Applying Real-time Multimedia Conferencing Techniques to the Web

Mark Handley
University College London

1 Introduction

This paper is about unifying two worlds - those of the real-time multicast multimedia conferencing community, and of the WWW community. The conferencing community has been developing solutions for live real-time media flows on the internet. The web community has been developing solutions for stored media. Both communities have much in common, and this paper attempts to identify ways in which the solution from both communities can be unified.

These two communities should have much in common - they share a common network infrastructure and a common interest in networked multimedia. However, it appears that the web community generally takes a more top-down approach and is primarily concerned with providing real solutions to real users *now*. The multicast conferencing community generally takes a more bottom up approach, with good understanding of network behaviour, but a more relaxed ivory-tower approach towards products. Both are of course necessary approaches.

This paper describes techniques and protocols developed for the multimedia conferencing community. We attempt to identify solutions that can be deployed immediately and which have the right properties for appropriate exploitation of ubiquitous multicast and real-time support in the internet infrastructure when they are eventually available. Some of this is taken for granted in the Mbone community, but the assumptions are rarely written down in a form accessible to the web community.

Firstly we shall spend a little time examining non-solutions that seem to be raised often enough that the deserve the issues being explored. Then we shall look at which building blocks exist that can be applied usefully in the context of real-time media flows and the web.

2 Dead Ends

There are many protocols and ideas that lead nowhere, but some seem to get raised often enough to be worth spending some time examining in order to explain why we discard them.

2.1 Multicast or Multimedia URLs

A URL is a Uniform Resource Locator. We all know that, but it bears saying it occasionally. It provides a location to obtain something, and a mechanism to use to ask for it. Every so often, someone suggests defining a URL syntax for real-time media streams. This is a *bad* idea.

A "real-time media stream" can mean almost anything to almost anyone, so to see why this is not a useful idea, let's look at two examples from different ends of the spectrum.

URLs for live multicast sessions

On the Mbone, the Internet's Multicast Backbone, live multimedia sessions are multicast out worldwide almost continuously. These range from small meetings to live audio and video from the space shuttle. To view or participate in such sessions requires decoders for each of the media flows, such as audio, video, still image multicasting, whiteboard, text editors and so on. In addition, a receiver must discover the multicast addresses for each media stream, UDP ports, protocols used, media formats, decryption keys and often format specific attributes too. For live sessions, this is typically done using a session directory tool such as sdr[2] which advertises such sessions using the Session Announcement Protocol (SAP)[3] and the Session Description Protocol (SDP)[5].

Many people would like to put links to these sessions in web pages to create a seamless information space, and this makes much sense. However, the suggestion is also made that this is best done using a URL that provides all the information that is necessary to receive the session, and this is not such a good idea.

A URL is a single link. A multimedia session consists of many media streams "flying in formation", but not bundled together. Some media are often considered optional - if insufficient bandwidth is available, the video stream is often dropped and receivers are more happy with good audio and whiteboard quality that seriously degraded audio, video and whiteboard quality. It is typical to provide separate decoders for each stream, as "mix and match" of protocols and media formats is very much the order of the day. So, how should a single URL convey all this information about multiple streams chosen from many formats, each of which requires its own configuration information and arbitrary parameters? What is required is an extensible protocol and not a URL. Sure, you could define a mapping of such a protocol into a URL, but it would be extremely inelegant, very long, and still give the incorrect impression that a multicast session was an all or nothing affair.

URLs for Stored Multimedia Sessions

The scenario here is that we have a recorded multimedia session that we wish to stream out audio, video and whiteboard "slides" in real time from a server, unicast to an individual receiver. We wish to embed the video stream and whiteboard display in a page of HTML and be able to play, pause, rewind and fast-forward the multimedia session.

All the arguments against a single URL for live multimedia sessions apply equally well to recorded media. In addition, there is now the problem that we require the session to be embedded in HTML, to be controlled, and we require two synchronised embedded windows (for the video and slides). The descriptions embedded in the page need to include a mechanism to contact the server to initiate playback and the other controls, and they need to cope with three separate media streams triggered from the same controls.

Clearly doing this with a single URL is infeasible, but what about having a "background audio" URL, a video URL associated with the video frame, a whiteboard URL associated with the whiteboard frame, and an HTTP URL for each of the control buttons? In this scenario, the audio, video and whiteboard embedded receivers are automatically started when we load the page of HTML they are embedded in, but because they are UDP based, they just passively listen until one of the control buttons is pressed and data starts to arrive. Perhaps the idea of *monomedia* URLs for real-time media streams has some merit if we assume they are automatically followed in the same manner as `<image>` URLs?

An example might be:

```
<video src="rtp://mmm.cs.ucl.ac.uk:5432/fmt=H.261/size=CIF">
```

or possibly

```
<video src="rtp/video://mmm.cs.ucl.ac.uk:5432/fmt=H.261/size=CIF">
```

There are several issues here.

