to-date session state (e.g., the signaling multicast address for the call session) for a Call Agent to carry out the correct actions. A recovered Call Agent joins the signaling multicast group, and re-builds the call session state (including the additional session state incurred during the Call Agent fault recovery) from the call state announcement of other Call Agents. This way, each call agent of a session is oblivious to the transient failures and recovery of the other Call Agents. Prolonged failures of a Call Agent will cause the device it serves to be cut off from the session.

Prolonged network partitions among iPOPs involved in a call are detected by the call state timeouts, and for a multi-party call, the call may be partitioned into multiple calls. Nonetheless, the partitioned calls will automatically re-integrate when the network partitions are healed.

Data paths can also fail. Data paths are separated from control paths, except at an IAP, since the network-specific gateway interacts with both control and data components of the access network. Thus, the failure of the IAP will jeopardize the data path as well. However, failures (although unlikely) of Call Agents, Name Mapping Servers, and Preference Registries will not affect the data path. The operators in a data path can also fail, but by running the operators on Ninja Bases, such failures can be masked.

IV. Multi-device Calls as Basic Call Services and a Substrate for Service Creation

In today's telecommunications world, the basic call service is a two-party call with homogeneous devices, usually telephones. Multi-party or conference calls are viewed as special services, and a significant amount of signaling is needed in addition to the basic call model. Our signaling system easily supports the multi-device call as a basic call service (see Section III). The only support requirement is that the IAP must be able to detect new call requests from end-devices in the middle of a call, and to relay these requests to the serving iPOP. The multi-device call differs from the conference call in that conference call requires at least three parties while a multi-device call can be made by only two parties. This means that a call party may be using more than one device in the same call session.

The operations that carry out the basic call service are the building blocks for additional services. By having multi-device communications as the basic call service, we have enriched the set of the building blocks. The multi-device call operations are adding a new communication endpoint to the call and removing an existing communication endpoint from the call. Changing a communication endpoint adds the new endpoint, and then removes the existing one. Services that require endpoint changes (such as call forwarding and call transfer) can be implemented easily on top of these *user-level* operations. For example, call forwarding can be implemented by having the forwarder invite the forwardee endpoint and then leave the call. Service handoff is another example, which we now describe in detail.

A. Service Handoff: Enable Service Mobility

Service handoff occurs when users change their communication devices during a call. It enables service mobility, the ability to retain access to services as users switch between networks. In telecommunications, the term "service mobility" usually means "service portability", the ability for diverse networks to share user service profiles and to carry out the same set of services in each network [10]. However, when one crosses the network boundary during an active service, that service is terminated. The user must then restart the service in the new network. In contrast, service mobility provides *seamless* integration of heterogeneous devices from diverse networks, as if they are accessing the same network, achieving the goal of *Potentially Any Network Service* (PANS).

Service handoff can also be viewed as a generalized call transfer service. Instead of call transferring between homogeneous telephones, service handoff allows one to perform call transfer across diverse devices from heterogeneous networks.

Service handoff is enabled by the multi-device call operations. Imagine the following scenario: Alice is on her cell phone. She walks into her office and decides to handoff the call to her laptop IP phone. She does this by first pressing some keys on her cellular phone to indicate the intent and the target of a service handoff (e.g., "*SHIP" – Service Handoff to IP phone). The serving IAP that contains a cellular gateway intercepts the DTMF message and relays it to the serving Call Agent for Alice's cell phone. The Call Agent looks up the end point information (IP phone's IP address and port number in this case) in the Preference Registry with the input of Alice's iUID and endpoint type ("IP"). Then the Call Agent invites the IP phone to join in the conversation, as illustrated in Section III-A (i.e., the operation of adding a communication endpoint to a call[10]). Finally, Alice turns off her cell phone, which informs the Call Agent that was serving the cell phone that it should leave the call (i.e., the operation of removing an endpoint from the call) and completes the service handoff.

V. Implementation

We have implemented a prototype ICEBERG system, including the signaling system with basic call service functionality, in a testbed setting. At the time of writing, the testbed consists of a GSM digital cellular network and a WaveLAN wireless local area network. We are in the process of building IAPs to interface to the PSTN through an H.323 gateway [21] and to interface to the paging network through a 2-way paging gateway.

The components run on top of Ninja (see Section II-A) and are written in Java. The current prototype has around 5,000 lines of code.

Ninja RMI, a re-implementation of Sun's Java RMI, is the primary communication mechanism for call session establishment. The IAP, Call Agent, Operator, Preference Registry and Name Mapping Service are all Ninja remote objects in our system. For call session management, we use IP multicast for group communication among the call agents. We are temporarily using LDAP [30] for service location and name mapping, and are working on using the SDS for such services. Our Preference Registry encodes user preferences as Perl scripts and takes as input caller iUID, callee iUID, and time of day.

