AT&T Labs - Research

subject: **Design Considerations for an RTSP-Based Prefix-Caching Proxy for Multimedia Streams**

date: **September 7, 1999**

from: **Stephane Gruber**
Dept. HA173000
FP B150
973-360-8279
gruber@eurecom.fr
HA173000-990907-01TM

Jennifer Rexford
Dept. HA173000
FP A169
973-360-8728
jrex@research.att.com
HA173000-990907-01TM

Andrea Basso
Dept. HA155000
NSL 3-219
732-345-3302
basso@research.att.com
HA155000-990907-01TM

TECHNICAL MEMORANDUM

Multimedia streaming applications typically experience high start-up delay, due to large protocol overheads and the poor delay, throughput, and loss properties of the Internet. Internet service providers can improve performance by caching the initial segment (the prefix) of popular streams at proxies near the requesting clients. This paper analyzes the protocol and architectural challenges of realizing a prefix-caching service within the RTP, RTSP, and RTCP standards. We emphasize how to exploit existing protocol features (such as Range requests), and suggest extensions to RTSP that would ease the development of new proxy services. In addition, we describe how to provide reliable transport, cache coherency, and RTCP feedback control under prefix caching. Then, we present a preliminary implementation of prefix caching on a Linux-based PC. We describe how the proxy operates with a commercial RTSP server and client. The paper ends with a summary of proposed extensions to RTSP and directions for future research.

The paper is available on the web at http://www.research.att.com/~jrex/papers/tm990907.ps

1 Introduction

The popularity of Internet multimedia applications has grown dramatically in the past several years, spurred by the penetration of the Web and the increasing capacity of backbone and local-access networks. However, multimedia streams typically experience high start-up delay, due to protocol overheads and the poor delay, throughput, and loss properties of the Internet. The route from the server to the client often traverses multiple ISP domains, giving the client's service provider little control over the end-to-end performance. Even when better quality-of-service models are available, such as Premium or Assured service, the customer ISP may wish to rely on best-effort service for lower financial costs. Response time can be reduced by caching popular streams at proxies near the requesting clients (e.g., at the head-end of the access network). The proxy-client path can be carefully provisioned for good quality-of-service. In this paper, we describe how to build a multimedia proxy service that isolates the proxy-client path from the server-proxy path, without support from the server.

Despite the potential performance benefits of proxy caching, the large size of most multimedia streams makes conventional caching techniques inappropriate. Rather than storing the entire contents of a large audio or video clip, the proxy could cache a set of frames from the beginning of the stream (i.e., the *prefix* of the clip) [1] . This allows the proxy to initiate transmission to the client while simultaneously requesting the remainder of the stream (the *suffix*) from the server. Previous work has explored how to size the prefix, how to acquire the initial prefix from the server, how to fetch the suffix, and how the proxy can exploit the prefix to perform workahead transmission into the client playback buffer.

For example, the prefix could be cached while satisfying a client request, prefetched from the server upon access to the enclosing Web page, or pushed explicitly by the server. The prefix should be large enough to hide the round-trip delays, jitter, and small time-scale fluctuations in server-proxy throughput (e.g., 5-second prefix). Caching a larger prefix reduces the amount of traffic between the server and the proxy on subsequent client requests, and allows the proxy to support additional functions, such as transcoding or workahead transmission, without introducing client delay. While transmitting the prefix to the client, the proxy can simultaneously request the remainder of the stream from the server using existing protocol mechanisms, such as the range requests available in HTTP/1.1 [2] and RTSP [3] .

Multimedia proxy services, such as prefix caching, can play an important role in improving user performance and network efficiency for streaming audio and video in the Internet. For ease of deployment, these services should not require changes to existing client/server software or network mechanisms. This paper explores the challenges of realizing the prefix-caching service in the context of the RTP, RTSP, and RTCP protocols. We identify how the proxy should alternate between acting as an application-level router or as a content server, when forwarding packets from the origin server or from the cache, respectively. In addition, we present the requirements for the proxy architecture, and describe how to exploit the prefix to allow reliable transmission on the server-proxy path, and perform transcoding and workahead smoothing on the proxy-client path. In addition, we propose extensions to RTSP that would ease the development of new multimedia proxy services.

Section 2 briefly reviews the RTP, RTCP, and RTSP protocols. In Section 3, we emphasize how to exploit existing protocol features, and consider how to support range requests, cache coherency, and sender/receiver reports. Drawing on the protocol analysis, Section 4 describes the architecture and preliminary implementation of our prefix-caching proxy on a Linux-based PC. We used a commercially available (RealNetworks G2) RTSP server and player as a source and sink for testing our proxy implementation. Section 5 concludes the paper with a discussion of future research directions. Appendix A describes the operation of the RealNetworks implementation of RTSP, including how to convince the RealServer to stream with RTP and RTCP. Appendix B shows a trace of RTSP messages between the RealNetworks client and server.

The paper complements recent work on multimedia caching and multimedia proxy services. Earlier research on multimedia caching proposed storing a sliding interval of successive frames to satisfy client requests that arrive close together in time [4–6] . Other recent work proposed having the proxy cache a fixed subset of frames, such as the prefix of the stream or a subset of frames in high-bandwidth regions of the stream, to reduce the overhead of transmitting to the client [1,7–10] . Our paper complements these studies by focusing on the protocol and systems issues that arise in caching partial contents. Other recent work has introduced multimedia proxy services that operate within existing streaming protocols, including support for transcoding, retransmission, workahead smoothing, and ad insertion [11–14] . We complement this work by focusing on the unique challenges of performing prefix caching at a proxy.

2 Multimedia Streaming Protocols

The RTP, RTCP, and RTSP suite of protocols supports multimedia streaming in the Internet, and serves as the starting point for our work on prefix caching.

2.1 Real-Time Transport Protocol (RTP)

RTP [15] provides the basic functionality for transferring real-time data over packet networks. Since RTP does not include mechanisms for reliable delivery or flow control, transport of RTP packets must rely on underlying protocols such as UDP and TCP. RTP typically runs over UDP, though TCP is sometimes used for reliable transport or to stream across firewalls that discard UDP packets. RTP provides source identification (randomly chosen SSRC identifier), payload type identification (to signal the appropriate decoding and playback information to the client), sequence numbering (for ordering packets and detecting losses), and timestamping (to control the playback time and measure jitter). Packets contain a generic RTP header with these fields, as well as payload-specific information to improve the quality of delivery for specific media (e.g., MPEG). Interpretation of some header fields, such as the timestamp, is payload dependent. The RTP header may also identify contributing sources (CSRCs) for the payload carried in the packet; such a list is typically inserted by mixers or translators.

