Program Insertion in Real-Time IP Multicasts

Jack Brassil
Bell Laboratories, Lucent Technologies

Sukesh Garg
Rutgers University

Henning Schulzrinne
Columbia University

We describe the design, implementation and operation of a prototype system which seamlessly mixes real-time audio and video streams originating from multiple, physically separated sources. Mixing is entirely decentralized, relying on new protocols to coordinate transfer of session control between IP multicast sources. The system is motivated by the desire to perform dynamic insertion of advertisements in active, real-time multimedia sessions. It permits content providers and viewers a far richer collection of relationships, opportunities, and features than possible with traditional broadcast media. Novel system features include 1) spontaneous meetings and business relationships between content providers, 2) program insertions from arbitrary locations on a multicast tree, 3) operation with no modifications to existing software players, 4) audience demographics from Real Time Control Protocol messages, and 5) mechanisms to allow users to select between multiple, simultaneous program transmissions. Additional system applications range from support of emergency broadcasts to the creation of novel *virtual broadcasting networks*.

1. Introduction

A steadily increasing amount of real-time and near real-time audio and video is being transmitted over networks using the Internet Protocol (IP). Expanding internet connectivity, network infrastructure improvements, and more powerful end systems are all supporting this growth. Many factors lead us to believe that this trend will continue. Access technologies already under deployment, such as cable modems, promise enough downstream bandwidth to support real-time multimedia transport to residences. The software required to receive and render low-rate video conferences is now bundled with new personal computers or freely available (e.g., NetMeeting). Personal computers are being equipped with inexpensive native peripherals, such as DVDs, cameras, and frame grabbers, supporting content creation, capture, editing and transmission. New video standards such as MPEG-4 are also emerging and are expected to improve IP video quality. Carrier backbone transmission and switching equipment is being upgraded to keep up with increasing traffic demands and quality of service requirements. To summarize, growing network and end system capabilities suggest continued growth in real-time multimedia transport over IP networks.

A recent NY Times article [1] indicates that broadcast radio stations (audio only) are emerging as leading users of internet transport:

> Today, 1,708 [radio] stations worldwide offer audio programs on the Internet, up from 763 a year ago. ... However, more than 4,300 other radio stations that have Web sites have thus far opted not to stream audio. Those that abstain, like WSOC, say there are good reasons -- namely, that streaming audio over the Internet can be expensive, audiences are typically tiny, and it is tough to prove the size of an Internet audience to advertisers."

The overwhelming majority of this audio content is unicast and streamed in near real-time (i.e., up to a few seconds delay). While packet loss recovery mechanisms in streaming transport protocols assure users of the highest quality reception, realizing this service quality comes at the expense of unicasting to emulate a broadcast medium. Reaching larger audiences, exchanging certain types of personal information, and satisfying advertisers seem better suited to distribution by IP multicast. Further, we believe that a number of exciting new services and opportunities are more easily achieved in a multicast setting.

But despite considerable advances in network and end system technology, use of real-time IP multicast has grown at a relatively slow rate. Four reasons are typically cited for this limited growth: 1) limited available transmission bandwidth (particularly for residential users), 2) the predominance of packet loss at congested network routers, 3) the existing infrastructure's lack of a simple, scalable multicast routing protocol, and 4) the lack of broadcast content for users to receive. These first two factors combine to yield what is perceived as poor reception quality — for video in particular — below what is acceptable to many potential users. To address the immediate problem of poor reception, most content providers have steered clear of multicast transport.

But, as discussed above, network infrastructure is being continuously upgraded, and routers and standards (e.g., RSVP, DiffServ, PIM) are being introduced to address existing network service quality deficiencies. Hence, the *technical* barriers to growth in real-time multicast are being addressed, and all indications are that this trend will continue. It is now possible to envision and plan for a global infrastructure capable of supporting high quality multicast transport of real-time programming on a large collection of audio and video 'channels' with local, regional, national and international distribution.

The fourth cited barrier to real-time IP multicasts growth is lack of content. It is certainly true that little video is transmitted on the global internet at this time. However, it is crucial to recognize that a huge amount of content is potentially available for an already huge audience; it is simply not being broadcast. Much of this content is of interest to smaller audiences than would be efficiently targeted by other technologies such as broadcast radio or television. Multicasting *all* 45 of the active FM radio stations in New York City has been estimated to consume only 2.5 Mb/s of internet bandwidth [2]. What limits further growth in IP multicasting, we claim, is less technology than the absence of a viable operational business model to justify an investment in broadcasting. With such a viable business model, desired infrastructure improvements will follow.

So why aren’t events broadcast? Consider the case of a sporting event. The interested audience normally far exceeds the actual number of attendees. Event organizers recognize the potential value of broadcasting their content, but currently are unable to capture that value. Recording content for deferred, on-demand playback has little promise, since a sporting event’s value diminishes rapidly as results become known. In some cases, organizers might consider a real-time broadcast on the multicast backbone (MBone). A significant barrier here is that multicasting forces the organizers to make an initial — and often substantial — upfront investment, yet no practical mechanisms exist to enable the content owners to be compensated for their expense. What event organizers seek is assured funding, which can be used to not only to finance broadcasting, but also ensure programming quality.

To realize such a viable business model we propose a system which supports audio/video program insertion in real-time IP multicasts. An important motivating application of this technology permits dynamic insertion of commercial advertisements in ‘live’ content. Yet many other applications of program insertion exist, including emergency broadcasts, video chat sessions, and virtual radio/television stations. We introduce a collection of protocols to seamlessly mix networked audio and video streams originating from multiple, physically separated sources. Mixing occurs in an entirely decentralized fashion, relying on coordination of nodes using the Real Time Transport Protocol (RTP) [3] and IP multicast. From a user perspective, inserted video programs are displayed within the same ‘window’ as the broadcast content to be viewed. If the inserted content is an advertisement, viewers have the familiar experience of commercial advertising on broadcast television. But beneath what users perceive as a familiar service, internet communications allows primary content providers, secondary content providers (e.g., advertisers) and viewers a far richer collection of relationships, opportunities, and features than possible with traditional broadcast mediums. Among the novel features our system provides are: 1) spontaneous meetings and relationships between primary and secondary content providers, 2) program insertions at arbitrary times from arbitrary locations in a multicast tree, 3) operation with no modifications to existing software viewers or players, 4) potentially detailed viewer demographics from Real Time Control Protocol (RTCP) [3] messages, and 5) mechanisms to support multiple, simultaneous program transmissions with user-selectable reception and viewing.