- The first is the use of HTTP as a control protocol, which we shall address below.
- The second issue is that we need to link these URLs together in some manner to allow us to convey to the media decoders that they need to communicate with each other to provide synchronisation between the media streams. If we have separate audio and video streams, we only need lip- synchronisation. If we have additional streams we may need multiway stream synchronisation. Performing such synchronisation with RTP[1] based streams is not difficult[6], but we do need to mark the URLs to do so. We cannot just assume that all URLs on the same page should be synchronised as we may have many such embedded multimedia sessions.

  This can be achieved by adding a synchronisation tag to the media frame tags to indicate which URLs within a page should be synchronised.
- If we embed several sessions in this way, we have to be able to cope with the audio decoders for multiple background audio URLs being active (or at least in ”active wait”) state simultaneously.

Most importantly though, these URLs are not URLs in the traditional sense. A URL gives an access method and a location. These media stream URLs must give an access method (eg RTP) a location (where to listen to: remote address and *local* port unlike the normal HTTP remote address and *remote* port) but additionally they must give format information and arbitrary configuration parameters. This is a new departure - an analogy is if URLs for HTTP contained the MIME content type for the data the link points to. Imagine if you couldn’t ever change a GIF image on your web-site to a JPEG image of the same format without tracking down everyone who had linked to it and getting them to change their URL. This dependency between the data and the link to it should be avoided - the indirection provided by using MIME in the HTTP reply is part of what makes the web so flexible, and should be maintained.

So RTP URLs are a bad idea. Couldn’t we solve the problem by adding a setup phase per media stream to provide this level of indirection, and provide a URL for this?

So we might have:

```
<video src="http://www.cs.ucl.ac.uk/mmc/sess27/video.sdp" sync=3>
```

and this URL is accessed immediately the page is loaded, and the server returns a media description giving the format and protocol. We’ve no longer got a multimedia URL at all - the media is described in the HTTP reply, which is perhaps as it should be.

2.2 TCP is good for everything

HTTP is based on TCP. TCP has two main features which make it at least reasonably appropriate for HTTP:

- It provides a reliable connection - no parts of your data get lost.
- It performs congestion control so that available bandwidth is shared out reasonably fairly amongst connections sharing a congested link.

The combination of these two features make TCP suitable for transferring static objects, such as images or HTML pages. The more congested the links, the less bandwidth everyone gets, and the longer you have to wait for an object, although it still arrives eventually.[1] Thus, what TCP is doing is trading loss for delay, and at least within TCP's operating region, it does a near optimal job of this.

For real-time multimedia, the requirements are different. A large web image doesn't have any predefined timing relationship between its data packets, so TCP can adjust the time of transmission as it sees fit. A real-time flow such as audio does have a predefined timing relationship between its data packets - you'd rather like your audio to play out at the same rate it was recorded! As a result, protocols based on TCP which add arbitrary delay are not useful. In the internet, this usually means basing your protocol on UDP, and in particular, for real-time flows, RTP (running over UDP) is likely to be the correct solution.

This is not to say that we can just run lots of UDP based real-time media streams with no congestion control and expect everything to work. In the face of congestion, an open-loop UDP stream will not back off its transmission rate. Well behaved TCP connections sharing the same link will. If there is enough UDP traffic, the TCP connections won't get a look in. If there is significantly more UDP traffic that the link can support, everyone will get bad service, and the network will be in a state of congestion collapse. UDP based real-time streams still need to perform congestion control, but this needs to be done with the cooperation of the sending application, which must increase the compression of the data stream to reduce the data rate in the face of congestion feedback.

However, what about control messages? For example, if we wish to provide buttons to perform fast-forward, pause, rewind, etc, to control the playout of multimedia streams from a server, what is an appropriate control protocol?

Several solutions to this problem make an HTTP connection to the server each time a button is pressed. Other solutions suggest using a single TCP control channel (or persistent HTTP) to convey the commands over. While TCP does provide the reliability required, it does so at the expense of arbitrary delay. So you have no idea how long it will take from you pressing the button to the command actually arriving at the server - if you're unlucky, even relatively low loss levels can result in significant delays.

Whatever protocol you use, on the Internet you cannot control whether your packets get lost or not[2]. What you can do though is keep control of the timing relationship within your application so it can retransmit after a timeout that is appropriate to the application, rather than depending on TCPs congestion control mechanism to decide the timing for you. This does mean you also need to implement reliability mechanisms in your application, but this is not a difficult problem for this sort of application. There is of course an implicit assumption here that the rate of control messages is low (at least when compared to what they're controlling), so that control messages themselves are not a significant cause of congestion.

2.3 Bundled Media Streams

The RTP model for media streams is that each media (audio, video, whiteboard, etc) is a separate stream, and is sent in separate packets to a separate destination port. This model has several advantages:

- it allows each stream to receive a different quality of service, and even different routing if necessary.

