A Soft State Protocol for Accessing Multimedia Archives

Angela Schuett, Suchitra Raman, Yatin Chawathe,
Steven McCanne, Randy Katz
{schuett,suchi,yatin,mccanne,randy}@cs.berkeley.edu
Department of EECS
University of California, Berkeley

Abstract

The advent and deployment of IP Multicast has fueled the growth of multi-user collaborative applications. The MBone, the Internet multicast infrastructure, extends the traditional best-effort unicast delivery model of the Internet to enable efficient multipoint packet delivery. Using IP Multicast to deliver archived multimedia content can provide significant advantages in scalability and efficiency. With the popularity of collaboration tools for audio/video conferencing on the MBone, we have seen a number of applications developed to record and play back these MBone sessions. In this paper, we look at the requirements for an archive system control protocol that allows the initiation and manipulation of a playback session. We present a control protocol which itself uses IP multicast. This provides benefits in scalability, fault recovery, and allows simultaneous control of an archive session by multiple clients. The interesting features of our protocol include the application of soft *server state techniques, and a built-in mechanism for fault detection and recovery. We have implemented and deployed a prototype and are actively using it in our research group.*

1 Introduction

The development and deployment of the IP multicast infrastructure (the MBone) in the Internet has enabled the development of a number of collaborative applications. IP Multicast [Dee91, DEF+96] provides an efficient mechanism for multi-point data delivery. The new class of multi-party applications enabled by IP Multicast includes a variety of audio and video conferencing tools [Sch92, MJ95, JM], shared whiteboards and text editors [McC92, RT96, HC97], and floor control applications [MR97]. If computer supported collaboration is to become successful and mainstream, it must at the very least provide functionality that was previously unavailable in face-to-face meetings. By harnessing the computing technology at our disposal today, instead of merely imitating face-to-face communication, we can *enhance* collaboration by intelligently adding functionality through the electronic medium [HS92].

A key functionality that is possible and useful in this new medium is the ability to record and play back collaborative sessions, for example, a group meeting. A recording agent can participate in the multicast session associated with the meeting and archive the entire session. Such an archive would be valuable resource for people who missed the meeting, or who wanted to review certain portions of the meeting. Moreover, the ability to play back archived data on to the MBone as part of a new session allows the archive itself to be a tool for collaboration.

Delivering stored multimedia content to users is a complex, multi-part problem. Such a system must include components for disk scheduling on the server, data replication algorithms, media codecs, admission control schemes, etc. This paper is concerned primarily with the archive control protocol, i.e., the protocol used to initiate contact with an archive server, request playback of a presentation, and control that presentation (for example, with pause and seek operations). However, the design of the archive control protocol does interact with and impact other areas of archive architecture, and this needs to be considered in order to achieve fault tolerance, scalability, and robustness.

In this paper we present a soft state protocol for controlling access to archived sessions. Soft state denotes a type of protocol where state is constantly refreshed by periodic messages. State which is not refreshed in time expires. Advantages of the soft state approach include implicit error recovery. The concept of soft state has been widely used for building robust and reliable systems. A number of network protocols for Internet routing [MRR80], IP multicast routing [DEF+96], and the Snoop protocol for improving TCP performance over wireless networks [BSAK95] use soft state to achieve robustness and good performance. We heavily rely on the use of soft state throughout our design. As we will demonstrate, this design choice greatly simplifies fault management resulting in high availability.

Soft state protocols are also amenable to implementation on multicast. We explore the use of multicast to carry the archive control protocol. This allows clients that are sharing a data session to also share a control session, enhancing scalability. We also use multicast in our archive architecture to mask failures in the archive service.

The remainder of the paper is organized as follows: Section 2 outlines functions of the archive control protocol. Section 3 describes SSAC, our soft state archive control protocol. Section 4 presents our implementation of SSAC-based client and server applications. Section 5 describes the mechanisms for evaluating this type of archive control protocol. Section 6 discusses related work, and Section 7 summarizes this paper.

2 Protocol Functionality

We describe here the three core pieces of functionality provided by our archive control protocol: (1) *maintaining shared control state*, (2) *group organization* and (3) *fault detection and recovery*. We discuss each of these issues in more detail below.

2.1 Maintaining Shared State

The primary function of an archive control protocol is the exchange of control parameters, for example, the title of the presentation to be played, data streaming addresses, the server's current position in time in the stream, etc. This parameter exchange causes both the server and the client to construct a shared state table.

One approach to establishing this state table is to use a reliable, connection-oriented byte stream transport protocol such as TCP. The Real-time Streaming Protocol (RTSP) [SRL98] uses this approach[1]. While this is straightforward in the single client-single server case, it is not suitable when multiple clients share a single control session. If each client in the shared session requires a separate connection to the server, the server must maintain expensive per-client connection state. An alternative to this approach is for a single client to contact the server, obtain a session multicast address and announce the multicast channel where the presentation is being played. This approach, however, does not allow more than one client to cooperatively control a session. If the controlling client fails, the "remote control"is lost!