Let’s return briefly to the plight of the sporting event organizers. With the ability for third parties to inject advertisements in real-time broadcasts, organizers now have a valuable commodity to sell — the ‘dead’ time in their programs. As we will see, the effort required is minimal and often simply an extension of what organizers are already doing by seeking industry sponsors and patrons. In the system we propose advertisers themselves are responsible for their ad production and insertion, freeing event organizers from involvement in rebroadcasting tasks. Advertisers would certainly seem to have considerable willingness-to-pay to place ads in content viewed by an audience with specific, identifiable and somewhat verifiable demographics. The additional revenues generated by advertising represent the assured funding required to enable organizers to broadcast a high quality production. These same revenues can indirectly contribute to network infrastructure improvement, since all parties are now interested in ensuring high quality viewer reception.

In this paper we describe the design and operation of a prototype of a decentralized multimedia mixing system and its applications to program insertion. Section 2 introduces the overall system objectives and architecture. An implementation of a basic system is presented in Section 3. The next section discusses alternative implementations of a program injection protocol. Section 5 describes a number of interesting extensions and novel applications of the basic system. Conclusions are stated in the final section.

2. Architecture

Several design objectives guided the initial architecture of our program insertion system:

1. Compatibility

   The first and most crucial requirement was to ensure that no changes need occur to existing software *players*. We tested operation with the following players:

   - vic version 2.8 [4] and vat version v4.0b2 on Solaris 2.6,
   - iCast Viewer [5] version 2.0 on NT 4.0, and
   - Precept Software's IP/TV [6] demonstration version 2.0 on NT 4.0.

   While these applications are not widely deployed relative to the number of internet users, we believe that initially requiring player modifications would greatly impede the deployment of program insertion systems.

   We also sought to require no changes to *transmitters*. While this is a less crucial requirement since we expect far fewer broadcasters than receivers, we again seek to minimize disruption.

2. Remote third party insertion

   Secondary content providers should transmit their own programs. This makes most efficient use of the underlying multicast topology; it is unnecessary for inserted content to be first forwarded to event broadcasters for broadcast. Letting secondary content providers do the work frees the event broadcaster from unwanted administration and retransmission tasks. Because our transport medium is a bidirectional multicast network, there should be need to impose topological restrictions on program insertion points. Despite the fact that the source of packets varies during a session, viewers should see seamless transitions between primary and (inserted) secondary content.

3. Spontaneous meeting of content providers

   Because a program is directly inserted by a content owner (or its broadcasting agent) and negotiating an insertion takes negligible time, advance scheduling of programs should not be required. Primary and secondary content providers should need no pre-existing relationship nor agreements. Because spontaneous agreements can be realized, secondary content providers can tailor their insertions (e.g., ads) to current programming content.

4. Multiple, simultaneous secondary program transmissions

    Multiple, simultaneous program transmissions should be supported. Software viewers are intelligent, and this intelligence should be used to select from among several multiplexed video streams. Each receiver should be capable of identifying which available inserted program, if any, should be received.

This model represents a dramatic departure from program insertion in ether radio or television broadcasting. Our goal is to ensure that we can emulate those services, albeit with entirely different mechanisms, yet also offer numerous innovative services to take advantage of the new medium.

We introduce our program insertion architecture by focusing on an application familiar to readers: advertisement insertion. We refer to the primary content source as the *sender*, and secondary (inserted) content source(s) as *advertiser(s)*. Inserted content is an *advertisement*. The sender and the advertiser(s) each have agents called *proxies* who broadcast content on their behalf to *receivers* tuned to the destination multicast session. At this point, it is natural to identify the sender as the owner of the content for which a typical receiver tunes in. However in other program insertion applications we will explore later in this paper, such a distinction is arbitrary, since all content providers are indistinguishable.

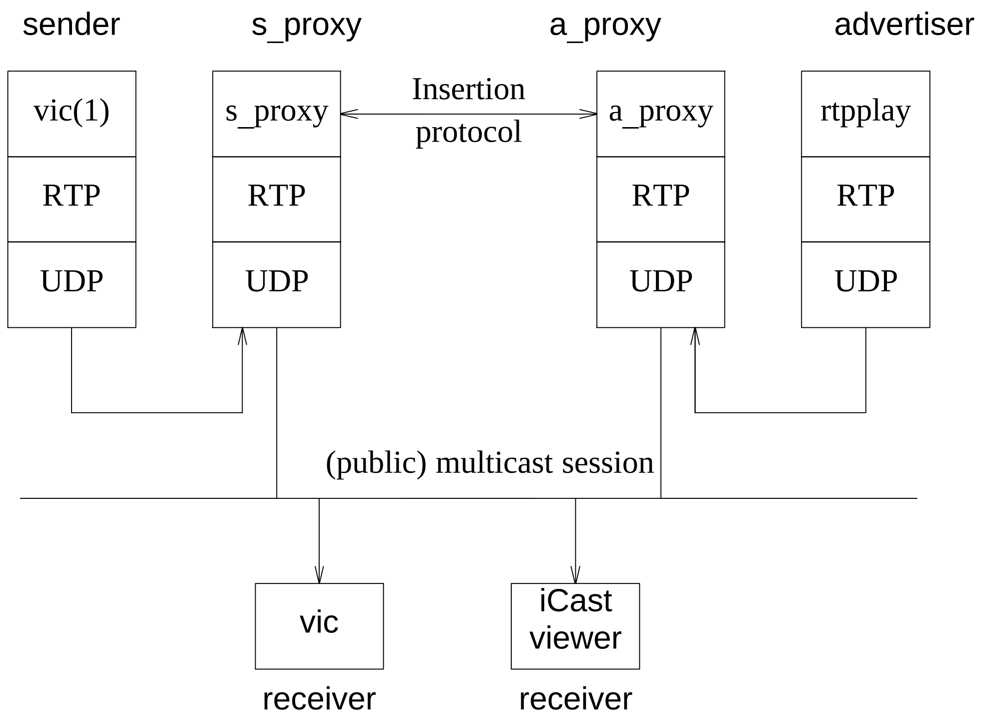


Figure 1 — Program insertion system architecture.