2.2 Real-Time Control Protocol (RTCP)

RTCP [15] monitors the delivery of RTP packets. The protocol provides feedback on the quality of data distribution, establishes an identification (CNAME) for each participant, scales the control packet transmission with the number of participants (to avoid generating excessive feedback traffic in large multicast groups), and provides minimal session control information. RTCP packets consist of source descriptors, sender reports, receiver reports, BYE packets (signifying the end of participation in a session), and APP packets (for application-specific functions). Receiver reports include the stream's SSRC, the fraction of RTP packets lost, the sequence number of the last RTP packet received, and the packet interarrival jitter. Senders can use this information to modify their transmission rates or to switch to a different encoder. The sender reports include the stream's SSRC, the sequence number of the last RTP packet sent, the wallclock time of the last transmission, and the number of RTP packets and bytes sent. The client can use the RTP timestamp and wallclock time information for media synchronization.

2.3 Real-Time Streaming Protocol (RTSP)

RTSP [3] coordinates the delivery of multimedia files, acting as a "VCR remote control." RTSP typically runs over TCP, though UDP can also be used. Though conceptually similar to HTTP/1.1 [2] , RTSP is stateful and has a number of different methods. The OPTIONS method inquires about server capabilities (e.g., RTSP version number, supported methods, etc.), and the DESCRIBE method inquires about the properties of a particular file (e.g., Last-Modified time and session description information, typically using SDP [16]). Client and server state machines are created with the SETUP method, which also negotiates transport parameters (e.g., RTP over unicast UDP on a particular port). The client sends a SETUP message for each stream (e.g., audio and video) in the file. Streaming of the file is initiated with the PLAY method, which can include a Range header to control the playback point. The TEARDOWN method terminates the session, releasing the resources at the server and client sites. The protocol also includes a number of recommended or optional methods.

3 Prefix Caching: Protocol Issues

The prefix-caching proxy operates within the RTP, RTSP, and RTCP protocols, as illustrated in Figure 1. The figure highlights the case where the proxy has stored the prefix of each stream (e.g., audio and video) of a presentation, the description of the presentation, and the options supported by the server. The proxy responds directly to the client OPTIONS, DESCRIBE, SETUP, and PLAY requests, and initiates transmission of the prefix while simultaneously requesting the suffix from the server.

3.1 Basic RTSP Operation

The handling of client requests depends on whether or not the proxy has cached the prefix and the related RTSP information. The client uses the OPTIONS message to learn which methods are supported by the server. The server (and therefore the proxy) should support the basic methods such as OPTIONS, DESCRIBE, SETUP, PLAY, and TEARDOWN. The proxy can respond directly to the client's OPTIONS request, avoiding the delay in contacting the server, as shown in Figure 1. In particular, the proxy could respond with a list of methods that were cached during a previous interaction with the server. Alternatively, the proxy could respond with a default list of basic methods. In either case, the proxy may occasionally send an incorrect list of methods (e.g., if the server ceases to support some method). If the client later attempts to invoke the unsupported method, the server would simply respond with an error message (response code 405 "Method not allowed") that includes the current list of methods, which the proxy can cache and forward to the client. Note that this error would *not* cause streaming to fail.

The DESCRIBE message is used to retrieve information about the particular presentation, which consists of one or more underlying streams (e.g., audio, video). Like the OPTIONS message, the DESCRIBE message is optional and does not affect the RTSP state machine at the client or the server. If the description resides in the cache, the proxy can respond directly to the client; otherwise, the proxy must contact the server. However, the cached DESCRIBE information may not be up-to-date. The proxy can employ a variety of techniques to reduce the likelihood of stale information, as discussed in Section 3.3.

The SETUP request is used to negotiate the transport parameters, including the transport protocol (e.g., RTP) and the TCP/UDP port numbers. Upon receiving the client SETUP message, port numbers are generated for the proxy's end of the RTP and RTCP connections, and a session

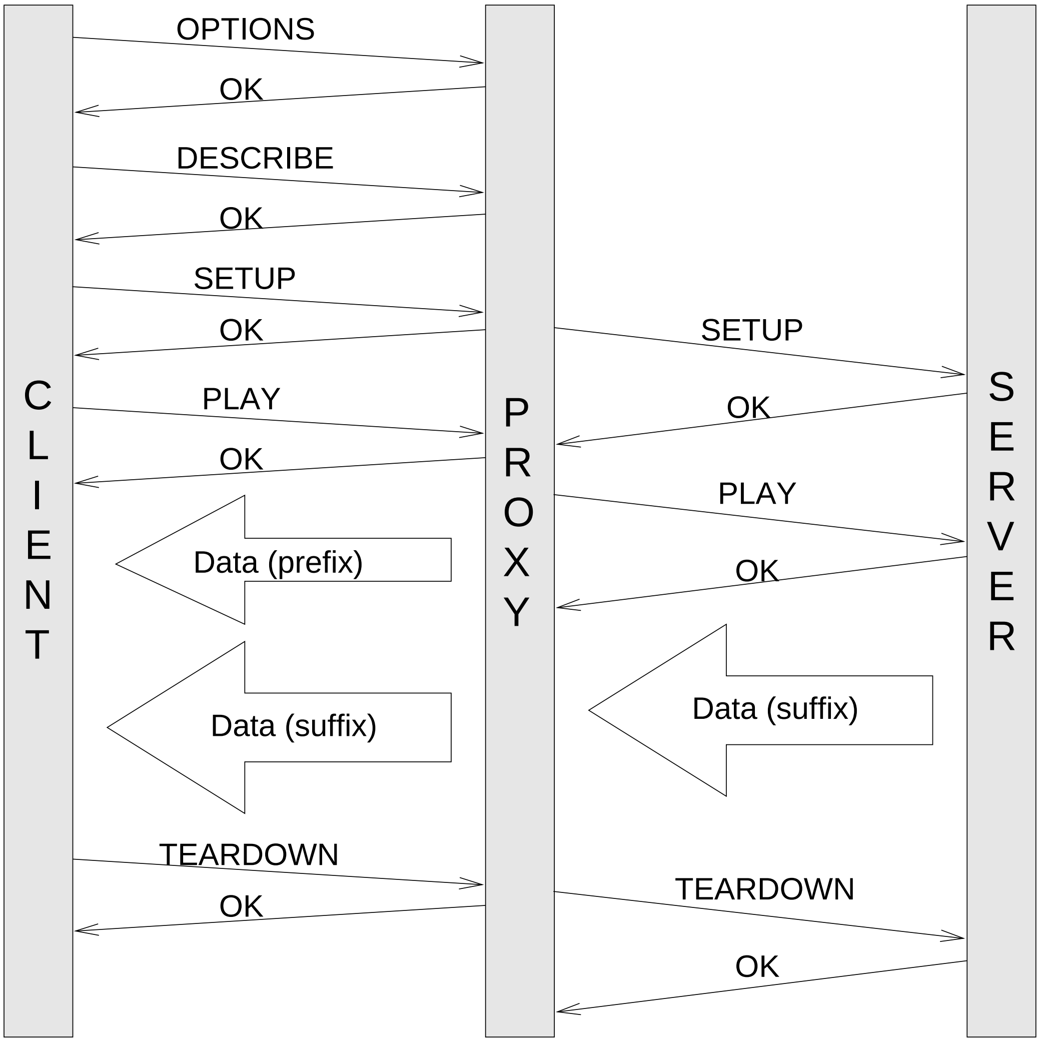

Figure 1: Protocol exchange under proxy prefix caching

identifier is selected. The port numbers and session identifier are sent to the client. In the meantime, the proxy generates a separate SETUP message to the server. Each stream in the multimedia presentation results in separate connections on both the proxy-client and server-proxy paths, and consist of TCP or UDP connections for RTP and RTCP. The proxy could request that the server stream the RTP packets over TCP (or a related semi-reliable transport protocol), since the cached prefix could hide transient periods of TCP congestion where the server does not send packets fast enough for continuous playback. To coordinate the transfer of RTP and RTCP messages, the proxy must maintain a mapping table to direct messages to the appropriate outgoing connection (and with the appropriate session identifier), as discussed in Section 4.2.