2.2 Group Formation and Management

The full benefits of using IP multicast for data and control depend on the ability of the control protocol to group multiple end-clients together. Two different scenarios exist for multiple client control: explicit and implicit client groups. An *explicit* group is formed when a group of clients wishes to view a single presentation together, under group control. For example, in an "online" distributed classroom setting, geographically distributed users interacting with MBone conferencing tools may wish to jointly view a recorded presentation. As the presentation progresses, participants may pause the playback in order to discuss interesting points among themselves. Using a VCR metaphor, each participant has a "remote control", which operates on a single archive system shared by all. If one group member performs a seek or pause operation, that operation must be propagated to all other members of the group.

An *implicit* group, on the other hand, is formed when independent clients simply happen to be viewing the same playback session at the same point in time. Servers may coalesce these clients into a single group to improve efficiency. If any of the members of such an implicit group perform a seek/pause operation, the group is automatically split up. The use of implicit client groups has been explored in several systems as a means to improve server scalability [DSS94, BNKL98].

2.3 Fault Detection and Recovery

Fault detection in the context of media streaming systems involves detecting both client and server failures. If a client-side failure occurs, the server should detect this, and stop delivering data. This is necessary so that server resources do not become "orphaned". Similarly, the client should be able to detect server failures and notify the user and, possibly, try to recover from the failure by switching to a different server with minimal disruption of service.

Client or server failure can be detected through several means, such as the loss of a TCP connection, or the timeout of some periodic "heartbeat" message. It is also possible to use the playback data stream as a source for fault detection. However, this approach has several problems:

- First, some data delivery protocols may not contain the periodic "heartbeat" messages required to detect faults.
- Second, detecting archive faults through data streams impedes reuse of mechanism. For example, a typical scenario for an Internet archive system may involve developers who provide data streaming modules corresponding to proprietary formats which plug into the archive system. If the archive control interacts on a fine-grained basis with these data streaming modules, then adding support for new modules will be more difficult.
- Finally, clients should not be required to receive data in order to control playback, and playback viewers should not necessarily be required to run the archive control protocol. As a motivating example, consider a presentation room which features a variety of display devices. The speaker controls the playback of a presentation from her palmtop, while the video data stream is actually received on a computer with a large display screen. With protocols that depend on detecting faults through the data stream, the palmtop controller would unnecessarily be required to receive all the data streams.

[1]RTSP does not require the underlying protocol to be TCP, but does expect RTSP messages to be received in-order and reliably.

We believe that the appropriate place for fault detection and recovery mechanisms in an archive system is the archive control protocol. Fault discovery and recovery are especially important for an MBone archive system. New protocols and codecs are constantly being developed, and software is being deployed in the wide area well before all the bugs have been worked out. In addition, network connectivity problems are especially frequent on the MBone. By incorporating a robust fault tolerance protocol in the archive design, we can ensure that the service continues to operate even in the presence of buggy software modules or network failures.

2.4 Design Principles

We have developed an archive control protocol that addresses maintenance of shared state, group formation and management, and fault detection and recovery through the use of the multicast protocol design principles of *light-weight sessions* (LWS) [McC98, Jac94]. Light-weight sessions refers to the fault tolerant, scalable, decentralized communications model that has been used successfully in many MBone conferencing protocols such as RTP/RTCP [SCFJ96] for audio and video and SAP [HJ97] for session announcements. Instead of instantiating "hard" server state as in centralized protocols, light-weight sessions use a "soft state announce/listen approach," in which an agent periodically transmits a status announcement. Listening members update their status information on receiving an announcement. If no announcements are received, state eventually times out.

An important design goal that has consistently underscored the development of the LWS architecture is an explicit attempt to accommodate the highly heterogeneous and loosely-controlled nature of the Internet. The Internet is large, complex, and composed of heterogeneous components with diverse failure modes and mixed levels of reliability. To account for this, LWS protocols are deliberately designed to be robust and forgiving. For example, if a network link outage causes a temporary network partition, then the higher-level LWS protocols do not "reset" or terminate the session. Instead, the session continues its existence as a set of partitioned sub-sessions. Later, when the partition heals, the session gracefully recovers and integrates independently generated activity and state across the healed partition. LWS achieves this robustness to network failures by exploiting receiver-driven, soft state protocols that treat recovery from such failures as part of normal protocol function.

We apply LWS to multicast archive control, to ensure that our protocol is scalable, fault tolerant and efficient. Specifically, we use soft state between clients and server, and, whenever possible, application and protocol decisions are performed in a decentralized manner. This lessens the server load and simplifies the implementation of explicit client groups.

3 Soft State Archive Control (SSAC) Protocol

The central functionality of the Soft State Archive Control Protocol (SSAC) is to implement the exchange of control information between the clients and server of an archived session. For example, before playback can begin, the client must inform the server about the session that it wishes to be played, and a playback point within the session. While the session is in progress, the client may decide to momentarily pause the session, or seek to a different point in the session. The SSAC protocol must robustly convey all such information between the clients and the server.

In addition to the above control information, the client must be able to determine other parameters such as the location of the server, the "name" of the presentation to be played out, media types in the presentation, data formats in the presentation, etc. For the purpose of the archive control protocol, we assume that the client has already previously gathered this information. This information may be collated in the form of a "presentation description file" and placed on the world-wide web, where the client can easily download it.