Given this nomenclature, Figure 1 depicts a block diagram of a system which realizes our desired objectives. The system components are:

1. sender - the source of the programming content. Sender can be any source of audio/video encapsulated in RTP. Sender ordinarily transmits packets continuously to its proxy on a private unicast or multicast address.

2. advertiser - the source of the advertisement content. Advertiser can be any source of audio/video encapsulated in RTP. Advertisements can be either stored video files or 'live'. Advertiser ordinarily transmits packets to its proxy on a private unicast or multicast address.

3. receiver - the receiver(s) of the mixed audio/video content. Receiver can be any destination capable of receiving and rendering audio/video in the encoding formats of both the sender and advertiser.

4. s_proxy - the sender's proxy is a server which transmits the sender's programming content on the destination (public) multicast session. S_proxy provides two services. First, when an advertisement is not in progress, it relays packets received from the sender to the destination session; RTP header fields are usually modified as part of the relay function. When an advertisement is in progress, s_proxy interrupts the packet relaying process, permitting the advertiser's proxy to transmit its own stream to the destination RTP session.

   Second, s_proxy is responsible for receiving and scheduling incoming requests for future advertisements. This task includes includes passing RTP header information to clients to permit them to inject advertisements seamlessly.

5. a_proxy - the advertiser's proxy is a client which transmits the advertiser's content on the destination (public) multicast session. A_proxy provides two services. First, when an advertisement is in progress, it relays packets received from the advertiser to the destination multicast session; RTP header fields are always modified as part of this relay function. When an advertisement is complete, s_proxy terminates the packet relaying process. The client is also responsible for passing RTP header information back to the s_proxy to permit a return to the primary programming content at the advertisement's conclusion.

   A_proxy is also responsible for issuing scheduling requests to place future advertisements. The client is responsible for properly handling denials of service (e.g., server responds that a desired time slot is unavailable) returned from the s_proxy.

Proxies for both the content provider and advertiser are introduced to avoid changes to viewers or players. Only one proxy is forwarding an RTP stream at any time; the proxies form a client-server pair that coordinates switching between the sources. The a_proxy (s_proxy) software can be run on the same system as the advertiser (sender) if desired. When context permits, we will use the term a_proxy (s_proxy) to refer to both the application software and the system running the software. The systems running the proxy software appear to the user to be the actual broadcast sources (i.e., not the sender nor the advertiser). As we will see they will share a single RTP synchronization source identifier.

The final element of the architecture is the insertion protocol. This protocol comprises a set of messages sent between the advertisers' and sender's proxies. The purpose of the protocol is to coordinate the transfer of a token; only the token holder is permitted to transmit packets on the destination multicast session. The protocol has 3 logical phases. During the *scheduling* phase a potential advertiser's a_proxy communicates with the s_proxy to arrange an insertion at a future time. Control is passed to the a_proxy at that scheduled time during the *transfer* phase, and control is returned to the s_proxy when the insertion is complete during the *return* phase.

The next section describes our initial implementation of the program insertion system. Understanding system operation requires an understanding of certain fields in the the Real Time Transport Protocol packet header, which we briefly review next; see [3] for a complete explanation of RTP.

RTP is an internet standard protocol for transporting continuous media. Our proposed program insertion system works with any applications capable of sending and receiving RTP packets. Program insertion works by manipulating RTP header information, and exploiting the underlying IP multicast network. Figure 2 shows the header format.

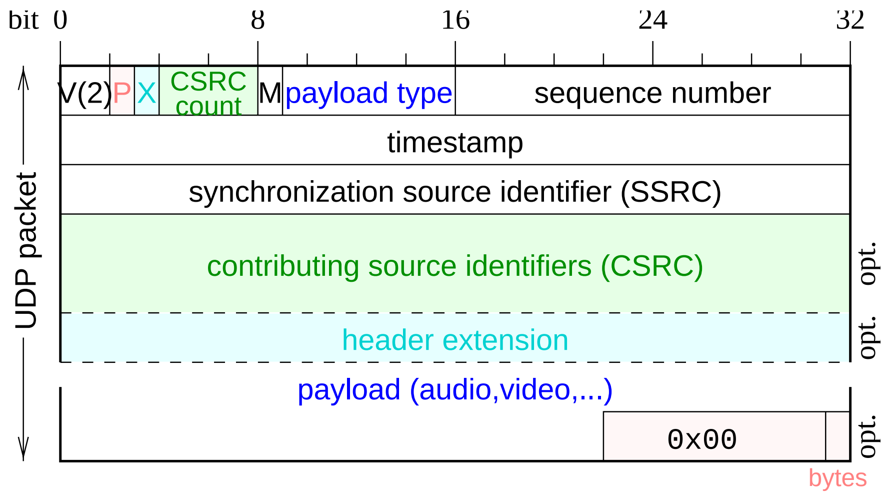


Figure 2 — RTP version 2 header.

The first twelve bytes of the header are required. The synchronization source (SSRC) is a random number which uniquely identifies the source of an RTP packet stream. Packets from a synchronization source are distinguished by a timestamp and sequence number. These fields are used by receivers for proper signal reconstruction and playout timing. The initial sequence number value is also random, and is incremented for each consecutively transmitted packet. Packets can and do arrive at their destination out-of-order.

The timestamp indicates the time of the sampling instant of the RTP payload relative to the initial timestamp value, which is random. The sampling rate for many audio/video encoding formats is constant, well known, and registered with IANA; other formats can have time-varying sampling rates. Media formats are specified by the Payload Type (PT) field. Multiple packets can have the same timestamp, as in the case where a large video frame is grabbed, encoded, but then transported in multiple packets.

A list of contributing source identifiers is present only if multiple RTP streams have been mixed. In this case, the CSRC count (CC) field indicates the number of contributors, and the CSRC list contains the original SSRC identifier of each contributing source.

RTCP is RTP's associated monitoring and control protocol whose primary functions are to provide feedback on reception quality, and distribute source identification and control information. This is realized by distributing messages including Sender Reports (SR), Receiver Reports (RR), and Source Descriptions (SDES). A session's RTCP and RTP messages are ordinarily transmitted to separate ports on a shared destination multicast address.