The client can send the PLAY message after receiving the proxy response to the SETUP request. If the prefix is cached, the proxy can respond immediately to the PLAY request and initiate the streaming of RTP and RTCP messages to the client. Note that the proxy may or may not have received the SETUP response from the server, depending on the delay on the server-proxy path. Once the proxy has received a SETUP response from the server, indicating the server's session identifier, the proxy can send the PLAY request to the server, with the appropriate Range header, as discussed in more detail in Section 3.2.

Depending on the size of the prefix, the proxy may decide to wait longer before asking the server to transmit the suffix. Fetching the suffix too early would require extra storage resources at the proxy, and may result in extra server load and network traffic if the client eventually pauses or stops the transfer of the presentation. Yet, the suffix should be requested sufficiently ahead of when the proxy would need to start transmitting this data to the client. Requesting the suffix a bit early allows the proxy to hide the server-proxy round-trip delay (which can be estimated from RTCP reports), or additional delays for other functions such as smoothing or transcoding.

3.2 Linking the Prefix and Suffix

When the entire stream resides in the cache, the proxy acts as a server in responding to the client PLAY request. When the cache does not contain any portion of the requested resource, the proxy forwards the PLAY request to the server and simply shuttles messages back and forth between the server and the client, acting as an application-level router. The operation of the proxy becomes more interesting when the cache stores only the prefix of the stream. In this case, the proxy can reply to the client PLAY request and initiate transmission of RTP and RTCP messages to the client, while requesting the suffix from the server. Fetching the suffix requires the proxy to initiate a Range request for the appropriate portion of the stream. As part of linking the prefix and suffix, the proxy must specify the appropriate Range, handle inaccuracies in the server response, and support seamless transmission of the RTP packets to the client.

As part of caching multimedia content, the proxy keeps track of the size of the prefix in terms of the timestamps in the RTP packets. The RTSP Range request is defined for both SMPTE Relative timestamps and NPT (Normal Play Time). SMPTE has the format hours:minutes:seconds:frames.subframes with the origin at the beginning of the clip. In contrast, NPT time consists of a decimal fraction, where the left part of the decimal may be expressed in either hours, minutes or seconds, and the right part of the decimal point measures fractions of a seconds. These formats require the proxy to convert the RTP timestamps into time, by applying the payload-specific clock rate [17] . According to the protocol specification, servers supporting the Range header *must* understand the NPT range format and *should* understand the SMPTE range format. In most cases, the SMPTE format is preferable, since it supports more indexing into individual packets within a frame. But, the proxy could easily avoid the need for such fine-grain indexing by ending the prefix at frame boundaries.

The proxy does not know in advance whether or not the server supports Range requests for a particular presentation (some streams do not allow seek operations), and whether or not SMPTE format is supported. The proxy can learn, and cache, this information for each presentation by sending Range requests to the server and noting the response (e.g., the server sends a 426 "Header Field Not Valid for Resource" response when Range is not supported). The proxy could avoid polling the server for this information if the RTSP SETUP or PLAY response included information about support for Range requests. To avoid changing existing RTSP implementations, providing the information about Range support could be optional, since the proxy could always infer the information and/or decline to cache partial contents when the information is not provided.

In addition, RTSP does not require the server to handle the Range request precisely. For example, the request may identify a starting time that does not correspond to any particular frame, forcing the server to initiate transmission from the previous or subsequent frame. In addition, the server may not support arbitrary indexing to individual frames[1]. The RTSP protocol does not dictate how accurately the server must handle the Range header, and does not provide a way for the server to indicate in advance how much inaccuracy could be introduced. As such, the proxy should be conservative, and ensure that the transmission to the client does not have duplicate or missing packets. Fortunately, the server reply includes a Range response header that indicates what range of time is currently being played. If the beginning of the suffix contains frames that are already at the end of the prefix, the proxy discards the overlapping packets. To avoid a gap between the prefix and the suffix, the proxy could initiate a conservative Range request, or issue a second request if a gap arises.

In order to link the prefix and the suffix, all RTP headers must be consistent, otherwise the client will not associate the two parts to the same stream. The sensitive fields are sequence numbers, timestamps, and source identifier (SSRC), which have been selected separately by the proxy (for the prefix) and the server (for the suffix). In streaming the suffix, the proxy overwrites the SSRC field in each RTP packet with the value it selected as part of initiating transmission of the prefix. The proxy knows the timestamp and sequence number used in transmitting the last RTP packet of the prefix. The base timestamp and sequence number for the server's transmission of the suffix are provided in the RTP-Info header in the PLAY response. The proxy can then add/subtract the appropriate constant for the timestamp and sequence number fields of each packet in the suffix.

3.3 Cache Coherency

Caching RTSP messages and multimedia content at a proxy introduces potential coherency problems when the server changes this information. Cache coherency is a critical issue in the Web, where many resources change very frequently. The problem is arguably less critical for multimedia presentations, which may change less frequently – the server could simply assign a new URL when new content is created. Still, the paradigms for authoring and updating multimedia content are not well understood, and the use of proxies should not place additional restrictions on how servers are managed. As such, the proxy should avoid sending a suffix that does not come from the same version of the presentation as the cached prefix.

The proxy could reduce the likelihood of introducing an inconsistent stream by sending a DESCRIBE message to the server for every client request, and checking that the Last-Modified time matches the cached prefix. However, the additional exchange with the server would increase client start-up latency, which would partially defeat the purpose of prefix caching. Instead, the proxy could apply a timeout (time-to-live) to the cached information, revalidating (with an "If-Modified-Since" request) only if the timeout has expired. To avoid forcing the client to wait for the proxy

[1]For example, an MPEG stream typically consists of I, P, and B frames within a group of pictures (GOP). The server may not precisely satisfy a Range request that starts in the middle of a GOP. In fact, allowing clients to index to any arbitrary frame could introduce substantial overhead at the server (to maintain fine-grain indices, or to parse the content to sequence to the appropriate frame), particularly for variable-bit-rate streams, or streams without a fixed GOP structure.

to check with the server, the proxy could periodically validate the cached information, even if no client has requested the stream. To reduce the overhead of creating a TCP connection to the server, the proxy could freshen the DESCRIBE information during other communication with this server (e.g., while coordinating the transmission of another stream from the same server). Such piggyback cache validation has proved to be a very effective technique for controlling the consistency of Web resources [18] .