Based on the soft state periodic update approach of light-weight sessions, we have designed the SSAC protocol to use "announce/listen" as the underlying transport mechanism. A participant in an announce/listen session periodically *announces* its state updates, while all other participants *listen* to and cache these updates. The updates are assumed to be soft state and are eventually aged out, unless over-ridden by a new update from the announcing client. Announce/listen protocols are robust and can be implemented over an unreliable datagram transport protocol such as UDP. Since no central hard state is involved in such a protocol, it makes the protocol much more resilient to failures.

The use of announce/listen eliminates the need for heavy-weight reliable protocols such as TCP or SRM [FJM$^+$95] in the implementation of SSAC. It allows SSAC to run over unicast UDP or multicast.

To present the protocol, we begin by explaining the case where a single client connects to the server using unicast. In Section 3.2 we expand this description to include multiple clients sharing a multicast control channel. To illustrate the fault recovery mechanisms of SSAC, throughout this section we will use a simple example distributed server architecture, shown in Figure 1. The client uses SSAC to communicate with a **server manager**. When the client requests a new presentation, an **archive agent** is launched, which streams the presentation onto a data channel.

3.1 Single Client SSAC

In an archive control session, certain pieces of information need to be exchanged, some originating with the client, and some with the server. The client originates three state parameters: the control session id, the presentation id, and the

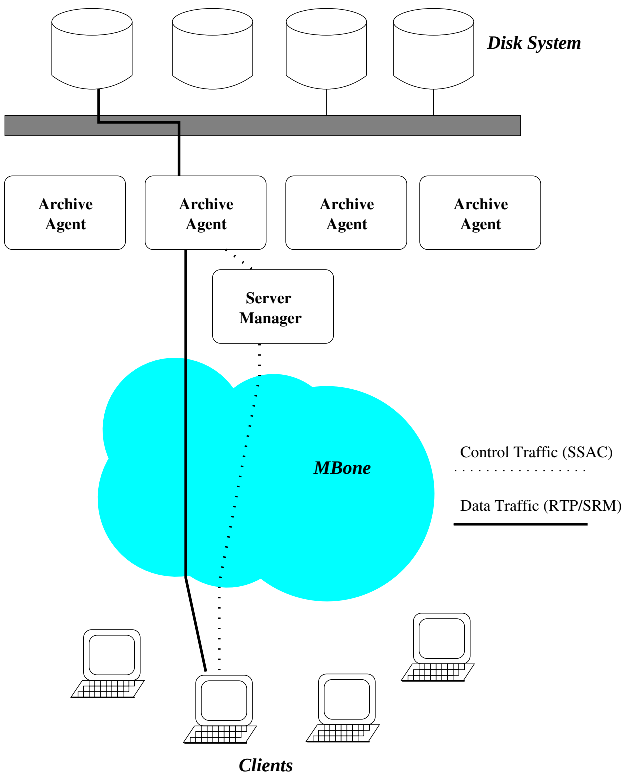

Figure 1: Archive server system showing control and data channels

offset into the playback session. If the client wishes to play the session from the beginning, the offset is 0. The server transmits the values of other parameters, including data address blocks, and the presentation duration.

CSID	210@128.32.130.14
PSID	ssac://pub/classes/CS294/jan30
Offset	0
Seq	1

Table 1: Example SSAC client announcement

Table 1 shows a sample client announcement packet. The control session id, (CSID) can be any client generated information which will be unique at the server for the life of the control session, for example, *client process id : client IP address*. The presentation id (PSID) gives the unique name of the program to be played. This could be similar to a web URL. The Offset field is the means to transmit information about the time offset into the data session. For example, a seek operation is implemented by modifying the Offset field. The Seq field is a sequence number which is incremented by the client or server whenever a change is made to the shared state.

CSID	210@128.32.130.14
PSID	ssac://pub/classes/CS294/jan30
Offset	5
Seq	2
Media	Video
Address	224.2.3.4/5555
Media	Audio
Address	224.2.3.5/6666

Table 2: Example SSAC server announcement

Once an archive agent is launched, and has received its first client announcement, it begins reflecting the control state, which includes the CSID, PSID and Offset, as well as some server-originated parameters, such as data address blocks. Table 2 shows a sample server announcement packet. The server chooses the multicast addresses where data sessions are transmitted. These addresses, along with information about which media type is transmitted on each address, are specified in the address blocks.

Each server packet uses the Offset field to transmit the current offset into the playback session. This offset is then reflected by the client in its announcement. As we will see, in the case of an archive agent crash, this allows a new agent to begin streaming from the proper place in the session. This periodic update of the offset field does not cause the sequence number to be incremented. If the client performs a seek operation, the client announces a new offset value, and increments the sequence number. When the archive agent performs the seek request, it sets the offset field to reflect the new value, and increments the sequence number field again. In this way, client and server announcements can “cross” in the network without harm. Messages which have out of date sequence numbers are discarded.

Throughout an SSAC session, the server and client periodically announce state information. This has two purposes. First, if an announcement is lost in transmission, the next announcements will serve as replacements. Second, if the archive agent crashes, the next client announcement to arrive at the server manager will contain all the information required to restart the playback session.