[1] Links can become so congested that TCP connections time out, but these links are broken, and ISPs providing such links deserve to go out of business - no protocol is going to be able to cooperatively provide a reasonable service over such links.

[2] The internet is evolving to support reservations providing a degree of assurance, but such reservations (if and when they are widely available) are not really appropriate for this sort of very sparse control data

- it allows a “mix and match” approach to choosing appropriate encodings for each media, both in the protocols themselves and in the choice or decoders or helper applications.

However, this is in contrast to more traditional approaches such as MPEG transport stream, or the ITU’s H.320 recommendation, which combine multiple encodings into a single transport stream. These approaches do so because they have to - they have a single channel that they control in its entirety, and they need to put multiple media down that channel. The internet isn’t like that - IP lets you arbitrarily multiplex streams together without combining them in one transport protocol.

The argument often made is that lip-synchronisation is simpler if you bundle multiple streams together into a single transport stream, but this argument does not hold true. Whether you bundle streams together or not, you still need playout buffers after decoding at the receiver to perform synchronisation. With bundled streams, you’re compensating for variance in decode times, plus network jitter affecting the bundled stream. With unbundled streams, you’re compensating for variance in decode times plus separate network jitter for each stream[6]. Bundling doesn’t significantly change the way you have to use these playout buffers and doesn’t make life any easier.

The only advantage bundling has is when audio is very compressed: sending in separate streams means that to keep delays down, you must put sufficiently few bytes in each packet that the packet header overhead becomes high. However, delay is much less critical for recorded media streams than it is for interactive conferencing, so this effect is less pronounced, and work is currently ongoing in the IETF to produce a combined IP/UDP/RTP compression which will make this effect negligible, even for interactive sessions.

In general then, bundling of multiple streams into a single transport encoding is a bad idea as it deprives you of much flexibility and adds pointless complexity.

3 Building Blocks

Many concepts and protocols developed for the multimedia conferencing community seem to be equally applicable in for real-time multimedia in the web.

3.1 RTP

RTP is the Real-time Transport Protocol, which is a standard packet format for real time data. Typically it is used over UDP, and it provides identification of payload formats, information for network jitter removal using adaptive playout buffers and cross media synchronisation, identification of senders and receivers, and a degree of feedback about reception conditions. Accompanying standards provide agreed methods to packetise a wide range of media formats including PCM, ADPCM, GSM and LPC audio and H.261 and MPEG video.

RTP works equally well for unicast data streams and for multiway multicast sessions. RTP is currently in use in many multimedia conferencing applications on the internet, and a scheme for combined IP, UDP and RTP header compression[7] is currently being defined[3] so that RTP will also work extremely well over very slow modem links.

[3]This is similar in principle to the well established TCP header compression[8], and compresses all these headers down to about 2 bytes

3.2 Redundancy

If a media stream is crossing the current internet, with no form of reservation or traffic prioritisation, it is likely to cross at least one link which is in some state of congestion. Whilst media streams should reduce their data rate in the face of congestion, it is also necessary to ensure that the data stream as it is received is useful, i.e., that the playback quality is acceptable.

For multimedia conferencing, the techniques available to cope with packet loss are constrained primarily by the delay constraints imposed by the need for interactivity. For media flows from a server a wider range of solutions are available, and the interactivity constraints are more relaxed. A second or two lag in responding to control inputs during playback due to the data packetisation scheme may be acceptable. This allows much larger playout buffers at the receiver, and more time to obtain a replacement for missing data. If interactivity is relaxed in this way, possible solutions break down into several categories:

- **Limited retransmission of lost data**. There is a short window during which lost packets can be retransmitted from the sender before the receiver's playout point is reached. These retransmissions are based on negative acknowledgments from the receiver. . However, retransmission without immediate congestion control is dangerous as it increases the data rate directly as a result of congestion, and so sets up a positive feedback loop which is a very bad idea.

- **Packet-level Forward Error Correction**. Extra data packets containing redundancy can be added to the data stream, and when a packet is lost, the missing packet(s) can be reconstructed from the packets that did arrive and the redundant packets. Packet level forward error correction is very effective if the loss rates are sufficiently predictable, but less so if loss tends to come in bursts, where it may be easy to exceed the level of protection provided by the FEC scheme and so recover nothing. FEC is very much an all or nothing thing.