To ensure that a client *never* receives a suffix that does not match with the prefix, the proxy could send a DESCRIBE request to the server just after requesting the suffix. This would allow the proxy to verify that the suffix comes from the same presentation as the cached prefix. In the rare event that the suffix comes from a more recent version of the presentation, the proxy could terminate the transmission to the client rather than send inconsistent information. Note that this approach does *not* delay transmission to the client in the common case when the presentation has not changed. From the client's point of view, the disruption of the transmission would appear as a transient server error. For example, the proxy could send a "500 Internal Server Error," "502 Bad Gateway," or "503 Service Unavailable" message. Note that the proxy could also send such an error message if the server does not respond to the proxy's request for the suffix, or is unable to devote resources to satisfying the proxy's request. After receiving the error message from the proxy, the client could optionally initiate the request a second time, and receive a fresh copy of the presentation from the server.

Finally, cache inconsistency can be avoided with additional support from the server. The simplest approach would be to have the server indicate an expiration time for the presentation, and ensure that the contents will not change in the meantime. An alternative solution would be to allow additional communication between the server and the proxy. For example, the server could push the prefix of each popular multimedia presentation to interested proxies, perhaps using multicast to distribute the content. In this case, the proxy acts as a (partial) replica of the server content, rather than as a conventional cache. The server could indicate the expiration time for the presentation, or promise to send updates as part of changing the presentation. The proxies could subscribe to receive updates for particular content of interest.

3.4 Relaying RTCP Sender/Receiver Reports

Prefix caching introduces additional challenges in generating accurate and timely reports of server and client performance. If the entire stream resides in the cache, the proxy acts as a server in generating sender RTCP reports and responding to the client's receiver reports. On a cache miss, the proxy acts as an application-level router and simply forwards the reports generated by the server and the client. Yet, the appropriate behavior is less clear when the proxy sends the prefix of the stream and fetches the suffix from the server. On the one hand, the proxy could view the server-proxy and proxy-client paths as distinct, and have two unrelated RTCP sessions with the server and the client, respectively. Despite the conceptual appeal of this approach, the proxy runs the risk of hiding important information about performance problems on the server-proxy or proxy-client paths. Instead, the proxy could generate and receive reports while transmitting the prefix, and switch to forwarding reports while transmitting the suffix. In forwarding the server-generated reports, the proxy would need to map the key fields, such as the SSRC, timestamp, and sequence number, that have a different base in the suffix than in the prefix.

Forwarding server and client RTCP reports during transmission of the suffix has the advantage of informing the server about the performance experienced by the client. However, the information could be rather out of date, since the suffix is not sent to the client until transmission of the prefix is complete. The proxy has a number of possible options to control the RTP reports sent to the server. Reporting transient performance problems (e.g., burst of loss) on the proxy-client path may not be particularly relevant, especially if this information is out-of-date. Instead, the proxy could choose to report only persistent problems that require a response by the server (e.g., to switch to a lower bandwidth version of the stream). In generating these reports, the proxy must be careful to reflect the delay that has been introduced in sending the RTP packets in the suffix to the client. Otherwise, the server may think that a large number of packets have not been received. The proxy could achieve these goals by basing the receiver reports on the packets that have been received by the proxy (even if they have not all been sent to the client), while using the loss rate experienced by the client to compute the number of lost packets to report to the server.

Correctly modifying the RTCP receiver reports is a delicate process. Instead, the proxy could take advantage of the cached prefix to change the properties of the server-proxy and proxy-client paths. The proxy could take advantage of the prefix cache to hide the delay for explicit retransmission of lost packets on the server-proxy path. For example, the proxy could issue a Range request (perhaps on a separate connection) to retrieve a previously-requested interval of frames from the server. This would isolate the client from the loss on the server-proxy path. Rather than issuing RTSP Range requests (which is arguably a rather coarse-grain way to effect packet retransmissions), the proxy could have the server transmit RTP packets over a *TCP* connection (by requesting TCP in the initial SETUP message). The TCP connection would ensure a reliable transmission of the RTP packets, and would obviate the need for an RTCP session between the server and the proxy. The proxy could still use UDP for transfers to the client.

To address potential performance problems on the proxy-client path, the proxy could perform additional functions, such as transcoding and workahead smoothing. Transcoding [11] would reduce the bandwidth requirements of the stream by lowering the quality. The proxy could perform transcoding either by discarding a subset of the incoming packets, or by reencoding the stream (e.g., by discarding high-frequency components in each MPEG slice). In contrast, workahead smoothing would involve transmitting frames into the client playback buffer ahead of the playout time [13] . The cached prefix allows the proxy to transmit frames ahead of time without incurring additional client playback latency. The transmission of additional frames would allow time for slower transmission of large frames (smoothing out a burst) or retransmission of lost packets (based on the receiver reports). The proxy could perform these functions on the without involving the server. If the proxy sends RTCP receiver reports to the server, these messages should reflect the improvements made in the proxy-client path, to avoid triggering conflicting recovery operations at the server.

4 Prefix Caching: Architecture

This section describes the architecture of the proxy, including the main design decisions, the per-session state, and the cache operation. A preliminary implementation of the prefix-caching proxy has been implemented in C on a Linux-based PC, and successfully tested using the RealServer and RealPlayer (from `www.real.com`) as a source and sink.

4.1 Main Design Decisions

We first consider two basic design decisions that determine how client requests are directed to the proxy, and how the proxy allocates resources between client sessions. The client sends RTSP messages directly to the proxy by configuring the player with the proxy's IP address. The client has a TCP connection to/from the proxy for RTSP operations, a UDP connection from the proxy to receive RTP packets, and a UDP connection to/from the proxy for RTCP. Having the client contact the proxy explicitly has advantages over transparent caching, where the proxy must lie in the path of client messages that are directed to the server. Transparent caching would also introduce extra complexity and overhead for the proxy to intercept and modify all of the client and server packets, including reconstruction of the TCP stream for the RTSP messages.

To handle requests from multiple clients, the proxy must have an effective way to allocate memory, processing, and I/O resources between different sessions. Our prototype implementation has an event-driven architecture, where the proxy coordinates all activity between client-server pairs in a single process. This avoids the overheads (and jitter) introduced by context switching between the processes, and simplifies the sharing of the cache amongst all of the sessions. In addition, the event-driven architecture enables the proxy to have greater control over resource allocation, by sending and receiving RTP data for each session in proportion to the bit-rate requirements. Instead of having a process scheduler implicitly allocate resources to each session, the event-driven approach shares resources by scheduling socket operations, as discussed below in Section 4.4. The proxy can timeshare the system by performing accounting of the number of bytes handled for each stream, relative to the time that these bytes are due to the client.