Fault discovery and recovery are simplified by the use of soft state. Recall that in a soft state protocol, state mappings time out if they are not refreshed periodically. If the client crashes, the server’s state will time out, and the server will stop sending data. If the client recovers and begins announcing again, the server will resume data transmission. If the archive agent crashes while streaming, SSAC messages which arrive at the server manager will cause a new archive agent to be created. The new archive agent receives the offset, and begins streaming from the logical position in the stream denoted by the offset. The client begins receiving new announcements, notes the new data channel addresses,

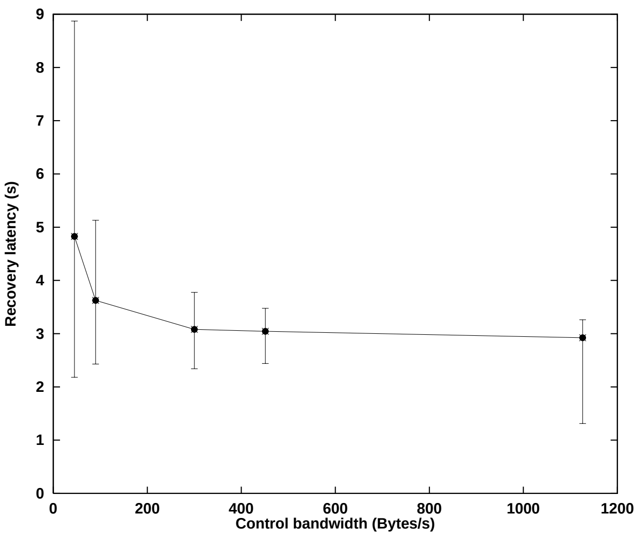

Figure 2: Tradeoff between control session bandwidth and time to discover a fault

and switches to receiving information from the new archive agent.

The time to detect and recover from a fault depends on the periodicity of client and server announcements. If a client announces once every T time units, it will take kT time to detect a fault, for some small value of k. Figure 2 shows the tradeoff between the time to detect a fault at the server side and the control session bandwidth. For these measurements we set $k = 4$, so when 4 announcements are missed, a fault is registered.

The three required parameters of control session id, presentation id, and offset are all that is required for most playback operation. However, there are cases, such as fast-view, or skimming, where other parameters will be required. In the case of fast-view, the playback rate, or scale is altered. To play back the presentation twice as fast, the scale should be set to 2. These optional parameters are usually not included in announcements. If the server does not receive values for these parameters it uses well-known defaults. For instance, the default scale is 1. However, if the client wishes to set one of these optional parameters, it must include the new value in the message, and continue to include this parameter in announcements until the end of the session, or until the session should return to the default value. This is necessary for two reasons. First, since SSAC is run over an unreliable data transmission protocol, the client needs to continue sending the new parameter until it has verified that the server has received it. This verification comes when the client receives a server announcement containing the parameter's new value and incremented offset field. However, even after receiving verification, the client needs to continue announcing this parameter so that in case of an archive agent fault, a new agent can be updated with the current state, and begin streaming.

3.2 Multi-client SSAC

In the previous section, we described the basic workings of SSAC using a single client example. In this section, we describe how the protocol is used when multiple clients control a playback session. Since the means of contacting the archive service is multicast, allowing multiple clients to share a control session requires only a few slight changes to SSAC. Four main problems need to be addressed in adjusting the basic SSAC mechanisms to multi-client operation: specifying client groups, associating control channels with groups, limiting control bandwidth, and performing state changes. We address each issue in turn.

3.2.1 Specifying Groups

The effectiveness of using IP multicast for data and control depends on the ability to group multiple end clients together. Two different scenarios exist for multiple client control: explicit and implicit client groups. An *explicit* group of clients is organized to view a presentation together, under group control. If one group member performs a seek or pause operation, the other members wish to see that operation reflected locally. An *implicit* group is composed of clients who happen to be viewing the same playback session at the same point in time. Servers may coalesce clients together in order to improve efficiency. However, if members of the merged group perform seek operations, they should be split into new groups. As we will see, the correct policy for each group can be achieved by setting the control session id field of SSAC.

To form an explicit group, all members of the group need to use the same control session id (CSID) value in their SSAC announcements. This will cause them to all be updating the same entry in the server's state table. The CSID for an explicit group can be any numeric or text string which will be unique, at the server, for the life of the control session, for example, a group member's e-mail address. This CSID is propagated to the group members through some separate means, such as *sdr* [HJ97]. In addition, group members need to share a starting time for their playback session.

Implicit groups can also be formed by using a shared CSID. This control session id can be constructed from the unique presentation name, PSID, and some timestamp which indicates the position in time within the presentation. We use a timestamp which indicates the wall clock time when the session would have started playing from offset 0. Thus, if the session began playback from offset 50 at time 10050, the start timestamp (STS) would be 10000. This timestamp remains constant throughout the session unless a seek operation occurs. Using these two fields to create the CSID handles implicit grouping of clients, and ensures that only clients viewing the same data stream will be grouped into a logical control session. When a client performs a seek, that client's PSID:STS will change, causing the client to leave the group. If another group is watching a presentation at the new position, the seeking client can join this group. If no other group exists with the new CSID, then a new archive agent is launched.