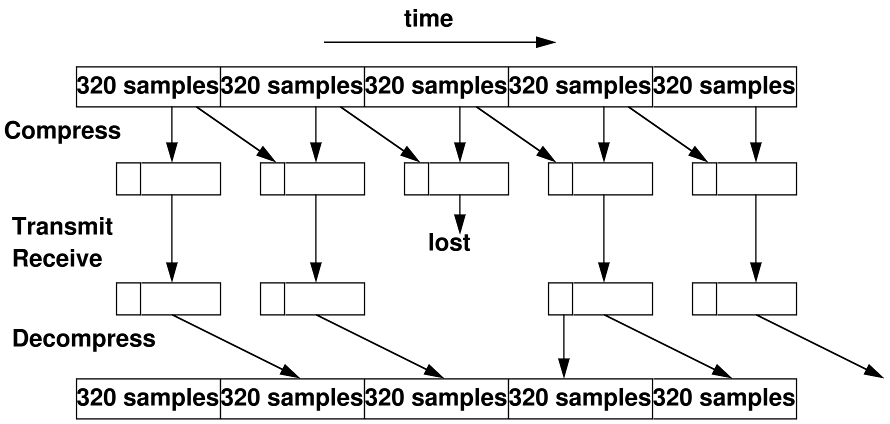

Figure 1: Adding a Low Quality Redundant Data Stream to Protect Against Packet Loss

- **Lower quality redundancy**[9][10]. A low quality (higher compression ratio) version of the main encoding is added piggy-backed onto a later data packet as shown in figure 1. If the packet containing the primary encoding is lost, the secondary can be decompressed and used as an approximate substitute. This differs from FEC in that the redundant encoding is only trying to approximate the missing *signal*, not reconstruct the actual missing *data*. Such a scheme is well suited for audio streams, where the human ear copes well with the low quality substitute because it is sufficiently close to what was expected. Unlike FEC, this form of redundancy has the big advantage that it degrades gracefully as loss rates suddenly increase.

- **Sample or time-slice interleaving**. In its simplest form, this involves taking two packets worth of audio samples and putting sample 1 in packet 1, sample 2 in packet 2, sample 3 in packet 1, and so forth. When one of the two packets is lost, alternate samples from both time-slices are lost, and it is fairly easy to interpolate to recreate the missing samples with some loss of quality. Simple sample interleaving doesn't work well with any significant compression scheme, but if the compression scheme can be made to work with time-slices on the order of 10ms, and these time-slices are alternated between packets, then whole phonemes are not lost when a packet is missing, and techniques based on repeating the previous time-slice can fill the gap with only a small amount of degradation. With larger playout buffers time-slices can be alternated between more packets, which spreads the damage caused by loss over a longer time interval in the final audio stream, and so lessens its noticability.

These schemes can also be combined in different fashions. Note that audio is almost always more important to protect than video as the human ear is very good at detecting discontinuities whereas so long as a robust encoding scheme (such as Motion JPEG, or robust H.261[12]) is used, video errors result in parts of the previous frame being retained slightly longer than desired, and this is not treated as such a significant feature by the human eye which is best at detecting motion.

3.3 Multicast

IP Multicast[11] provides efficient one-to-many or many-to- many distribution of data. There are many possible uses of multicast in the web, including using it for cache preloading and update for normal web documents.

In the context of real-time media flows, the simplest use is to have a continuous playback stream from a server, and have receivers join the multicast at whatever point it has reached, rather than actual video-on-demand. This solves the traffic concentration problem of attempting to get many streams for the same data out of the same server simultaneously.

An extension of this for busy servers requiring on-demand playback would be to cache requests for a few seconds and then multicast the multimedia clip to all the receivers that requested the clip in that time interval. Fast forward and rewind can be performed by skipping between the multicast groups for different time-slices producing a near-video-on-demand effect at the expense of many simultaneous streams (though much fewer than if unicast were used).

Multicast also leads to interesting possibilities for caching, as the cache does not need to be on the actual data path from the server to the requesting client so cache loading is possible without relaying the traffic directly through the cache.

3.4 Layered Encodings

The concept of a layered encoding[13] has been around for a long time, but it is only recently that suitable codecs have been developed that are well suited to the internet environment and to producing many layers. A layered encoding is a compression scheme which produces a number of separate media streams which can be combined to increase the quality of the encoding.

For multicast conferencing, layered encodings are a valuable technique because, as each layer is sent to a different multicast group, they allow receivers to join only as many groups as there is avail-