4.2 Per-Session State

The proxy coordinates communication with the client and the server using a data structure that stores key information about the ongoing session. The fields in this data structure are populated as the session progresses. A session can include multiple streams, simultaneously or back-to-back; for example, a video transmission typically includes both an audio and a video stream, and perhaps a third stream for subtitles. The reply to the OPTIONS message contains cacheable information about the server, but does not include any session-level information. The reply to the DESCRIBE message includes cacheable information about a resource, such as its URI and Last-Modified Time. For illustration, we describe how the session data structure is populated on a cache miss:

- Client RTSP connection: Upon establishing a TCP connection with the client, the proxy records the client IP address and the identifier for the TCP socket.
- First client RTSP request: The first RTSP message could be an optional OPTIONS or DESCRIBE message, or a SETUP message. The proxy extracts the server name and performs a gethostbyname() to learn the server IP address. The proxy establishes a TCP connection to the server and stores the identifier for the TCP socket.
- Client SETUP request: The client SETUP message(s) includes information about the transport protocol and port numbers that the client wishes to use (e.g., for RTP and RTCP). After binding the two UDP sockets, the proxy records the two sets of client and proxy port numbers, and replies to the client. In the meantime, the proxy generates and records the port numbers for its UDP connections with the server, and sends a SETUP message to the server.
- Server SETUP response: After receiving the server reply, the proxy records the server port numbers for the UDP connections and binds the UDP sockets. The proxy also stores the session identifier, generated by the server.
- Client TEARDOWN request: The proxy closes the sockets to the client, forwards the TEARDOWN to the server, and deletes the session data structure.

The session structure also includes a number of flags that indicate whether or not the requested streams are in the cache, and which packet should be sent next. The session data structure is populated in a similar manner when the proxy cache contains the prefix, except that the session identifier, base timestamp, and base sequence number are chosen by the proxy.

4.3 Caching Architecture

The desire to efficiently transmit cached multimedia content to a large collection of clients drives most of the design decisions for the caching architecture. For example, the proxy caches RTP packets, rather than raw multimedia content. This obviates the need to parse the body of the RTP packets and repacketize the content, at the expense of additional storage overhead for RTP headers. Each item in the cache represents a particular multimedia resource, including the DESCRIBE information and a linked list of RTP packets, ordered by sequence number. In the simplest case, the incoming RTP packets are appended to the linked list as they arrive. Since packets sometimes arrive out of order, the proxy could insert each packet in the appropriate place in the list based on the sequence number field in the RTP packet header.

Each time a stream is transmitted from the cache, the proxy must select an SSRC and a base timestamp and sequence number (at random). As such, the proxy must alter these header fields in the cached RTP packets before transmitting to the client. To avoid extra data copying, the proxy can send the packet using a *gather* function (e.g., writev()) that can combine buffers from multiple locations. However, not all systems have efficient support for the gathering function. In this case, the proxy can overwrite the RTP header fields directly in the cache, and initiate a regular socket send operation for the entire packet. To avoid alteration or deletion of the packet during the copy into the socket buffer, each packet includes a lock bit to indicate that a (non-blocking) transmission is in progress; the lock bit is cleared after completion of the socket call. Also, since the proxy overwrites certain RTP header fields, the cache stores additional information to allow computation of the sequence number and timestamp fields for subsequent transmissions of the packet.

4.4 Scheduling RTP Packets

The proxy, like a server, must carefully schedule transmission of RTP packets to avoid underflow and overflow at the client site. If the proxy sends packets to the client over a TCP connection, the TCP flow control could implicitly pace the transmission. However, sending UDP packets requires greater care. The proxy has several ways to learn the appropriate sending rate. For example, the server response to the DESCRIBE message typically includes the stream's target bandwidth. The bandwidth value could be used to coordinate packet transmission, particularly if the proxy has kernel or hardware support for traffic shaping. However, the bandwidth parameter represents an average or maximum value, which may not be representative of any particular portion of the stream, especially for content with variable-bit-rate encoding[2].

Instead, the proxy schedules packets based on the RTP timestamps and packet sizes. The timestamp clock rate is payload dependent [17] , requiring the proxy to maintain a mapping between payload types and clock rates to determine how many packets to transmit during a given time interval. Ideally, the proxy could schedule the transmission of each packet at the appropriate departure time. However, such fine-grain scheduling is typically not possible, and is arguably not necessary. Instead, the proxy should pick an appropriate time interval (say, one second) for coordinating packet transmissions for each session. When streaming multimedia content to multiple clients, the proxy proxy can sequence through the set of sessions, transmitting the appropriate number of packets for each active session. A system with kernel support for multimedia streaming could explicitly support packet scheduling [19] . Ideally, in addition to allowing per-session scheduling and variable transmission rates, the kernel could conceivably compute the appropriate rate directly from the RTP timestamps, payload-specific clock rate, and packet sizes.

[2]Although the bandwidth parameters in the DESCRIBE response are not appropriate for coordinating fine-grain packet scheduling, this information could be used to determine if the proxy has sufficient resources to support the additional stream.

4.5 Implementation Status

The initial prototype of proxy prefix caching implements a subset of the architecture, along with the modifications necessary for interacting with the RealNetworks server and player. Using the basic implementation, we can successfully process client RTSP messages, requesting the full contents from the server on a cache miss, or partial contents when a portion of the stream resides in the cache. The playback at the client is seamless in both cases; the continuity of the RTP stream was also verified by packet inspection. The proxy can sustain continuous transmission to multiple clients.

The prototype is sufficient to test our uses of the RTP and RTSP protocols. However, supporting the full use of the protocols and scaling to a large number of simultaneous sessions would require enhancements to our implementation. The current prototype transfers RTSP messages over TCP, and RTP and RTCP messages over UDP, and assumes that a session has at most two simultaneous streams. The proxy forwards RTCP messages between the client and server, but does not generate reports itself. For simplicity, transfers of RTP packets to the client use a blocking send call. Future work can focus on extending the implementation and evaluating the performance.

5 Conclusions

Prefix caching offers an effective way for Internet service providers to improve user performance and network efficiency, without requiring additional support from the client, the server, or the network. The service can operate within existing standard protocols. Still, the service would benefit from improvements in the capabilities exchange in RTSP to provide more upfront information about the supported submethods, such as the Range header and acceptable time representations. In addition, prior knowledge of a bound on the server's inaccuracy in satisfying Range requests would be useful. These features could be added to RTSP, or supported in an alternate protocol with similar functionality.

As part of our ongoing work, we are pursuing these possible extensions to the RTSP standard, and are interacting with RealNetworks to resolve the RTSP interoperability issues with the G2 player. We are also studying the cache coherency and RTCP feedback issues in greater detail. These issues become more complicated when the server can dynamically adjust the quality of the media stream in response to network congestion or server load. In addition, we are investigating how to combine prefix caching with other multimedia proxy services, such as workahead smoothing, retransmission, and transcoding.