3.2.2 Limiting Control Bandwidth

Since multiple clients are participating in each control session, it is important to set the rate of announcements correctly. The rate at which each group member announces is adjusted as the group size grows, in a mechanism similar to that used in RTCP [SCFJ96]. The goal is for each of n members to announce at a rate of p/n, where p is the announcement period. As we discussed in Section 3.1, the length of the announcement period determines the length of time before a fault is discovered and corrected.

Limiting a client to announce keep-alive messages at a rate of p/n is no sacrifice, since the client benefits from all other group member's announcements. However, if a client wishes to perform a seek operation, this message should not be limited to the usual announcement cycle, or the response time would be quite poor. We propose that clients be allowed to immediately send seek messages when needed. We do not believe this will have a major impact on control bandwidth since seek operations are relatively infrequent.

3.2.3 Performing State Changes

As we discussed in Section 3.2.1, state changes should be reflected to group members differently depending on whether the group is explicit or implicit.

Implicit groups reflect parameters in their control session id. If a member wishes to change a parameter, for instance to perform a seek operation, it is reflected by a change in the CSID. This causes the client to leave its current group and join or start a different control session. If another group is watching a presentation at the new position, the seeking client can join this group. If no other group exists with the new CSID, then a new archive agent is launched. Depending on the server workload, various policies could be used to associate users into groups. For example, programs may only be allowed to begin on the hour and half hour. If a client requests a program at an intermediate time, that request will be denied. Or, clients could be coalesced into groups by the server, through a server message which requests the client to switch to a new channel.

Explicit groups wish the state changes of one member to be reflected to the entire group. When a member attempts to perform a state change the resulting state update message is multicast to the entire group. The state change has taken place when the server announces the new state. Since all clients are allowed to make state changes, it is possible to have race conditions. Two state changes could be announced at the same "time". Members of the group would receive these messages in different orders depending on their position in the network. For this reason, if two state change messages collide, it is up to the server to decide which action to perform. Clients consider state change messages to be provisional until a server reflection is heard.

4 Implementation

We have implemented the core SSAC protocol in a distributed archive server which we call MARS (**M**ASH **AR**chive **S**erver), and a client application, the MARS rover. We utilize the established MBone tools *vic* [MJ95], *vat* [JM] and *mediaboard* [RT96] for transmitting content. We have recorded around 20 Gigabytes of content from MBone lectures, research group meetings, and local classes, and provide access to these sessions through the MARS system.

4.1 Client Application

A screen shot of our client application, MARS rover, is shown in Figure 3. The client can run as a helper application from Netscape. We have constructed a web page which lists presentations which are available for playback. Clicking on a presentation causes rover to be launched, already configured with the server multicast address, and the presentation id. The client immediately begins announcing using the SSAC protocol. When the first response from the server arrives, the client configures itself to listen on the indicated data sessions, and begins displaying data.

The MARS rover was constructed using objects from the MASH toolkit. The MASH toolkit [MBKea97] is an array of composable multimedia networking and application objects. High performance objects, such as media encoders, decoders, recording and playback objects are written in C++. These objects are then glued together using the Tcl scripting language, along with Tcl/Tk components for user interfaces, parsers, and policy components. Along with our SSAC implementation, the MARS rover includes code taken from MASH vic and MASH vat, allowing us to display audio and video data within the client application without reimplementing any of these difficult objects.

4.2 Archive Server

In order to make the best use of the scalability and fault recovery features of the SSAC protocol, an appropriate server architecture and implementation is required. An ongoing research effort at Berkeley is developing a service toolkit framework, AS1, which uses an announce/listen soft state protocol for launching service agents [AMK98]. This fits in well with the SSAC protocol design. Using AS1 we have built an archive server which utilizes SSAC messages to automatically restart failed archive agents. The AS1 framework is also implemented using the MASH toolkit.