We have set the heartbeat rate and soft state timeout values to the ad hoc values of one second and five seconds respectively. We are still investigating the appropriate choices for these parameters.

[10] A data path is not created between two devices of one call party.

VI. Performance

We have measured the call processing time for our prototype implementation. Our measurement is done on a 166 MHz IBM Thinkpad with 64MB memory. Table VI shows the communication cost which comes from loading Ninja services (like the Preference Registry) and Ninja RMI calls. "Local" refers to loading services from or performing RMI calls on the localhost; and "Remote" refers to loading services or performing RMI calls over a WaveLAN network (with a perceived bandwidth of 300−400kbps, and a latency of 3.6 ms). These figures are obtained by continuously making empty RMI calls for 1000 times.

Items	Num of items		Mean	Standard Deviation
Ninja service loading	5	Local	11 ms	6
		Remote	21 ms	4
Ninja RMI	9	Local	20 ms	9
		Remote	50 ms	17

TABLE II
COMMUNICATION COST FOR THE ICEBERG PROTOTYPE

Table III shows the call setup latency, inbound and outbound call processing time. "Local" refers to the case when an iPOP serves both the caller and callee with all services (like the Preference Registry) running co-located with the Call Agents. "Remote" refers to the case when the caller and callee are served by two different iPOPs, and all other services are running remotely from the Call Agents as well. These figures are obtained from measuring 200 calls.

		Mean	Standard Deviation
Call setup time	Local	880 ms	153
	Remote	1200 ms	176
Outbound call processing	Local	415 ms	105
	Remote	585 ms	120
Inbound call processing	Local	465 ms	112
	Remote	585 ms	120

TABLE III
CALL SETUP

VII. Related Work

In this section, we compare our work with the IETF's Session Initiation Protocol [1] and with H.323 [21].

A. *Session Initiation Protocol*

The IETF's Session Initiation Protocol (SIP) [1] is a client-server application-layer signaling protocol. Its protocol agents are user agents and servers of three types: user agent, proxy and redirection servers. *User agents* issue SIP requests (i.e., INVITE or BYE) to make or terminate calls. A *user agent server* handles the SIP request by accepting, redirecting or refusing the call. SIP requests can traverse many *proxy servers*, each of which receives a request and forwards it on towards a next hop server. A *redirection server* informs the client to redirect its request to a different server. A *registrar* is used to track user locations. Proxy/redirection servers are usually combined with a registrar to achieve mobility.

Both call setup (session initiation) and control (during a call session) in SIP are based upon pairwise communications. We believe that it is sufficient to use SIP for session initiation. However, using a pairwise communication protocol to perform call control does not adapt well to call session dynamics and it complicates service implementation. We illustrate these two aspects in detail in the next two subsections. We believe our lightweight call session management protocol can serve as an extension to SIP for call control after call session establishment.

A.1 Adaptivity to Call Session Dynamics

Call session dynamics include call session membership changes (i.e., a call party leaves the call, or a new call party is invited to join the call) and protocol agent failure. We illustrate each in turn.

Call session membership change

In SIP, each call party in a call session maintains the list of call participants. When an existing call participant invites a new call party to the call, it sends the list of all participants along with the invitation. The callee is responsible for sending invitation messages to each existing participant in the list to establish the communication. This is called "mesh conference" in SIP.

The drawback of this approach is the inherent difficulty of keeping track of the accurate call session membership in a dynamic call session. If a call participant leaves the call during the invitation, then the invitation contains an obsolete call participant who will still be invited by the new call party. Even worse, if multiple call participants are making invitations to several new call parties simultaneously, how do new call parties know of the existence of one another? It is true that a sophisticated protocol could be developed as an extension to SIP to solve these special cases. However, that would significantly complicate the signaling protocol.

Another drawback of this approach is that such a tightly coupled session does not scale to a large number of call participants in a call session.

SIP has a special conference call service called a "bridged conference" which addresses the above drawbacks. A bridged conference has a central control point for the conference, called MCU, that maintains the call session membership. Every new call party must communicate with MCU to join the conference. However, this centralized approach leaves the MCU as the bottleneck and the single point of failure.

SIP also has a "multicast conference" service. However, "multicast" does not refer to multicast signaling, but rather multicast data transport. The signaling is exactly the same as "mesh conference". SIP [1] mentions the process of transitioning from a tightly coupled control session to a loosely coupled one, but how this action is done is unclear.

SIP does have multicast signaling. However, it is reserved for special services such as inviting a group of friends represented by a single identifier, automatic call distribution, etc. These groups are not dynamically formed communication groups, but pre-defined fixed groups (i.e., no group membership changes).