A RealNetworks Server and Client

In this section, we describe how the proxy modifies RTSP messages from the RealPlayer to make the RealServer stream multimedia content using RTP and RTCP packets, rather than the proprietary RDT transport protocol, and highlight how the RTSP messages differ from the standard protocol.

A.1 Testbed Configuration

Testing the prototype proxy requires multimedia clients and servers that implement RTP, RTCP, and RTSP. Currently, reference RTSP software is available for a source and a sink. However, the reference source cannot support multiple simultaneous clients, and the reference client does not display the multimedia stream; in addition, the reference code implements an earlier version of RTSP. Rather than extending the reference code, we decided to experiment with the RealNetworks server and

```
SETUP rtsp://135.207.15.90:554/g2video.rm/streamid=0 RTSP/1.0
CSeq: 3
RealChallenge2: 7c7687849ca62b5a063fd037f9d13d7a01d0a8e3, sd=78920df3
Transport: x-real-rdt/udp;client_port=6970;mode=play,
           x-pn-tng/udp;client_port=6970;mode=play,
           rtp/avp;unicast;client_port=6970-6971;mode=play
If-Match: 31278-2
```

Figure 2: Trace of RealPlayer SETUP message

player, which both support RTSP/1.0. This provided an opportunity to test the interoperability of our proxy with the dominant Internet multimedia components, and to study the RealNetworks RTSP implementation.

Our testbed consists of a Windows NT 4.0 PC running a trial version of RealServer G2 (version 6.0.3.353), and a Windows 95 PC running RealPlayer G2 (version 6.0.3.143). The RealPlayer is configured with the IP address and port number of the proxy. The decision to experiment with a local copy of the G2 server, rather than with existing multimedia servers in the Internet, was motivated by several issues. First, AT&T Research has a firewall that filters UDP packets. This would force RTP and RTCP transfers to use TCP, whereas we wish to experiment with UDP. Later, we could conceivably put the proxy outside of the firewall, or experiment with RTP and RTCP transfers over UDP. Second, our initial implementation operates on a private Ethernet, to prevent accidental generation of excess traffic on the local area network. As such, the proxy does not have direct access to the larger Internet.

A.2 RTSP Operation

Working with the RealNetworks client and server introduced three main challenges – convincing the RealServer to transmit RTP and RTCP, convincing the RealPlayer to accept transfers directly from the proxy, and judging the accuracy of the server's response to the Range request. Although the RealServer can generate RTP and RTCP packets, the server typically employs a propriety transport protocol called RDT. The selection of the transport protocol is initiated by the client SETUP message. For example, the example SETUP message in Figure 2 shows three candidate transport protocols, starting with RDT. Upon receiving this message, the server selects RDT as the transport protocol. To convince the server to send RTP and RTCP packets, the proxy modifies the Transport header to include only the RTP protocol. Then, the server selects RTP as the transport protocol, and transmits RTP and RTCP messages after receiving the PLAY request. Modifying the SETUP message is sufficient to convince the server to use RTP for most encoding formats (e.g., .wav, .au, and .avi files). However, the RealServer will not send RealNetworks files (e.g., .ra or .rm) using RTP.

Exploiting the benefits of prefix caching also requires the client to accept RTSP messages directly from the proxy, without communication with the server. The RealNetworks documentation claims that the RealPlayer should be able to communicate with a standards-compliant server. In experiments with our proxy, the RealPlayer accepts the responses to the OPTIONS and DESCRIBE methods, and generates a SETUP message. However, after receiving the response to the SETUP message, the RealPlayer generates a pop-up window with the error "The connected server is not valid!" and does not send any further RTSP messages[3]. We are currently interacting with someone at RealNetworks to resolve this apparent interoperability problem. In the meantime, the prototype

[3] To verify that the problem does not arise from a mistake in our RTSP implementation, we also experimented with RTSP messages generated by the RealServer. The test involved a simplified proxy that simply shuttled messages

proxy forwards OPTIONS, DESCRIBE, and SETUP packets to the server in order to generate appropriate responses (including non-standard headers, such as encrypted challenge fields) to the client. The actual multimedia content can still be streamed directly from the proxy cache.

Prefix caching also depends on server support for the Range header in the PLAY request. Although the RealServer does support Range requests, the Range response header is missing from the server's reply to the PLAY request. As such, the server does not indicate what range is actually played, making it impossible for the proxy to detect overlaps or gaps with the prefix stored in the cache. Experimentation with the range request suggests that the server satisfies the request exactly, for the payload types we have tested so far. As such, the proxy currently assumes, in the absence of a Range response header, that the Range request was satisfied exactly.

B Typical RealNetworks RTSP Transfer

In this appendix, we present a trace of an exchange between a RealNetworks client and server, with no intervening proxy.

B.1 OPTIONS

The client sends an OPTIONS request to learn about the capabilities of the server, and the server replies with a list of supported methods.

B.1.1 Client OPTIONS Request

```
OPTIONS rtsp://135.207.15.90:554 RTSP/1.0
CSeq: 1
User-Agent: RealMedia Player Version 6.0.3.177 (win32)
ClientChallenge: e62af50b36f6bc9989ae8daaa85532b6
PlayerStarttime: [15/03/1999:15:55:15 00:00]
CompanyID: 2euF/n8Rod7IBCnGxX2DAg==
GUID: 0bc05cc0-d248-11d2-9566-0000f4a40b9e
RegionData:
ClientID: Win98_4.10_6.0.3.143_play32_PN01_en-US_586
```

Only the first two lines of the message are necessary in a standard RTSP OPTIONS request. The remaining lines provide information about the RealNetworks player (version number, user identifier, etc.) and an encrypted client challenge.

B.1.2 Server OPTIONS Response

```
RTSP/1.0 200 OK
CSeq: 1
Date: Mon, 15 Mar 1999 20:56:17 GMT
Server: RealMedia Server Version 6.0.3.352 (win32)
Public: OPTIONS, DESCRIBE, ANNOUNCE, SETUP, GET_PARAMETER,
```

between the RealPlayer and RealServer, and removed any non-standard RTSP header fields. Hence, the server received requests from a standard RTSP/1.0 client, and the client received responses from a standard RTSP/1.0 server. Yet, the RealPlayer still generates the same error, at the same point in the RTSP exchange.

```
SET_PARAMETER, TEARDOWN
RealChallenge1: d803996050e8b18e49ac69e19d6ebfa9
StatsMask: 3
```

Only the first four lines of the message are standard. The last two lines are particular to the RealNetworks implementation and appear to continue an ongoing challenge-response between the server and the player.

B.2 DESCRIBE

The client sends a DESCRIBE message to learn about the desired presentation, and the server responds with information in SDP format [16] .