The authors of RTP did not and could not possibly have anticipated the development of an application requiring two or more end systems to share a single SSRC identifier as we propose. Later in this paper we will see that this sharing may not be necessary for future, intelligent viewers we envision. Indeed, the RTP specification *requires* that the identifiers are unique for each source in a session, and recommends actions for sources to take in the unlikely event of collision. Nonetheless, our architecture enables us to largely circumvent this restriction.

3. Implementation

Our initial experiments were designed primarily to test the feasibility of creating a program insertion system capable of realizing the stated design objectives. Our experimental testbed comprised sender, s_proxy, a_proxy, and advertiser software running on four separate SUN workstations using Solaris 2.6. However, as mentioned above, the sender and its proxy, or the advertiser and its proxy, or even both proxies could equally well be run on a single machine. We envision that ultimately the proxy functions could be integrated into the conferencing tools themselves. The overwhelming majority of experiments were performed at low bandwidths (e.g., 128 kb/s) with all workstations connected to a single 10 Mb/s LAN. Operation on wide area networks is discussed later in the paper.

The initial system we describe injects programs on demand. That is, an advertiser wishing to insert an ad contacts the content provider immediately prior to insertion. We will discuss scheduling of insertions at future times in the next section. Our modular system design handles audio and video streams separately. If both media are used in a particular application, then each stream requires a separate proxy. For simplicity we describe our implementation for a video-only session, and remind readers that operation for audio is completely analogous. System components were implemented as follows:

Sender

An analog video signal is captured from a live camera, encoded, packetized and transmitted as an RTP stream using vic. An associated RTCP stream is transmitted on a different port on the same destination address. The RTP and RTCP streams are transmitted to the s_proxy as either unicast or multicast. While in many instances we envision the signal privately unicast to a single s_proxy, we chose to support multicast in anticipation of multiple s_proxies. As in broadcast television, it is desirable to permit multiple distributors of the same content, with each distributor having its own relations with advertisers. A typical use would be broadcasting an event of international interest (e.g., World Cup Soccer) with different agents assigned distribution rights by country. Different distributors could then cater to their own sets of advertisers, and their respective audiences.

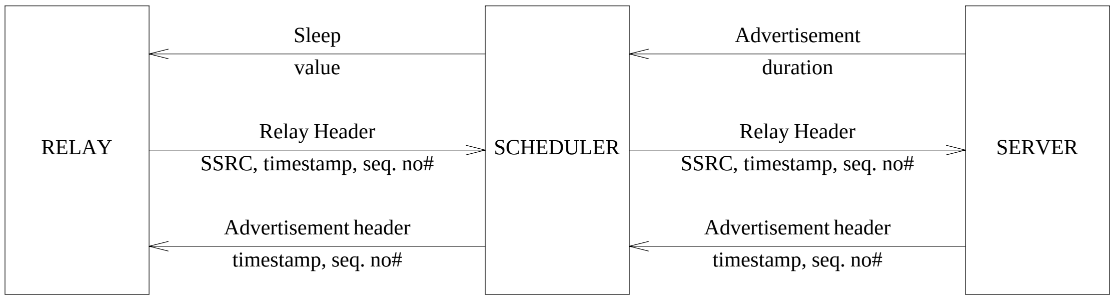
S_proxy

The s_proxy controls and manages the process of forwarding the sender's RTP session to the receivers on the destination multicast session. This function includes receiving, manipulating and forwarding the RTP and RTCP packets. It also receives and processes insertion requests issued by the advertisers, interrupting the relaying of the sender's stream for the duration of the advertisement. Finally, it takes control of the RTP session at advertisement termination time, and resumes relaying the sender's stream.

The s_proxy is partitioned into three separate tasks, implemented as the relay, scheduler and server processes. The relay process opens a socket for receiving the sender's RTP stream, and a socket for transmitting the modified stream to the receivers. The process stores the sender's SSRC, as well as the timestamp and sequence number of the last packet received. It also maintains an offset timestamp and offset sequence number. These values are initially zero but change according to the offset values returned by the advertiser. The offsets are added to the timestamp and sequence numbers field in the header of each incoming packet to be forwarded.

The server process listens for advertiser requests on an incoming TCP connection. Each accepted connection creates a child server process. Information contained in a request is passed to the scheduler block using a queue structure; the interprocess communication uses UNIX pipes. The advantage of using pipes is that the requests are queued and are served on a first-come, first-serve basis as needed to handle multiple incoming insertion requests. Each request from the advertiser contains the desired duration of the advertisement in seconds. Relaying the sender stream is stopped for the desired advertisement duration. At the end of the advertisement the server process receives current timestamp and sequence number values from the advertiser (on the still open TCP connection) and passes it to the relay process to calculate new offsets.

The scheduler process receives the incoming request from a (newly forked) server process and schedules to interrupt the relay process. Note that in our initial implementation all incoming ad insertion requests became interrupts immediately; they were not deferred in time (i.e., the scheduler is a null process). The scheduler process receives the value of the advertisement duration from the server process, and informs the relay process to *sleep* for the specified duration. At the end of the advertisement, new timestamp and sequence number values are received from the advertiser (through the scheduler process) and are passed to the relay process. Figure 3 illustrates this interprocess communication.



RELAY
SCHEDULER
SERVER
Sleep value
Advertisement duration
Relay Header SSRC, timestamp, seq. no#
Relay Header SSRC, timestamp, seq. no#
Advertisement header timestamp, seq. no#
Advertisement header timestamp, seq. no#

Figure 3 — Interprocess communication in s_proxy.

advertiser

We experimented with both "live" and stored video advertisement content. In the case of "live" content, we again used vic. Stored program content was recorded using rtpdump and played out using rtpplay from Henning Schulzrinne's RTP toolset rtptools-1.10 [7].

a_proxy

By design, a_proxy implementation is very similar to s_proxy; considerable amounts of code are shared between the two. However, the scaling issues that arise for a server (possibly needing to handle frequent incoming scheduling requests from many a_proxies) are not anticipated for a_proxy, permitting a simpler implementation

The a_proxy has a single process that does three jobs. First, it establishes a TCP connection to the s_proxy and requests an immediate time slot for insertion. Only the intended ad duration is specified. For an immediate insertion, the sender's SSRC and RTCP Session Description (SDES) parameters are returned, as well as the timestamp and sequence number of the most recently received sender packet. A_proxy then calculates offsets for both timestamp and sequence number. That is, for seamless insertion, it is necessary for the stream transmitted by a_proxy to "pick up where the s_proxy left off". This requires receiving the incoming advertiser's private stream, modifying RTP headers, and multicasting the modified stream on the public destination session.