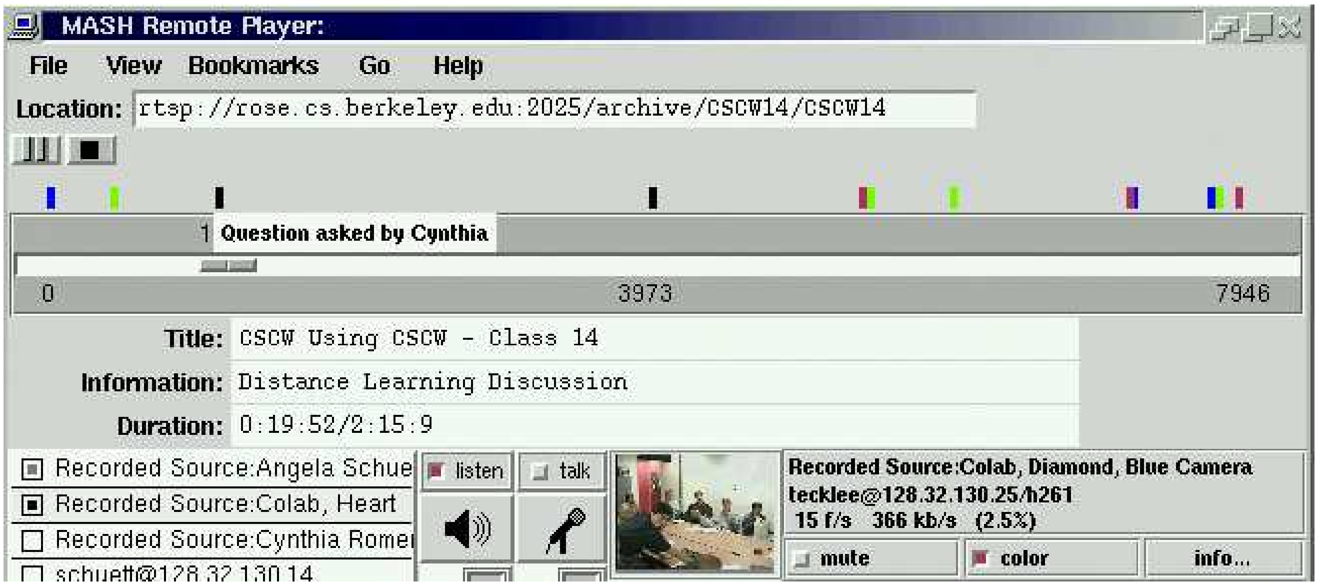

Figure 3: MARS rover user interface

The basic system architecture of AS1 is shown in Figure 4. The server is constructed out of a number of host machines, all listening to a single multicast address. Each host machine runs a **host manager** (HM) process. This process is responsible for listening to control traffic and launching **archive agent** (AA) processes when needed. See the AS1 paper [AMK98] for details on how a minimum number of host managers can be maintained in the system at all times. The AS1 system allows the set of member host machines to be completely fluid. The entire service can migrate to a new set of machines within a very short time period. In our server implementation, the host machines are on the same local network, and have access to the same disk system.

Each host manager maintains a state table matching control session ids to archive agent ids. In keeping with the soft state philosophy, this state must be updated periodically, or it will expire. Arrival of a message with a CSID which is not in the state table causes the host manager to attempt to launch an archive agent. In order to avoid launch floods, where multiple archive agents are launched for each request, timers and multicast damping are used. When a host manager attempts a launch, it first sets a randomized timer. After the timer expires, the manager will launch a new archive agent, which immediately announces its state binding to that CSID. Other host managers with launches pending for that CSID will cancel their launches. For information about other heuristics which are used to handle launch collisions, see [AMK98].

If an archive agent aborts, the soft state mapping which binds that agent to a CSID will expire. After the state expiration, the next announcement with that CSID will be treated by host managers as a new control session, and the host managers will attempt to launch a new archive agent. No action is required by the client to request a new server, and no special code is required at the server to restore state or enter a special fault recovery mode.

This architecture offers some unique advantages as the basis for an archive server. First, since there is no single point of failure, such as a centralized manager, the system is more robust. Since host machines can be added and removed on the fly, the system can adapt to server load. Finally, since the AS1 framework is designed to use multicast for service creation, fault recovery is automatic and efficient.

5 Evaluation

Evaluation of an archive control protocol can take place according to several metrics: functionality, scalability, and latency. By functionality, we refer both to the set of functions supported by the protocol, and also the set of functions which can be implemented on top of the protocol mechanisms. Scalability in this case refers to the additional cost of adding users to a control group, and the additional cost of adding sessions to a control channel. Latency covers the time required to join an existing control session, the time execute a state change, such as a seek operation. We provide a preliminary discussion of SSAC based on each of these metrics.

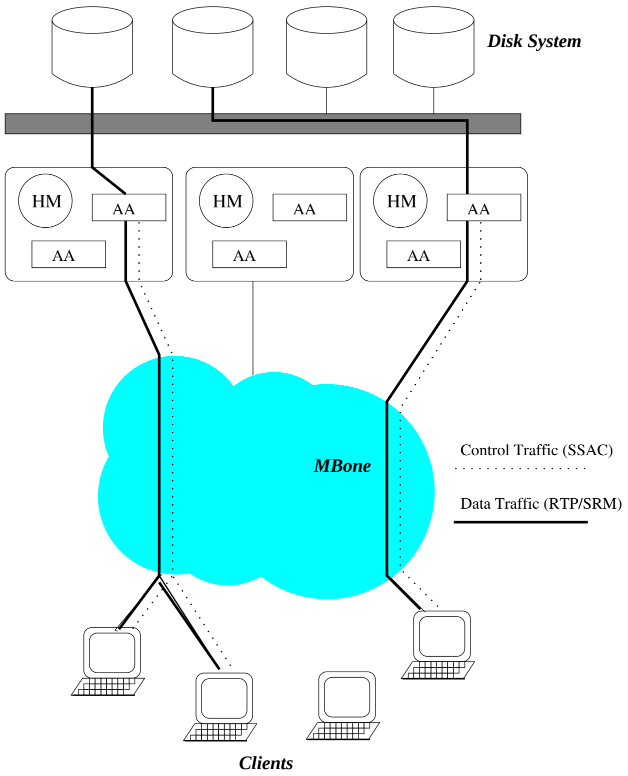

Figure 4: AS1 Architecture

5.1 Functionality

SSAC provides all of the operations needed to initiate and control a playback session. Using SSAC mechanisms, applications can request playback, perform seek operations, and set a variety of playback control parameters. SSAC also provides the ability to group together viewers into either implicit or explicit viewing groups. We believe that the combination of these mechanisms provides a platform on which to construct a rich interactive playback environment, with a variety of playback control policies, including moderated playback control, token based and lock based control schemes, or laissez-faire cooperative control.