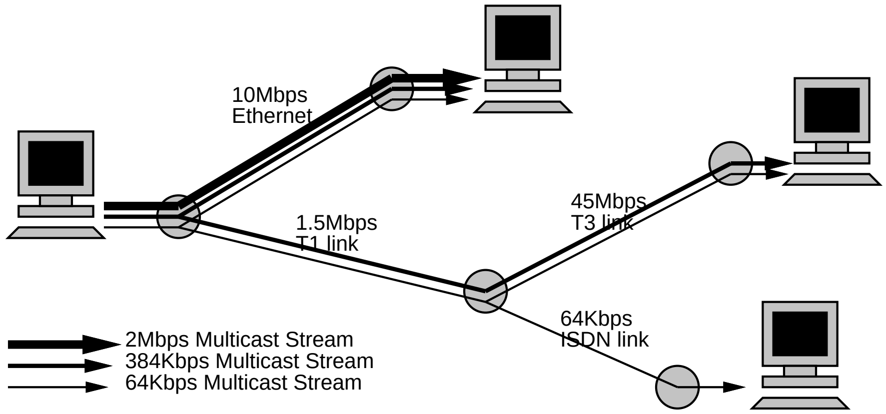

Figure 2: Layered Encoding Supporting Heterogeneity in a Multicast Session

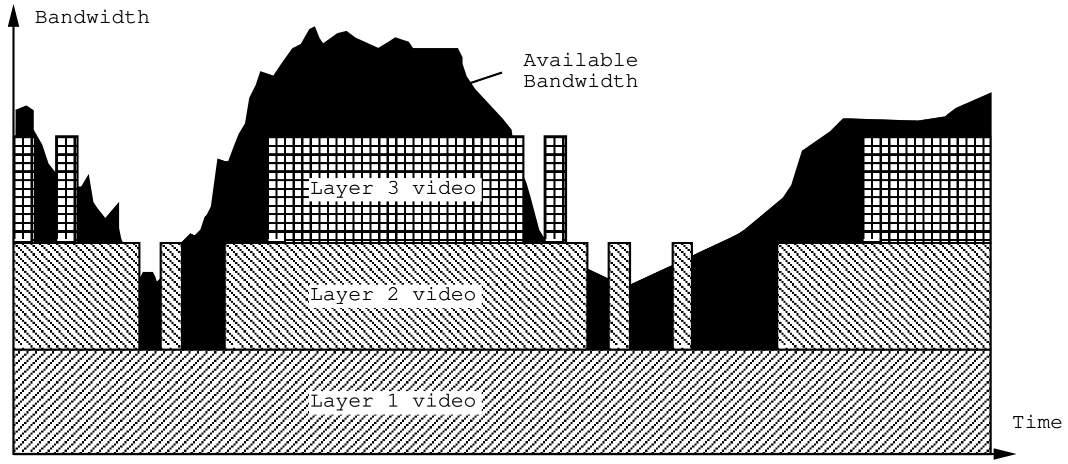

Figure 3: Layered Encoding for Congestion Control from a Multimedia Server

able bandwidth to carry, and so receivers with good connectivity get good quality and receivers in the same session with less available bandwidth get lower (but near optimal) quality with low loss. This is illustrated in figure 2. For servers, the same property is useful if media streams are being multicast.

For unicast, they also allow a compressed data stream to be stored on the server as a layered encoding, and for congestion feedback from a receiver to select how many layers to send without the server having to recompress the data on demand - a task that is unlikely to be possible on busy servers. This is shown in figure 3.

3.5 Caches and Mirrors

Caching is probably the single most important aspect of web development because it one of the few techniques that changes the scaling properties of the web. In general, much web traffic is long distance, because much of the web is long distance. Caching changes this, and so in improving locality, significantly changes the requirements upon the internet backbones, improves performance, reduces delays, and drastically reduces the cost of network provision on inherently expensive long distance links such the transatlantic connections. All these arguments become even more important when huge amounts of streaming multimedia data involved.

Caching for RTP data flows is significantly different from caching for normal web objects because packet loss usually means that parts of the stream being cached are missing. When recording RTP streams in multimedia conferences, the same problem exists. The typical solution is to store the stream as received *packets* as this preserves the timing relationships in the original data plus it keeps low quality redundancy separate from the stream it will be used to patch.

When a cached stream with loss is requested for playback, the cache server can contact the original server and have the original server fill in the gaps in the stream as the cache server is playing out the cached packets. If unicast is being used, the cache server will have to relay those packets, but if multicast is being used, both servers can play the stream out together, with the cache server playing most of the packets and the original server playing only the missing packets[4].

Another technique, perhaps a little further off, is to have several mirrors of a data stream, and to link these mirrors together so that each mirror site sends only a share of the packets. With UDP, so long as the socket is bound to a wildcard source, several servers can each send to the same port on a single client simultaneously. Two or more servers can cooperate to send one flow (load-sharing) and the source of the data can even be migrated between servers in real-time depending on network loss conditions, partially filled caches, and so forth. To do this the servers must know of each other and know which data each holds, but the control messages to exchange this information are small compared to the amount of data in a typical real-time stream. Neither client nor servers need know which server is closest to the client - closest is not as relevant as which path is best, and this can change dynamically. Using multiple servers in this manner can track such changes effectively. However, looking at the large scale picture, care must be taken to ensure that the whole worldwide mesh of real-time flows from servers doesn't exhibit oscillatory behaviour as servers adapt to each others adaption.