Relaying the advertiser's RTP stream begins immediately; for each incoming packet the advertiser's SSRC is changed to that of the s_proxy, and appropriate timestamp and sequence number offsets are added. Finally at the end of the advertisement transmission, a_proxy sends back to the s_proxy its last transmitted packet's timestamp and sequence number. At this time, the s_proxy calculates new timestamp and sequence number offsets, then begins relaying the sender's signal once again "picking up where the advertiser left off".

3.1 RTCP Handling

Discussing RTCP was postponed above, partly because proper handling of RTCP packets in our system can be application-dependent. First, consider the relationship between a content provider and its proxy. In a simple case, a sender and an s_proxy might be jointly owned and operated by a single entity (e.g., a business). In this case, s_proxy could simply relay a sender's SDES and sender report (SR) messages, without modifying identifying information (e.g., CNAME, EMAIL, PHONE, TOOL fields in SDES messages). But in other applications a content provider and its proxy (or proxies) must be distinguished. This is the case, for example, if a content provider and one or more broadcasters are entirely separate entities. Here, each s_proxy should transmit its own session description and sender report messages, rather than relay these messages received from the sender.

Next consider the relationship between an s_proxy and one or more a_proxies. In ad insertion applications, editorial integrity (and possibly legal and regulatory guidelines) might require 'program' and 'advertising' content to be distinguishable by viewers. Yet, it is possible to conceive of other applications in which a single broadcaster might wish to hide — or at least not call attention to — content being transmitted from distinct sources.

3.2 Empirical Results

We tested our program insertion system in many configurations with the software players mentioned in Section 2 for both audio, video and mixed RTP streams. Our implementation permitted us to experiment with all of the RTCP alternatives discussed above. The proxies opened additional UDP ports as necessary to forward (and, if necessary, modify) sender's and advertisers' RTCP messages. In one case, we simply chose to drop (rather than relay) any RTCP messages received from the sources. In a second case, RTCP messages were modified and forwarded by both proxies, ensuring that identical RTCP source identifying information was transmitted from both. We also consider the case where all RTCP messages were relayed unmodified.

In video program insertion tests, the three players we used all exhibited different behavior. In an initial test, we performed a program insertion with RTP packets only — all RTCP packets were suppressed. Both vic and the iCast Viewer performed as we desired by playing the inserted program within the same open window displaying the sender's content. However the IP/TV Viewer bound to the s_proxy's IP source address and refused to accept the insertion from a_proxy. None of the viewers we tested accepted a program insertion if two distinct IP sources 'sharing' the same SSRC transmitted RTCP messages with different SDES identifying information (i.e., different CNAMEs); each viewer opened a second window to display the inserted program. In another test, a_proxy modified and relayed SDES messages from the advertiser, ensuring that these modified messages had exactly the same CNAME as any SDES messages issued by s_proxy. For this case, the iCast viewer performed the insertion with no difficulty, while the others two viewers again bound to s_proxy's IP address, the initial source of the SDES packets.

A key reason to test multiple viewers was to discover how each would behave when confronted with an 'SSRC collision.' The current RTP standard states that "If a receiver discovers that two sources are colliding, it may keep packets from one and discard the packets from the other when this can be detected by different source transport addresses or CNAMEs." Handling collisions in this fashion has some undesired side effects, such as facilitating denial-of-service attacks. If applications such as program insertion become popular, then clarification of proper handling of SSRC sharing can be written into future versions of the standard.

In summary, it appears that programs can be successfully inserted while maintaining the benefits of the various sources exchanging RTCP information. However, simply manipulating source identifying information might not be enough for program insertions to work with *all* existing players. Note that this does not necessarily require changing any existing player. Source IP address ‘spoofing’ can always be performed by a separate software mixer/translator application program located at any receiver currently unable to accept program insertions. Such tools are widely available in the public domain.

3.3 Proper Audio/Video Encoding

Recall that one of our objectives was to require that no pre-existing relationship need exist between content providers. This objective would not be realized if it were necessary for secondary content providers to adhere to a video encoding format dictated by the current programming content. This is fortunately not the case. The RTP specification [3] states that "A synchronization source may change its data format, e.g., audio encoding, over time." Hence, an RTP stream can be inserted in any standard format receivers can decode. It is each viewer’s responsibility to identify the payload type change and correctly adapt to the change.

A secondary content provider should ensure that, independent of any chosen video encoding scheme, a full frame is sent first (e.g., an I-frame in MPEG-2). Such a recommendation is common in any video mixing system [8]. This is easily realized in stored program advertisements. But achieving this becomes more difficult for "live" insertions, and also when returning control to "live" programming content. One possible solution is to have each proxy, upon receiving session control, discard video data until a full frame is received. The disadvantage of such a technique is obviously that switching between sources will take a longer, more variable period of time (e.g., perhaps 1/2 second before receiving the next I-frame in a typical MPEG sequence).

4. Design of Program Insertion Protocols

Recall that an *insertion protocol* comprises three logical phases: *scheduling*, control *transfer* and control *return* phase. The purpose of the protocol is to coordinate a token transfer; only the token holder can transmit packets to the destination session. In our initial implementation, the scheduling and control transfer phases were coincident. In general, however, arbitrary duration intervals could exist between each phase. Note that in some applications scheduling insertions might be preceded by regular broadcast *announcements* of the availability of open time slots in a program schedule, however we do not consider this part of the insertion protocol.

The scheduling phase is necessarily message based. An advertiser seeking to insert a program must explicitly contact the sender’s proxy with a desired program start time and duration. The s_proxy must explicitly accept or deny the request. The response can optionally contain additional information. For example, if an insertion request is denied because of scheduling conflict, the s_proxy could return alternate available time slots. Reliability demands suggest that this portion of the protocol should be TCP-based.