B.2.1 Client DESCRIBE Request

```
DESCRIBE rtsp://135.207.15.90:554/g2video.rm RTSP/1.0
CSeq: 2
Accept: application/sdp
Bandwidth: 10485800
GUID: 0bc05cc0-d248-11d2-9566-0000f4a40b9e
RegionData:
ClientID: Win98_4.10_6.0.3.143_play32_PN01_en-US_586
Language: en-US, en, *
Require: com.real.retain-entity-for-setup
```

Only the first three lines are standard. The remaining header fields provide information about client bandwidth capabilities and various client identifiers.

B.2.2 Server DESCRIBE Response

```
RTSP/1.0 200 OK
CSeq: 2
Date: Mon, 15 Mar 1999 20:56:17 GMT
Last-Modified: Tue, 23 Feb 1999 22:51:44 GMT
X-TSPort: 7802
Content-base: rtsp://135.207.15.90:554/g2video.rm/
ETag: 31278-2
Content-type: application/sdp
Content-length: 3226

v=0
o=- 919810304 919810304 IN IP4 135.207.15.90
s=G2 Video Experience
i=RealNetworks (c)1998
a=Flags:integer;2
a=IsRealDataType:integer;1
a=StreamCount:integer;2
a=Title:buffer;"RzIgVmlkZW8gRXhwZXJpZW5jZQA="
a=Copyright:buffer;"qTE5OTgA"
```

```
a=Author:buffer;"UmVhbE5ldHdvcmtzAA=="
a=ASMRuleBook:string;"#($Bandwidth < 42000),Stream0Bandwidth = 6000,
Stream1Bandwidth = 27600;#($Bandwidth >= 42000) && ($Bandwidth <
56000),Stream0Bandwidth = 6000, Stream1Bandwidth = 36000;#($Bandwidth
>= 56000),Stream0Bandwidth = 6000, Stream1Bandwidth = 50000;"
t=0 0
m=audio 0 RTP/AVP 101
a=control:streamid=0
a=rtpmap:101 x-pn-realaudio
a=length:npt=59.773000
a=mimetype:string;"audio/x-pn-realaudio"
a=MinimumSwitchOverlap:integer;200
a=StartTime:integer;0
a=AvgBitRate:integer;6000
a=AvgPacketSize:integer;288
a=SeekGreaterOnSwitch:integer;0
a=Preroll:integer;4608
a=MaxPacketSize:integer;288
a=MaxBitRate:integer;6000
a=RMFF 1.0 Flags:buffer;"AAQAAgAAAAIAAA=="
a=OpaqueData:buffer;"TUxUSQAEAAEAAQAAAAAAAAgAAAFYucmH9AAUAAC5yYTUAAAAAAAA
UAAABGAAgAAAEgAACvgAAAr8gAAK/IAAYBIAAYAAAAAB9AAAAfQAAAABAAAWdlbnJjb29rA
QIAAAAAAAAgBAAACAQAACQAAAFYucmH9AAUAAC5yYTUAAAAAAAAUAAABGAAgAAAEgAACvgAAA
r8gAAK/IAAYBIAAYAAAAAB9AAAAfQAAAABAAAWdlbnJjb29rAQIAAAAAAAAgBAAACAQAACQ=="
a=StreamName:string;"audio/x-pn-multirate-realaudio logical stream"
a=ASMRuleBook:string;"#($OldPNMPlayer),AverageBandwidth=6000,priority=5
,PNMKeyframeRule=T;#($OldPNMPlayer),AverageBandwidth=0,priority=5,PNMNo
nKeyframeRule=T;AverageBandwidth=6000,Priority=5;AverageBandwidth=0,Pri
ority=5;"
m=video 0 RTP/AVP 101
a=control:streamid=1
a=rtpmap:101 x-pn-realvideo
a=length:npt=57.333000
a=mimetype:string;"video/x-pn-realvideo"
a=MinimumSwitchOverlap:integer;0
a=StartTime:integer;0
a=AvgBitRate:integer;50000
a=AvgPacketSize:integer;538
a=SeekGreaterOnSwitch:integer;1
a=Preroll:integer;5707
a=MaxPacketSize:integer;607
a=MaxBitRate:integer;50000
a=RMFF 1.0 Flags:buffer;"AAoAAgAAAAAAAAgACAAAAAgAAAAIAAA=="
a=OpaqueData:buffer;"TUxUSQAKAAMAAwAEAAUAAgACAAEAAQAAAAAAABgAAACIAAAAiVk
lET1JWMjAA8AC0AAwAAAAAAAB2AAAAAiDggEAABAAAAIgAAACJWSURPUlYyMADwALQADAAAA
AAAHYAAAACIOCAQAAEAAAAiAAAAIlZJRE9SVjIwAPAAtAAMAAAAAAAAdgAAAAIg4IBAAAQAA
ACIAAAAiVklET1JWMjAA8AC0AAwAAAAAAAB2AAAAAiDggEAABAAAAIgAAACJWSURPUlYyMAD
wALQADAAAAAAAAHYAAAACIOCAQAAEAAAAiAAAAIlZJRE9SVjIwAPAAtAAMAAAAAAAAdgAAAAI
g4IBAAAQ=="
a=StreamName:string;"video/x-pn-multirate-realvideo logical stream"
a=ASMRuleBook:string;"#(($Bandwidth >= 50000) &&
($OldPNMPlayer)),AverageBandwidth=50000,priority=9,PNMKeyframeRule=T;#(
```

```
($Bandwidth >= 50000) &&
($OldPNMPlayer)),AverageBandwidth=0,priority=5,PNMNonKeyframeRule=T;#((
$Bandwidth < 50000) &&
($OldPNMPlayer)),TimestampDelivery=T,DropByN=T,priority=9,PNMThinningRu
le=T;#($Bandwidth <
27600),TimestampDelivery=T,DropByN=T,priority=9;#($Bandwidth >= 27600)
&& ($Bandwidth < 36000),AverageBandwidth=27600,Priority=9;#($Bandwidth
>= 27600) && ($Bandwidth <
36000),AverageBandwidth=0,Priority=5,OnDepend=\"4\";#($Bandwidth >=
36000) && ($Bandwidth <
50000),AverageBandwidth=36000,Priority=9;#($Bandwidth >= 36000) &&
($Bandwidth <
50000),AverageBandwidth=0,Priority=5,OnDepend=\"6\";#($Bandwidth >=
50000),AverageBandwidth=50000,Priority=9;#($Bandwidth >=
50000),AverageBandwidth=0,Priority=5,OnDepend=\"8\";"
```

The ETag header appears to be (mis)used to convey a session identifier. The associated ETag value appears later in the SETUP message. In theory, an RTSP server does not establish session state until the SETUP message, since the OPTIONS and DESCRIBE messages do not affect the state machine and could be sent (or not sent) at any time. The RealNetworks client, though, always sends these two messages and uses information from the server responses in subsequent RTSP requests.

B.3 SETUP

The client sends a SETUP message for each stream in the presentation, and the server responds with information about the transport method (RDT) and the associated UDP port numbers.