Protocol Agent Failure Recovery

When a protocol agent recovers from faults, it may have a new identity or location. Conveying the new identity of the newly started protocol agent to the rest of session has the same difficulty as the call session membership change mentioned above. Even if the newly restarted protocol agent maintains the same identity, the recovery of the call session changes that were made during its failure is a difficult task for pairwise communication protocols.

ICEBERG

In comparison, the ICEBERG use of lightweight call sessions for group signaling and as a level of indirection for communication adapts to call session membership changes in nature. In addition, the use of soft state automatically informs newcomers to a call session the current state of the session. Soft state maintenance does incur bandwidth overhead. However, the soft periodic messages have small size (less than 100 bytes per call party).

A.2 Simplicity in Service Implementation

Because ICEBERG signaling manages the adaptation to call session dynamics and signaling agent fault recovery, service implementation is simplified and service developers do not have to handle special cases due to call session dynamics and system component failure. Also, the group signaling enables conference calling as a first-class service in a trivial way. Multi-device call operations become building blocks for services as explained in Section IV.

B. *H.323*

Schulzrinne, et. al., in [3], provides an extensive evaluation of H.323 as the signaling protocol for Internet Telephony in four aspects: complexity, extensibility, scalability and services. H.323 is complex due to its numerous component protocols. The protocols do not have a clean separation of functionality among themselves, which leads to many options for doing a single task. H.323 is limited in extensibility: it requires full backward compatibility between its versions; each codec is centrally registered and standardized; and its design is not modular. H.323 has poor scalability: it uses TCP for signaling transport which is stateful, and it has a central control for conference calling which is the bottleneck and a single point of failure. Finally, H.323 has limited user preference support for highly customizable services.

VIII. Future Work

Since an ICEBERG environment may include multiple service providers and portions of the public Internet, we are actively exploring security issues for the signaling systems including user and system component authentication, billing models between different iSPs, and user privacy.

In terms of service creation, we are experimenting with more interesting services such as policy-based call waiting allowing a callee to specify preferences for handling incoming calls during a call. We are also implementing other existing telephone services and new services to test the functionality of our system.

We are enhancing our Data Path Factory to support connectors, and automatic path creation.

In our current implementation, we have provided hooks for admission control, billing functions, and interactions with the PAT. These functions represent separate problems from signaling, but we will provide them to make our system more complete.

We are also performance-tuning our implementation to improve the call setup time and are in the process of deploying our prototype in a wide-area testbed. This testbed will help us to better understand wide-area performance and functionality issues.

IX. Conclusions

Powerful new mechanisms in the ICEBERG architecture, such as the announce/listen group communication model and fault tolerant cluster computing environments, have spurred us to revisit the fundamental assumptions about basing a signaling system on hard state and the atomicity of a point-to-point communications session between signaling elements. We have successfully constructed a lightweight signaling architecture based on IP multicast and soft state that we believe may drastically simplify signaling protocols in the ICEBERG environment. As future work, we intend to quantify the relative benefits and drawbacks of our approach versus traditional approaches, but our initial results look promising.

Acknowledgments

We thank Andrew Begel, Adam Costello, Mark Handley, Tom Henderson, Todd Hodes and Megan Thomas for their very detailed and insightful comments and discussions on this work. We thank Sharad Agarwal, Rahul Biswas, Chen-nee Chuah, Emre Kiciman, Bhaskaran Raman, Suchitra Raman, Angela Schuett and Tina Wong for their help and encouragement on this work. We also appreciate the anonymous reviewers for their insightful feedback. This work was supported by Internet Systems Research Group (ISRG/ICEBERG) from the U. C. Berkeley.