For the control transfer and control return phases we are experimenting with both implicit and explicit messaging. In the implicit messaging model, no messages are sent between s_proxy and a_proxy during the control transfer and control return phases. Instead, it is assumed that both parties are time synchronized by a separate, out-of-band mechanism such as the Network Time Protocol [9]. This mechanism should be able to realize modestly precise time synchronization (e.g., 20 ms). This synchronized time is used by both a_proxy and s_proxy to determine when to start and stop relaying the advertiser's and sender's content, respectively. Each party monitors RTP packets on the destination session, and infers from this an appropriate timestamp and sequence number offset to use at the assumption of control.

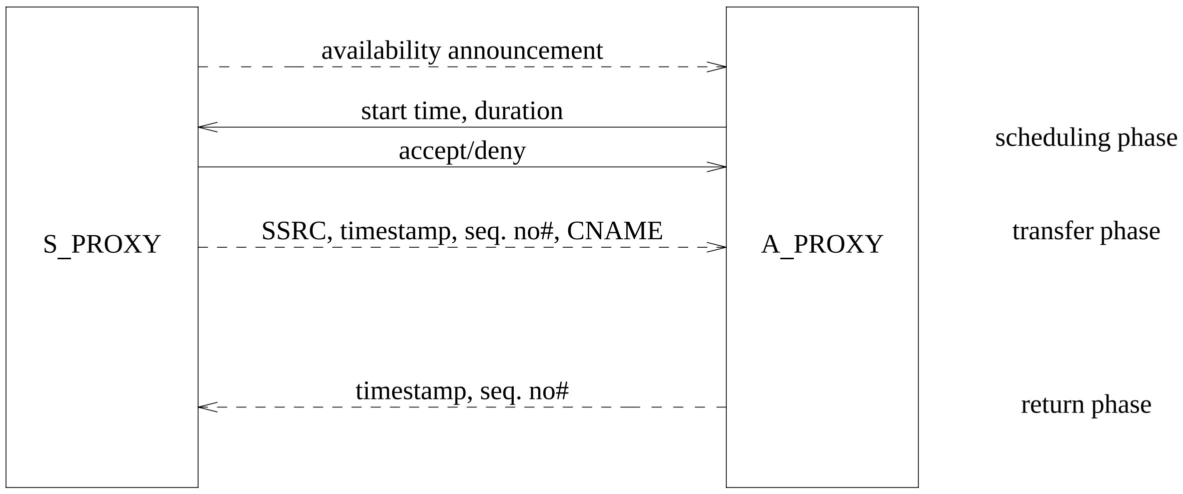


Figure 4 — Example of a program insertion protocol.

For an explicit message passing model, several alternative implementation approaches exist to transfer control, as follows.

1. Proprietary

    In our initial implementation, we created 2 messages with a proprietary format. These were sent over a single TCP connection. These messages simply contained the RTP header of the last packet transmitted to the destination multicast session by the active proxy. This approach was chosen because our initial implementation supported only immediate program insertions between two geographically closely located proxies.

2. RTSP

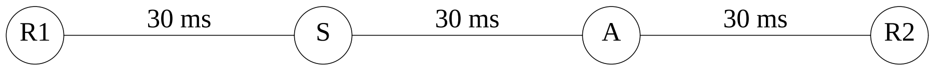
    A potentially attractive and robust protocol could be based on the Real Time Streaming Protocol (RTSP) [10]. Existing RTSP methods (e.g., SETUP, PLAY) could be used to trigger playout of stored advertisements located on video servers. Indeed, with appropriate extensions, it is possible for the program insertion protocol to be consumed entirely by RTSP.

3. RTCP Extensions

   If we assume that the destination multicast session will be monitored by the proxies, then new RTCP APPLICATION messages represent an explicit way to exchange necessary information to facilitate control transfer and return. Note, of course, that this approach is neither private (i.e., not unicast) nor reliable.

A promising and robust approach to insertion protocols might combine aspects of these approaches. Deferred, remote playout could be triggered by an RTSP PLAY message sent in advance (e.g., several seconds or minutes) of a scheduled control transfer. Such a message would contain appropriate session control information (e.g. timestamp, sequence number and start time). In the event a deferred trigger message failed to arrive, an advertiser could simply monitor the session and estimate these parameters independently and begin when scheduled. In the worst case, an advertiser would begin only after detecting that active transmissions have ceased.

When considering these alternatives in protocol design, it is important to recall that our goal is to achieve "seamless" switching between sources. Switching between video sources transmitting less than 30 frames/sec fortunately affords some leeway in both the timing of transmitted packets and the calculation of timestamp offsets. However, achieving seamless switching in a wide area setting is potentially difficult. Consider the linear topology below where two receivers, *R1* and *R2*, are connected to the advertiser proxy *A* and sender proxy *S* over links with specified mean packet latency.



Suppose we assume that *A* and *S* are perfectly time synchronized, *S* is to transfer control to *A*, and *A* has the benefit of advance knowledge of the timestamp, sequence number and admission time of the last packet that *S* will transmit.

When should *A* transmit its first packet? Suppose that *A* does so at exactly the time *S* would have transmitted its next packet. Because of the additional delay in reaching *R1*, the packet nominally arrives there 'late.' On the other hand, *A's* first packet might arrive to *R2* before *S's* final packet. Depending on the receivers' states, these ill-timed arrivals might not result in a perceived disruption when played out at either receiver. But suppose that the sources are imprecisely synchronized, and *A* uses both an *estimated* timestamp and sequence number. Poorly chosen sequence numbers could lead to duplicates, causing receivers to discard one or more packets. Poorly estimated timestamps could easily produce unintended, perceptible 'freeze frame' or 'fast forward' playout effects. Considerable empirical work still needs to be done to establish the extent to which source transitions can be made transparent to receivers in a wide area network setting. In the meantime, the next section introduces some simple ways in which undesired transition effects can be concealed.

5. Future Extensions and Applications

This section explores numerous opportunities to extend our work on program insertion systems, and introduces some new applications.

5.1 Synchronization Source (SSRC) Identifier Masks

A desirable feature in ad insertion applications is the ability to simultaneously transmit different ads to different viewers. At a minimum we seek to emulate broadcast television's ability to inject ads locally, as at a cable headend. But we aspire far more flexibility, with individual users being able to customize their reception by choosing among different simultaneously broadcast advertisements, or perhaps choosing no advertisements at all.