5.2 Scalability

As stated above, in the context of an archive control protocol, scalability refers to the additional cost of adding a client to a control session.

Since SSAC was designed using the techniques of lightweight sessions, scalability of the protocol is not limited by expensive join and leave operations. When a new client connects to an existing session, no explicit server or group notification is required. Instead, clients merely start receiving state messages and repeating the state in their own heartbeat messages.

Adding a new client to a control session can have several potential costs. If the server needs to keep a list of attached clients, then each new client requires an expenditure of processing and memory at the server. Using SSAC, however, the server needs no knowledge about the number of users, only that there is at least one client attached. As long as heartbeat announcements are being received, the server need not differentiate whether they originate from a single client, or multiple clients. However, clients are required to keep track of other clients in the session. This is necessary in order to keep control bandwidth constant as clients are added to the session. To adapt the per client control bandwidth to the total number of clients, we use the mechanisms of RTCP, where clients build a soft state table listing participants. Each heartbeat message updates this table. Several papers have recently proposed schemes to improve the scalability of RTCP [RS97, Abo97], and SSAC should be able to incorporate this work.

While client heartbeat messages are rate controlled, client initiated state change messages may be sent immediately. We believe that these operations are infrequent enough that they will not affect the control session bandwidth.

5.3 Latency

According to the soft state model used by the SSAC protocol, state changes do not need to reach the entire group before they can take effect. The archive server is the arbiter and control point for state changes, and a change takes effect across the control session when it is announced by the server. In order to provide a fast response time for state changes, we advise that state change messages be sent immediately. So, the SSAC-imposed state change latency may be as low as 2 round-trip times, which is comparable to the latency of a connection-oriented protocol.

6 Related Work

Video on demand systems have been a research topic for much longer than the MBone has been part of the Internet. Several commercial systems have been deployed and tested in thousands of homes. These commercial systems have recognized the importance of fault recovery mechanisms in server implementation [NLO95, BFD97].

Recently, a large effort has been undertaken to standardize many aspects of archive control. The resulting protocol, RTSP, addresses the issues of a common connection oriented control protocol which answers the needs of a variety of current streaming application vendors [SRL98]. However, RTSP does not address multiple client control, and does not provide any explicit mechanisms for fault discovery or recovery.

The MBone VCR on Demand service [Hol97] allows multiple clients to connect to a server for session playback.

However, only one client may have control of the session, with other clients designated as listeners only. In addition, each client keeps a separate connection open to the server throughout the session, which limits the system scalability.

The Internet Multimedia Jukebox [AA98] is a research effort which is exploring the viability of grouping users together into implicit groups. The IMJ system uses the web as a means to gather requests and present play schedules. Users may request playback, but have no means to control the resulting schedule, cancel playback or perform seek or pause operations.

A number of other systems outside the realm of archive control have used the notion of soft state to achieve better performance and robustness. Networking systems for routing protocols [MRR80, DEF+96] and performance optimization [BSAK95] rely on soft state to update their tables; this allows them to continue operating in the event of partial failures and ensures that no special recovery protocols are needed to regenerate state that might have been lost in a crash. A number of soft state systems such as Bayou [DPS+] have explicitly traded consistency for availability in application-specific ways. [FGC+97] use the principles of soft state and eventual consistency heavily in their design of a general-purpose Internet service framework.

7 Summary

We have presented a new protocol for archive control that allows multiple clients to initiate and control playback of stored multimedia content in a scalable manner. Because this protocol is based completely on soft state and the announce/listen communication model, fault detection and recovery are easily achieved, in contrast to the difficulties encountered in these areas with connection-oriented protocols. The use of shared multicast control and receiver driven design allows additional clients to be added to a control session without impacting server performance or requiring additional server state.

As multicast capabilities become available to more users, large scale archives will use multicast to provide a mechanism for efficiently delivering multimedia content to clients. Our control protocol is designed to efficiently utilize the multicast delivery model to allow clients to form groups either implicitly (because they happen to be viewing the same session at the same time), or explicitly. The archive server incorporates fault detection and recovery mechanisms to ensure high availability of the system.

We have been using our prototype implementation to view our recorded weekly research group meetings. We are continuing to explore the applicability of this protocol to a variety of archival workloads, and expect to refine it as we better understand the interactions between the various pieces of the system.

8 Acknowledgments

We would like to thank Elan Amir for his help throughout the design and development of SSAC and MARS. Thanks also to James Landay for his input into the design of rover. In addition, we are grateful to Cynthia Romer, Andrew Huang and the entire MASH group for using the MARS server and client, and for their feedback. Finally, thanks go to the anonymous reviewers for their comments and suggestions.