3.6 SDP

The Session Description Protocol (SDP[5]) is a way to describe multimedia sessions, defining multiple media streams, their destinations (typically multicast addresses), ports, protocols, formats, bandwidth and arbitrary attributes. It was developed for multicast session directories and is in wide-spread use on the Mbone. SDP is ASCII text based, and can equally well be conveyed by HTTP or SMTP as by the Session Announcement Protocol[3] which conveys it in session directories. An example of a simple session description is:

```
v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=Sdr Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.02.1.ps
e=M.Handley@cs.ucl.ac.uk (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 3456 RTP/AVP 0
m=video 2232 RTP/AVP 31
c=IN IP4 224.2.17.13/127
m=whiteboard 32416 UDP WB
c=IN IP4 224.2.17.14/127
a=orient:portrait
```

[4] IP multicast allows senders *not* to be a member of the group, so the original server can playout data without receiving the data sent from the cache

3.7 SIP

The Session Invitation Protocol[4] is another way to convey SDP session descriptions. SIP provides a mechanism to call a user or service to invite them to join a session, and so it is analogous to the dialing part of making a telephone call. SIP is UDP based which allows it to be multicast if required, and it has provisions for both referral and chaining of calls to other users or services.

3.8 The Multimedia Conferencing Stack

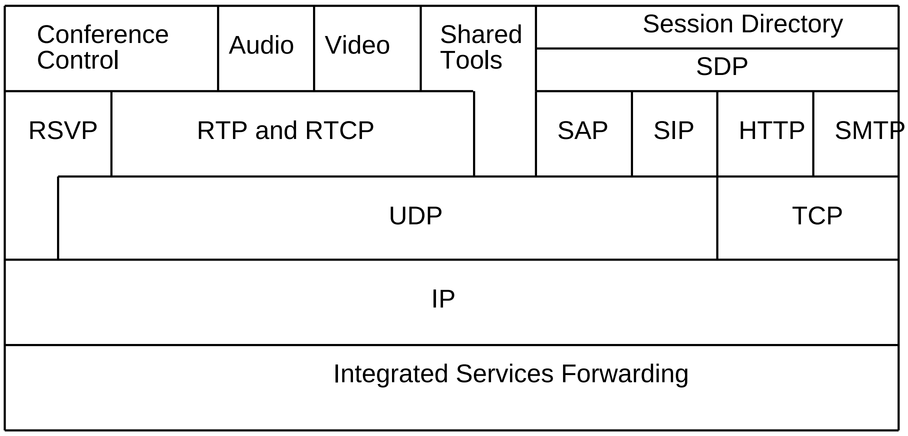

Figure 4:

Figure 4 illustrates how these protocols fit together for multimedia conferencing. SDP can be used over many different transport protocols, to initiate multimedia streams. In many cases a web browser can be used as a session directory to initiate sessions, and helper tools or plugins can be used as media decoders.

4 One Solution

The multimedia conferencing community has a need for multimedia servers to record live multimedia sessions and to play them back; both play-back on a one-to-one basis, and play-back into a pre-existing live session. Clearly integrating such multimedia servers into WWW infrastructure is necessary.

The solution currently being developed at UCL and a similar system being developed by Jeff Smith at NTT use SIP and RTP to initialise and control such servers. The scenario considered is that multimedia servers exist on the internet, and that these servers can be remote controlled both to record and playback sessions.

4.1 Recording

Initiating a recording is performed (see figure 5) by making a SIP request to the server, inviting the server to join the session. This SIP request conveys an SDP description of the session to be recorded and includes an extra "media stream" which specifies the use of the Server Control Protocol[5] (SCP) for control of the recording. When the server responds to the SIP request agreeing to make the recording,

[5] SCP is currently being specified

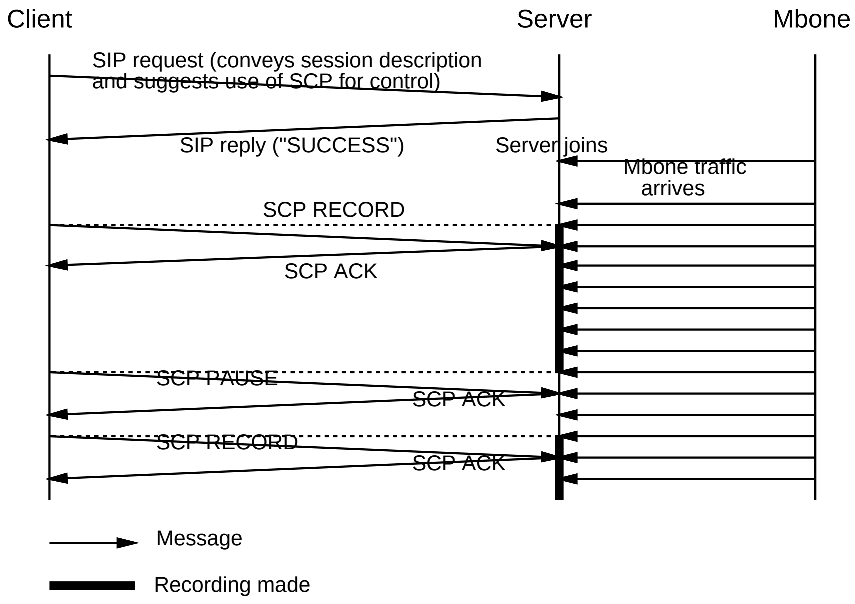

Figure 5: Recording a Multicast Session

it is at “record-cue” status, awaiting an SCP message to tell it to start. The SIP response contains the “filename” to be used to access the recording after it has finished. SIP is now no longer involved in the session - control is handed to SCP. In the meantime the server has joined the appropriate multicast groups, is receiving the traffic, and is keeping a short cache of the incoming data streams.