"Local" advertisements could be implemented by varying the scope (i.e., time-to-live) of multiple, simultaneous advertisements launched from different ad inserters. But this crude mechanism is both flawed and insufficient to support the features we seek. We consequently propose the more versatile mechanism of allowing viewers to specify an SSRC *mask*. The mask is a 32 bit number AND'ed with each incoming packet's 32 bit SSRC value. If the result equals the SSRC value of the desired session, then the packet is accepted as belonging to that stream.

This approach effectively creates a set of SSRCs which can be used by advertisers to create separate, simulcast advertisement channels. For example, suppose receiver A has mask 0xffffff0f and receiver B has mask 0xfffffff0. Both receivers are tuned to a multicast session with SSRC identifier 0x55555500. Then, if during a 'commercial' break one advertiser begins transmitting with an SSRC identifier 0x55555501, this content will be accepted and played out by receiver B as a logical continuation of a programming sequence. If a second advertiser begins transmitting with an SSRC identifier 0x55555510, this content will be accepted and played out by receiver A. In the event of conflicts (i.e., multiple simultaneous streams with valid SSRCs), a conflict resolution procedure is invoked (e.g., packets with the lowest valid SSRC are played out, packets with other SSRCs are discarded).

Since filtering and merging content is performed at the receiver, SSRC masks have the disadvantage that all advertising channels are distributed to every viewer in the multicast group, regardless of what content is being viewed. This severely limits the maximum number of simultaneous channels one could support per group. To conserve bandwidth when supporting larger numbers of simultaneous channels, use of multiple multicast channels might be necessary.

Note that any masking mechanism raises a number of questions which remain to be solved. Masking would immediately require coordination of SSRCs among content providers. It is unclear how each receiver learns of and specifies its desired SSRC mask, though it is possible to imagine distinct *session directory* listings for a session and its related "advertisement sub-channels." Should masks be assigned on a session-by-session basis, or more universally? How should a receiver behave if no content is available on an "advertisement channel" to which it is tuned? The answers to these questions could well be application dependent.

5.2 Source Transition Concealment

The previous section mentioned the possibility of perceptible artifacts during source switching times, particularly in wide-area network settings. *Concealment* techniques can and should be used by both advertiser and sender proxies to hide aesthetically displeasing transitions. Perhaps the simplest video concealment technique is for both sources to agree on content surrounding the transition; the proxy completing its transmission can fade-to-black, while the proxy initiating its transmission can begin with multiple black frames. Any number of special effects can be used to aesthetically improve the receiver's transition. In a similar fashion, proper volume control can be used to minimize audio artifacts (e.g., clicks).

5.3 Electronic Commerce

Once a program insertion system exists, it is necessary to arrange for payment for insertions. Since programs can be scheduled spontaneously and inserted nearly immediately, we seek a payment system with similar characteristics. Throughout the last decade there have been many research proposals for ecommerce systems that appear to have many of the properties we desire (e.g., NetBill, Secure Electronic Transactions), though these systems are yet to be widely deployed. But since only broadcasters and advertisers — not viewers — engage in business transactions, it is possible that the limited deployment of existing payment systems will not be a substantial barrier. In the short term, however, it appears to be necessary to consider out-of-band payment mechanisms in which advertisers are prevalidated or even prepay (though possibly only a few moments in advance) for future insertions.

5.4 Back-to-back Ads

While our initial implementation does not yet perform scheduling of advertisements at arbitrary future times, a limited form of scheduling does work. An ad request arriving when an advertisement is in progress results in the two ads played out back-to-back. To minimize the number of program interruptions, back-to-back ads are very common in broadcast media, and must be supported in our system.

5.5 Time Stamp and Sequence Number Estimation

The perceived time required to switch between two sources can be minimized by properly estimating an observed RTP stream's time stamp and sequence number at a future time. We have performed a number of simple experiments which suggest that obtaining sufficiently accurate estimates of these parameters is not difficult. In general, conservative overestimates of future sequence numbers are preferred because receiving an RTP stream with missing sequence numbers is less harmful than receiving duplicates. Estimating a future timestamp for an RTP stream with a *known* constant timestamp rate is trivial. For unknown constant timestamp rates, acceptable estimates can be readily obtained after observing the session for just a few seconds.

5.6 Security and Robustness

Our system suffers from all of the well known security and robustness problems of the underlying transport mechanism. Because programs can be inserted almost instantly, a content provider places considerable trust in an advertiser. Methods to authenticate advertisers need to be explored, though it appears that this can be achieved using either standard RTSP or transport-layer security mechanisms.

There are currently no formal means to prevent malicious parties from interfering with a broadcast. There are also numerous ways to launch denial-of-service attacks on individual viewers, or even groups of viewers. In an IP multicast environment, it is fundamentally impossible to restrict the ability of any host to send to the multicast group. IGMPv3 can restrict sources based on their IP address, but this protocol is not widely deployed. Restricting sources may also not be desirable, since it would suppress both RTCP messages as well as legitimate advertisements. Given that the IP address of a source's RTP packets cannot easily be forged without interfering with multicast transmission, it may be possible to have the station indicate other legitimate sources of ads in special RTCP extensions. Signing a packet using a public-key algorithm is computationally difficult and would introduce at least 64 bytes of overhead per packet. Thus, it appears that one has to resort to non-technical means to discourage intrusions by malicious third parties.

An additional weakness of this system is the ability for receivers to use software filters to block advertisements at their end systems. While hardware-based ad *blankers* exist for broadcast receivers, their use is not widespread; viewing or hearing nothing in lieu of an undesired advertisement offers viewers little advantage. But software filters are easy to construct, so their use is a threat. If filter use were to become common, broadcasters could consider countermeasures based on frequent SSRC changes. Ultimately, however, a blanker can filter based on source IP addresses. A broadcaster's attempt to spoof source IP addresses would likely interfere with the correct operation of wide area network multicast routing.

5.7 Enhanced Information Services

Carrying real-time services over the Internet allows improved directory services, easy addition of side information such as content labeling, bandwidth diversity for different kinds of programming, as well as opportunities to receive more diverse programming particularly in more sparsely populated areas. Labeling also makes it easy to have receivers assemble custom programs for listening at some later time (i.e., time-shifting).