References

[1] M. Handley, H. Schulzrinne, E. M. Schooler and J. Rosenberg, *SIP: Session Initiation Protocol*, Internet Engineering Task Force, Internet Draft, January 15, 1999
[2] H. Schulzrinne and J. Rosenberg, *Signaling for Internet Telephony*, Columbia University Technical Report CUCS-005-98
[3] H. Schulzrinne and J. Rosenberg, *A Comparison of SIP and H.323 for Internet Telephony*, Network and Operating System Support for Digital Audio and Video (NOSSDAV), (Cambridge, England), July 1998.
[4] Steven McCanne, *Scalable Multimedia Communication with Internet Multicast, Light-weight Sessions, and the MBone*, Technical Report CSD-98-1002, Computer Science Division, U.C. Berkeley, July 8, 1998.
[5] Jacobson, V., *Multimedia Conferencing on the Internet* SIGCOMM '94 Tutorial
[6] W. Biemolt, M. Kaat and H. Steenman *A Guide to the Introduction of IPv6 in the IPv4 World* Internet Engineering Task Force, April, 1999
[7] Anthony D. Joseph, B. R. Badrinath and Randy H. Katz *The Case for Services over Cascaded Networks* The First ACM/IEEE International Conference on Wireless and Mobile Multimedia (WoWMoM), 1998
[8] R. Katz et. al. *A Scalable Service Architecture for Computer-Telephony Integration* To be submitted to a conference, 1999
[9] Elan Amir, Steven McCanne and Randy H. Katz *An Active Service Framework and its Application to Real-time Multimedia Transcoding* Proceedings of ACM SIGCOMM 98, Vancouver, British Columbia
[10] Nadege Faggion and Cegetel Thierry Hua *Personal Communications Services Through the Evolution of Fixed and Mobile Communications and the Intelligent Network Concept* IEEE Network, Jul/Aug, 1998
[11] Ninja Project. http://ninja.cs.berkeley.edu/
[12] T. E. Anderson et al. The Case for NOW (Network of Work Stations) IEEE Micro, February 1995
[13] A. Fox et al. *Scalable Network Services* Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16), St. Malo, France, October 1997.
[14] Eric Brewer *ProActive Infrastructure* the DARPA project proposal for Ninja, March 1998
[15] Steven D. Gribble et. al. *The MultiSpace: an Evolutionary Platform for Infrastructural Services* Usenix Annual Technical Conference, June, 1999, Monterey, CA
[16] Guido Appenzeller et. al. *The Mobile People Architecture* Technical Report of Stanford Computer Science Lab CSL-TR-99-777, Jan, 1999
[17] E. Brewer et. al. *A Network Architecture for Heterogeneous Mobile Computing* IEEE Personal Communications Magazine, October, 1998
[18] Mohammed Zaid *Personal Mobility in PCS* IEEE Personal Communications, Fourth Quarter, 1994
[19] S. Deering and D. Cheriton. *Multicast routing in datagram internetworks and extended LANs* ACM Transactions on Computer Systems 8, May, 1990, 85-110
[20] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph and R. Katz *An Architecture for a Secure Service Discovery Service* Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99), Seattle, WA, August 1999
[21] ITU-T Recommendation H.323 *Packet-Based Multimedia Communications Systems* February 1998
[22] Hugh W. Holbrook and David R. Cheriton *IP Multicast Channels: EXPRESS Support for Large-Scale Single-Source Applications* Proceedings of ACM SIGCOMM, Harvard, MA, Sep. 1999
[23] R. Perlman and C-Y Lee and A. Ballardie and J. Crowcroft and Z. Wang and T. Maufer *Simple Multicast: A Design for Simple, Low-Overhead Multicast* Internet Draft (work in progress), Nov, 1998
[24] Yang-hua Chu, Sanjay Rao, Hui Zhang *EndSystem Multicast* http://www.cs.cmu.edu/ kunwadee/research/other/endsystem-index.html
[25] Ion Stoica, T. S. Eugene Ng, Hui Zhang *REUNITE: A Recursive Unicast Approach to Multicast* IEEE Infocom 2000, Tele-viv, Israel, March, 2000
[26] Ljubica Blazevic and Jean-Yves Le Boudec *Distributed Core Multicast (DCM): a multicast routing protocol for many groups with few recievers* NGC '99, Pisa, Scuola Superiore S. Anna, 17-20 November 1999
[27] David Thaler and Mark Handley *On the Aggregatability of Multicast Forwarding State* Technical Report MSR-TR-99-34, Microsoft Research, 1999
[28] Pavlin Ivanov Radoslavov and Ramesh Govindan and Deborah Estrin *Exploiting the Bandwidth-Memory Tradeoff in Multicast State Aggregation* IEEE Infocom 2000, Tele-viv, Israel, March, 2000
[29] Helen J. Wang et al. *ICEBERG: An Integrated Communication System* submitted to IEEE Personal Communications
[30] M. Wahl, T. Howes and S. Kille *Lightweight Directory Access Protocol (v3). IETF, Request For Comments 2251*
[31] D. D. Clark *The design philosophy of the DARPA Internet protocols*, ACM SIGCOMM '88 Symposium, Stanford, CA pg 106-114
[32] B. Y. Zhao *XSet: A High Performance Search Engine* http://www.cs.berkeley.edu/ ravenben/xset/
[33] M. Handley *Session Directory and Scalable Internet Multicast Address Allocation* Proc. ACM SIGCOMM 98, September 1998, Vancouver, Canada
[34] Steve Deering *Host Extensions for IP Multicasting* IETF RFC-1112, Aug. 1989
[35] Maryann P. Maher and Colin Perkins *Session Announcement Protocol: Version 2* IETF Internet Draft: draft-ietf-mmusic-sap-v2-00.txt, Nov. 1998