SCP messages quote the RTP timestamps giving the time the command should have taken take effect, so when the server receives the SCP “Record” command, it can look back in this short cache to start recording at the moment the client pushed the record button, so eliminating control lag in spite of network propagation delays.

Other SCP messages allow pausing the recording, muting recording of individual streams or users, and terminating the recording.

4.2 Playback

For playback, there are two scenarios which we consider.

Playback to Active Session

In the first scenario (see figure 6), a multicast session is currently taking place, and we wish to playback a previously recorded session into the existing session. In this scenario, we send a SIP request to the server asking it to play the given filename into the session. The SIP request specifies using SCP for control, and it gives the SDP session description of the session currently underway to play the recording into. The server merely replies with whether or not it can comply, and if it can, control is handed over to SCP. In this scenario the request is composed dynamically from the details of the session underway.

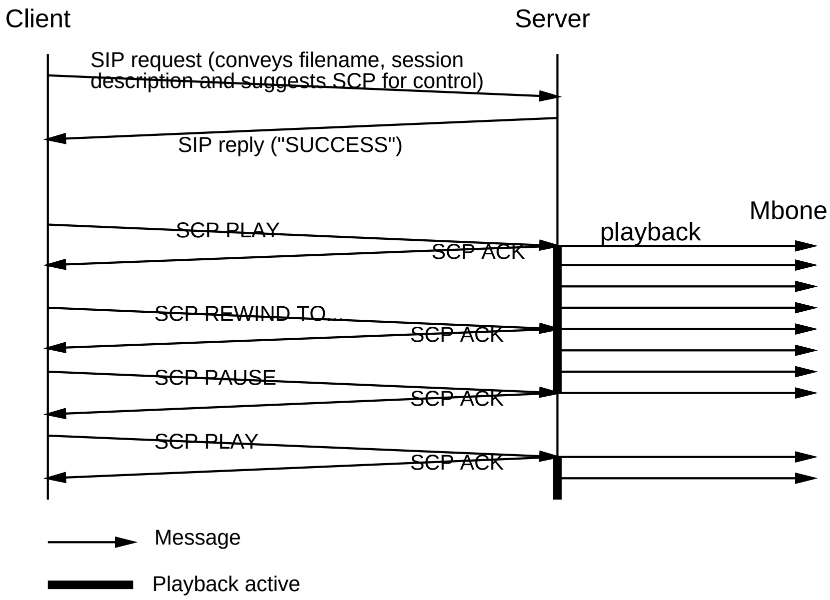

Figure 6: Playback of a Recorded Session into a Multicast Session

Dedicated Playback Session

In the second scenario (see figure 7), the recorded session has been indexed on a web server, and a user merely wishes to play the session back to herself. In this case, the client can merely make an HTTP GET request to the server for the "file". In response, the server sends a description of the media in the recording using SDP and a MIME Content-Type of `application/x-sdp`. The server also adds an extra "media" to the reply to specify that SCP should be used for control. Control is then handed over to SCP and the server is in a "pause-play" state.

4.3 SDP and Embedded Decoders

In section 2.1 we made the claim that multimedia URLs were a bad idea, and that a MIME-equivalent designed for this purpose such as SDP should be used instead. For a multimedia session SDP has the right properties, grouping the media streams together and allowing the specification of session wide attributes. However, this seems at odds with embedding the individual media streams comprising a multimedia session into an HTML page, where we would really like separate tags for each medium.

This can be handled using standard URL syntax for tags within a page. For example, if we wish to embed a video stream and a whiteboard from a multimedia session, we might have the following HTML:

```
...
<HR>
<AUDIO href=http://www.x.uk/session17.sdp#audio>
Take a look at this cool video:<br>
<VIDEO href=http://www.x.uk/session17.sdp#video><br>
<WHITEBOARD href=http://www.x.uk/session17.sdp#whiteboard>
<HR>
...
```

The precise details of the HTML don't of course matter in this example - the point is simply that a set of embedded media all reference the same URL, and when any one of them is selected, they all