One well known problem with internet-based radio/TV ads is the inability for a user to do "click through" to obtain more information. For multicast distribution, the creation of a URL field in RTCP messages has been suggested to tackle this problem [11]. One challenge here is to ensure proper presentation to avoid confusion between URLs associated with the program and the inserted content.

Our proposed system relies entirely on RTCP messages (e.g., Receiver Reports) to provide audience demographics to advertisers. RTCP was obviously not designed to provide the detailed, verifiable viewer information that advertisers might demand. Though it is possible to enrich existing RTCP messages, or create new ones, doing so seems to stretch the protocol well beyond its intended capabilities.

As those familiar with RTCP monitoring know, it is challenging enough to obtain reasonably accurate estimates of group membership size. RTCP may not be able to provide audience size estimates for large groups. For example, if transmission bandwidth is 128 kb/s, there can be at most 36,000 RTCP SDES packets per hour, since the RTCP bandwidth is limited to 5% of the session bandwidth. For larger audiences, size estimation via sampling techniques may be most appropriate. This also avoids the problem of having each receiver send multicast data, which can cause scaling difficulties for some multicast routing protocols.

Privacy issues abound when discussing demographics. Conflicts are sure to arise over differences between the information viewers wish to share (both with advertisers and their fellow viewers) versus what advertisers might like to know. How to balance the needs of these parties remains an open problem. It is worth noting, however, that there is considerable willingness-to-pay to place ads in print media, where no real-time feedback exists.

5.8 Virtual Broadcasting

Program insertion opens the door to many new applications. One difference between traditional broadcast media and internet distribution is that the content provider and "transport" provider are much more closely associated in the former case. Unlike broadcast media, internet distribution can achieve complete independence between a communication channel and its "owner" or dominant content provider. With program insertion, one can create *virtual broadcasters* which produce little if any programming. Such a broadcaster simply exercises editorial judgement by combining programming content from other sources (as insertions) with advertisements (also as insertions). This notion easily extends to *virtual broadcasting networks* as an affiliation of virtual broadcasters who mix ‘national content’ with ‘local advertising’. Such an organization provides great flexibility in the creation of broadcast entities; one can image a virtual college radio station as a National Public Radio affiliate. Indeed, any computer user can potentially become a broadcaster, with unprecedented access to quality programming content.

6. Conclusion

We have designed and implemented a decentralized mixing system which allows remote third parties to insert programs in continuous media IP multicasts. While we have concentrated on ad insertion applications, the system can also be used for more general program insertion applications including emergency broadcasts, video chats, and virtual broadcasting.

Our work on program insertion systems optimistically anticipates that global IP network infrastructure improvements will continue at a rate sufficient to support acceptable quality real-time transport. Such quality is not available today on an end-to-end basis. However, considerable opportunities and applications exist if IP multicast is used on a more limited basis, such as with satellite distribution of audio/video feeds to "headends" for local internet or intranet distribution.

We have also proposed a business model for funding programming events. Alternative funding models (e.g., pay-per-view) will likely coexist with commercial advertising as they do on broadcast television. It is resoundingly clear from early feedback we have received that many potential viewers find commercial advertising unappealing. For these viewers we have proposed mechanisms to ensure that ads can be user-selectable, or not viewed at all.

Dynamic program insertions by third parties also raises serious editorial control issues. In some contexts, complete control of all transmitted content is required, and it simply will be unacceptable to permit untrusted parties to insert content not subject to prior review. Clearly this would be the case if some part of internet real-time transmissions were subject to regulatory oversight as in traditional broadcast environments. We also recognize that traditional broadcasting relies heavily on contractual business relationships. So though our proposed system can support spontaneous insertions between parties without prior business relationships, it is likely that some broadcasters will choose not to exercise this feature.

Though it was possible to implement a program insertion system, we did so only by circumventing some restrictions imposed by RTP. The opportunity exists to have future versions of the RTP specification contain modifications to support SSRC sharing or masking. Having software players disassociate an SSRC and an IP source address will likely also help support mobility applications.

The rapid acceptance of software viewers such as *RealPlayer* and *Windows Media Player* and the emergence of commercial internet broadcasting agents such as *Broadcast.com* suggest continued strong demand for dissemination of multimedia content. Advertising is rapidly becoming a viable — if not preferred — funding model in that context. The commercial market is obviously not waiting for transport service guarantees, higher speed residential access technologies, new electronic payment systems, nor improved IP multicast infrastructure. Looking ahead, it appears as though the existing multimedia distribution approaches will serve only to accelerate demand for increased bandwidth and service quality. It remains to be seen just how much of this growing demand for multimedia 'broadcasts' will be satisfied through the existing distribution technology.

References

1. M. Richtel, "Radio Stations Are Cautious About Audio Online," NY Times on the Web, http://www.nytimes.com/library/tech/98/10/cyber/articles/16radio.html (October 16, 1998).
2. H. Schulzrinne, "Re-Engineering the Telephone System," *Proceedings of SICON'97*, Singapore (April 1997).
3. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," *Internet RFC 1889* (January 1996).
4. S. McCanne and V. Jacobson, "vic: A Flexible Framework Framework for Packet Video,' *Proceedings of ACM Multimedia '95* (November 1995).
5. iCast Corp., *iCast Viewer 3.0 Datasheet*, http://www.icast.com/ (1998).

6. Precept Software (Cisco Systems), *IP/TV Viewer*, http://www.cisco.com/warp/public/732/net_enabled/iptv/ (1998).

7. H. Schulzrinne, *RTP Toolset Version 1.10*, http://www.cs.columbia.edu/˜hgs/software.

8. Cable Television Laboratories, Inc., "Digital Program Insertion: Request for Information," (April 1997).

9. D. Mills, "Network Time Protocol (Version 3) Specification, Implementation & Analysis," *Internet RFC 1305* (March 1992).

10. H. Schulzrinne, A. Rao, and R. Lanphier, "RTSP: A Transport Protocol for Real-Time Applications," *Internet Draft* (February 1998).

11. P. Parnes, "Proposal for additional RTP-SDES types," http://www.cs.columbia.edu/˜hgs/rtp/drafts/draft-parnes-rtp-sdes-00.txt, *Internet Draft* (November 1998).