This work was supported by DARPA contract N66001-96-C-8508, by the State of California under the MICRO program, and by NSF Contract CDA 94-01156. Angela Schuett is supported by a National Physical Science Consortium Fellowship.

References

[AA98] K. Almeroth and M. Ammar. The Interactive Multimedia Jukebox (IMJ): A New Paradigm for the On-Demand Delivery of Audio/Video. In *Proceedings of the Seventh International World Wide Web Conference*, April 1998.

[Abo97] Bernard Aboba. Alternatives for Enhancing RTCP Scalability. Internet Draft, Internet Engineering Task Force, January 1997.

[AMK98] Elan Amir, Steve McCanne, and Randy Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. Submitted for publication, 1998.

[BFD97] William J. Bolosky, Robert P. Fitzgerald, and John R. Douceur. Distributed Schedule Management in the Tiger Video Fileserver. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, October 1997.

[BNKL98] P. Basu, A. Narayanan, R. Krishnan, and T.D.C. Little. An Implementation of Dynamic Service Aggregation for Interactive Video Delivery. In *Proc. SPIE - Multimedia Computing and Networking*, 1998.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP Performance Over Wireless Networks. In *Proceedings of the first ACM Conference on Mobile Computing and Networking*, Berkeley, CA, November 1995.

[Dee91] Stephen E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, December 1991.

[DEF+96] Stephen Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. An Architecture for Wide-area

Multicast Routing. *IEEE/ACM Transactions on Networking*, 4(2), April 1996.

[DPS+] A. Demers, K. Peterson, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The Bayou Architecture: Support for Data Sharing Among Mobile Users.

[DSS94] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-demand Video Server with Batching. In *ACM Multimedia*, 1994.

[FGC+97] Armando Fox, Steven Gribble, Yatin Chawathe, Eric Brewer, and Paul Gauthier. Cluster-based Scalable Network Services. In *Proceedings of SOSP '97*, pages 78–91, St. Malo, France, October 1997.

[FJM+95] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proceedings of SIGCOMM '95*, Boston, MA, September 1995. Association for Computing Machinery.

[HC97] Mark Handley and Jon Crowcroft. Network Text Editor (NTE): A scalable shared text editor for the MBone. In *Proceedings of SIGCOMM '97*, Cannes, France, September 1997. Association for Computing Machinery.

[HJ97] Mark Handley and Van Jacobson. SDP: Session Description Protocol. Internet Draft, Internet Engineering Task Force, November 1997.

[Hol97] Wieland Holfelder. Interactive Remote Recording and Playback of Multicast Videoconferences. In *Proceedings of the Fourth International Workshop on Interactive Distributed Multimedia Systems and Telecomminication Services (IDMS)*, 1997.

[HS92] Jim Hollan and Scott Stornetta. Beyond Being There. In *Proceedings of CHI '92*, 1992.

[Jac94] Van Jacobson. SIGCOMM '94 Tutorial: Multimedia conferencing on the Internet, August 1994.

[JM] Van Jacobson and Steven McCanne. *Visual Audio Tool*. Lawrence Berkeley Laboratory. Software available at ftp://ftp.ee.lbl.gov/conferencing/vat.

[MBKea97] Steve McCanne, Eric Brewer, Randy Katz, and Lawrence Rowe et al. Toward a Common Infrastructure for Multimedia-Networking Middleware. In *Proceedings of the Fifth International Workshop on Network and OS Support for Digital Audio and Video (NOSSDAV)*, May 1997.

[McC92] Steven McCanne. A Distributed Whiteboard for Network Conferencing. Class report, UC Berkeley, May 1992.

[McC98] Steven McCanne. Scalable Multimedia Communication with Internet Multicast, Lightweight Sessions, and the MBone. *Proceedings of the IEEE*, 1998.

[MJ95] Steven McCanne and Van Jacobson. *vic*: A Flexible Framework for Packet Video. In *Proceedings of ACM Multimedia '95*, pages 511–522, San Francisco, CA, November 1995.

[MR97] Radhika Malpani and Lawrence A. Rowe. Floor Control for Large-Scale MBone Seminars. In *Proceedings of ACM Multimedia '97*, 1997.

[MRR80] J. McQuillan, I. Richer, and E. Rosen. The New Routing Algorithm for the ARPANET. *IEEE Transactions on Communications*, 28(5):711–719, May 1980.

[NLO95] Michael N. Nelson, Mark Linton, and Susan Owicki. A Highly Available, Scalable ITV System. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, December 1995.

[RS97] Jonathan Rosenberg and Henning Schulzrinne. Timer Reconsideration for Enhanced RTP Scalability. Internet Draft, Internet Engineering Task Force, July 1997.

[RT96] Suchitra Raman and Teck-Lee Tung. Mediaboard using the Scalable Reliable Multicast Toolkit. Class report, UC Berkeley, December 1996.

[SCFJ96] Henning Schulzrinne, Steve Casner, R. Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. Internet Engineering Task Force, Audio-Video Transport Working Group, January 1996. RFC-1889.

[Sch92] Henning Schulzrinne. Voice Communication Across the Internet: A network voice terminal. Technical Report TR-92-50, University of Massachusetts, Amherst, 1992.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). Internet Draft, Internet Engineering Task Force, February 1998.