B.3.1 Client SETUP Request (Stream 0)

```
SETUP rtsp://135.207.15.90:554/g2video.rm/streamid=0 RTSP/1.0
CSeq: 3
RealChallenge2: 7c7687849ca62b5a063fd037f9d13d7a01d0a8e3, sd=78920df3
Transport: x-real-rdt/udp;client_port=6970;mode=play,x-pn-
tng/udp;client_port=6970;mode=play,rtp/avp;unicast;client_port=6970-
6971;mode=play
If-Match: 31278-2
```

Note the use of the If-Match field to convey a string that was previously returned in the ETag header of the server's DESCRIBE message. Also, the client request contains another encrypted challenge header.

B.3.2 Server SETUP Response (Stream 0)

```
RTSP/1.0 200 OK
CSeq: 3
Date: Mon, 15 Mar 1999 20:56:17 GMT
Session: 31278-2
RealChallenge3: 1eeb464049ce5f71a672c758fc7679244f213d09,sdr=1445acf7
Transport: x-real-rdt/udp;client_port=6970;server_port=10276
```

Again, note the string in the Session header, and the use of an encrypted challenge header.

B.3.3 Client SETUP Request (Stream 1)

```
SETUP rtsp://135.207.15.90:554/g2video.rm/streamid=1 RTSP/1.0
CSeq: 4
Transport: x-real-rdt/udp;client_port=6970;mode=play
Session: 31278-2
```

The second SETUP message concerns the second stream of the presentation and does not include the challenge header.

B.3.4 Server SETUP Response (Stream 1)

```
RTSP/1.0 200 OK
CSeq: 4
Date: Mon, 15 Mar 1999 20:56:17 GMT
Session: 31278-2
Transport: x-real-rdt/udp;client_port=6970;server_port=10276
```

As before, the server selects a method of transport and the associated UDP port numbers.

B.4 SET_PARAMETER

The client appears to use the SET_PARAMETER method to prepare for future adaptations to the transmission rate.

B.4.1 Client SET_PARAMETER Request

```
SET_PARAMETER rtsp://135.207.15.90:554/g2video.rm RTSP/1.0
CSeq: 5
Subscribe:
stream=0;rule=2,stream=0;rule=3,stream=1;rule=8,stream=1;rule=9
Session: 31278-2
```

B.4.2 Server SET_PARAMETER Response

```
RTSP/1.0 200 OK
CSeq: 5
Date: Mon, 15 Mar 1999 20:56:17 GMT
Session: 31278-2
```

B.5 PLAY

The client initiates playback of the presentation with a single PLAY request, and the server responds.

B.5.1 Client PLAY Request

```
PLAY rtsp://135.207.15.90:554/g2video.rm RTSP/1.0
CSeq: 6
Session: 31278-2
Range: npt=0-59.773000
```

Note the use of NPT format for the Range header.

B.5.2 Server Play Response

```
RTSP/1.0 200 OK
CSeq: 6
Date: Mon, 15 Mar 1999 20:56:17 GMT
RTP-Info:
url=rtsp://135.207.15.90:554/g2video.rm/streamid=0;seq=0;rtptime=0,
url=rtsp://135.207.15.90:554/g2video.rm/streamid=1;seq=0;rtptime=0
```

After sending this message, RTP packets begin to flow from the server to the client on two UDP sessions. Note that both streams have an initial sequence number and timestamp of 0. These should be set to random values.

B.6 SET_PARAMETER

The client initiates a SET_PARAMETER request, presumably to adjust the server's delivery bandwidth, and the server responds.

B.6.1 Client SET_PARAMETER Request

```
SET_PARAMETER rtsp://135.207.15.90:554/g2video.rm RTSP/1.0
CSeq: 7
SetDeliveryBandwidth: Bandwidth=75920;BackOff=0
Session: 31278-2
```

B.6.2 Server SET_PARAMETER Response

```
RTSP/1.0 200 OK
CSeq: 7
Date: Mon, 15 Mar 1999 20:56:17 GMT
Session: 31278-2
```

References

[1] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE INFOCOM*, April 1999.

[2] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," *Request for Comments 2616*, June 1999.

[3] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)," *Request for Comments 2326*, April 1998.

[4] M. Kamath, K. Ramamritham, and D. Towsley, "Continuous media sharing in multimedia database systems," in *Proc. International Conference on Database Systems for Advanced Applications*, April 1995.

[5] A. Dan and D. Sitaram, "Multimedia caching strategies for heterogeneous application and server environments," *Multimedia Tools and Applications*, vol. 4, pp. 279–312, May 1997.

[6] R. Tewari, H. M. Vin, A. Dan, and D. Sitaram, "Resource-based caching for Web servers," in *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[7] Y. Wang, Z.-L. Zhang, D. Du, and D. Su, "A network conscious approach to end-to-end video delivery over wide area networks using proxy servers," in *Proc. IEEE INFOCOM*, April 1998.

[8] R. Rejaie, M. Handley, H. Yu, and D. Estrin, "Proxy caching mechanism for multimedia playback streams in the Internet," in *Proc. International Web Caching Workshop*, March 1999.

[9] D. Eager, M. Ferris, and M. Vernon, "Optimized regional caching for on-demand data delivery," in *Proc. Multimedia Computing and Networking*, January 1999.

[10] Z. Miao and A. Ortega, "Proxy caching for efficient video services over the Internet," in *Proc. Packet Video Workshop*, April 1999.

[11] E. Amir, S. McCanne, and H. Zhang, "An application level video gateway," in *Proc. ACM Multimedia*, November 1995.

[12] N. F. Maxemchuk, K. Padmanabhan, and S. Lo, "A cooperative packet recovery protocol for multicast video," in *Proc. International Conference on Network Protocols*, October 1997.

[13] J. Rexford, S. Sen, and A. Basso, "A smoothing proxy service for variable-bit-rate streaming video," in *Proc. Global Internet Symposium*, December 1999.

[14] J. Brassil, S. Garg, and H. Schulzrinne, "Program insertion in real-time IP multicasts," *ACM Computer Communication Review*, vol. 29, pp. 49–68, April 1999.

[15] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," *Request for Comments 1889*, January 1996.

[16] M. Handley and V. Jacobson, "SDP: Session description protocol," *Request for Comments 2327*, April 1998.

[17] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," February 1999. Internet Draft ietf-avt-profile-new-05.txt, work in progress.

[18] B. Krishnamurthy and C. E. Wills, "Study of piggyback cache validation for proxy caches in the World Wide Web," in *Proc. USENIX Symp. on Internet Technologies and Systems*, pp. 1–12, December 1997.

[19] G. Hjalmtysson and S. Bhattacharjee, "Control on Demand: An efficient approach to router programmability." To appear in *IEEE J. on Selected Areas in Communications*, 1999.

Stephane Gruber

Jennifer Rexford

FP-HA173000-SG/JR/AB

Andrea Basso