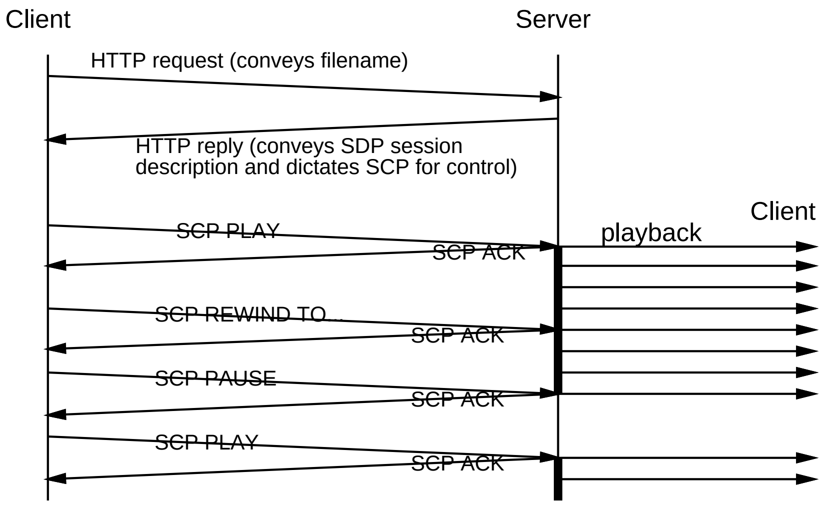

Figure 7: Playback of a Recorded Session to a Web Client

are activated. The tags simply identify which of the media streams described by the SDP description should be displayed.

There may be several (logical) video sources arriving at a single video decoder simultaneously as RTP allows multiplexing of multiple streams onto a single flow from a single source host. Thus the tags can also identify the individual sources and create appropriate video frames:

```
...
<TABLE>
<TR>
<TD>
  Mark Handley <BR>
  <VIDEO href=http://www.x.uk/session17.sdp#video/mark@128.16.64.4>
</TD>
<TD>
  Jon Crowcroft <BR>
  <VIDEO href=http://www.x.uk/session17.sdp#video/jon@128.16.64.16>
</TD>
</TABLE>
...
```

5 Conclusions

The Multimedia conferencing community and the Web community have very similar requirements for stored multimedia streams. Many of the techniques developed in the conferencing community will work well for the web community, but there are still many hurdles to be cleared. In particular, there are many in the Web community do not have a knowledge of multimedia conferencing and show reluctance to use a wider range of transport protocols than have been in common usage on the web.

The framework that has been growing up for multimedia conferencing on the internet is both powerful and flexible. It has to include storage of multimedia sessions and their playback, and in so doing there is much overlap with similar work in the web community. When deciding on mechanisms for multimedia stream playback and their control both communities much pool their efforts and converge on a common set of formats and control protocols. This paper has attempted to suggest some that might be appropriate, but its suggestions are by no means exhaustive. Much work remains to be done.

References

[1] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, RFC 1889

[2] Sdr, “UCL Session Directory Tool”. ftp://cs.ucl.ac.uk/mice/sdr/

[3] M. Handley, “Session Announcement Protocol”, Internet-draft, ftp://ftp.isi.edu/confctrl/docs/

[4] M. Handley, E. Schooler, “Session Invitation Protocol”, Internet-draft, ftp://ftp.isi.edu/confctrl/docs/

[5] M. Handley, V. Jacobson, “Session Description Protocol”, Internet- draft, ftp://ftp.isi.edu/confctrl/docs/

[6] I. Kouvelas, V. Hardman, A. Watson, “Lip Synchronisation for Use Over the Internet: Analysis and Implementation”, UCL Research Note RN/96/17, to appear in Globecom96 http://www.cs.ucl.ac.uk/staff/I.Kouvelas/publications/sync_globe.ps

[7] S. Casner, V. Jacobson, “Compressing IP/UDP/RTP Headers for Low-Speed Serial Links”, draft-ietf-avt-crtp-00.txt

[8] V. Jacobson, “TCP/IP Compression for Low-Speed Serial Links,” RFC1144.

[9] V. Hardman, A. Sasse, M. Handley, A. Watson, “Reliable Audio for Use over the Internet”, Proceedings of INET 95, Hawaii, Internet Society, Reston, Virginia, 1995.

[10] M. Handley, V. Hardman, I. Kouvelas, C. Perkins,J. Bolot, A. Garcia, S. Parisis, “Payload Format Issues for Redundant Encodings in RTP”, Internet Draft, draft-perkins-rtp-redundancy-01.txt

[11] H. Erikson, “Mbone: The Multicast Backbone”, Communications of the ACM 37, 8, pp 54-60, 1994

[12] S. McCanne, V. Jacobson, “Vic: a Flexible Framework for Packet Video”, Proc ACM Multimedia ’95, ACM, 1995

[13] S. McCanne, V. Jacobson, M. Verletti, “Receiver-driven Layered Multicast”, Proc. Sigcomm 96, Stanford, ACM Sigcomm 1996.