A NETWORK BASED REPLAY PORTAL

Service Scenarios, Architecture and Implementation

J.E. van der Merwe[1], C.J. Sreenan[2], A.N. Donnelly[3],
A. Basso[1], C.R. Kalmanek[1]

[1] *AT&T Labs - Research*
Florham Park, NJ, USA
{kobus,basso,crk}@research.att.com

[2] *University College Cork*
Cork, Ireland
cjs@cs.ucc.ie

[3] *Cambridge University*
Cambridge, UK
and1000@cl.cam.ac.uk

Abstract

Technologies based on cable modems currently use the capacity of a single TV channel to offer 25 - 30 Mb/s downstream for Internet access. With the advent of Digital TV and the significant bandwidth savings it gleans from video compression, it is expected that providers will increase the access capacity available for IP traffic, while retaining the bulk of the bandwidth for the primary service of broadcast TV. Motivated by the increasing popularity of on-demand streaming media, our work questions this IP-versus-broadcast distinction, and proposes a hybrid model which combines the familiar broadcast model with the conveniences of on-demand TV viewing over IP. In this paper we discuss the potential benefits of this approach, and some of the difficult technical challenges it raises. We propose an architecture of network-based portals and sketch the types of services we envisage it will enable. Further, we describe the design and implementation of a replay service that operates within this architecture to offer a new on-demand TV-viewing experience.

Keywords: streaming media, network services, on-demand TV

Introduction

Recent years have seen the rising popularity of streaming media applications on the Internet. While the amount of this traffic is still small relative to web traffic [van der Merwe et al., 1999], there is general agreement that it will grow significantly, driven by bandwidth-efficient compression techniques, a wider array of globally accessible content, and increased capacity of backbone and access networks. Our research is motivated by this expected growth in Internet streaming content and by developments in the Cable-TV industry, where there is a concerted effort to provide high-speed, full-duplex Internet access to residential customers. Current standards in the US allow up to 30 Mb/s for Internet access by allocating a single 6 MHz wide TV band. In just a few years it is expected that the broadcast TV industry will fully embrace and deploy digital TV based on MPEG-2. The benefits of digital compression will allow several TV channels to be carried in a single 6 MHz band, freeing up a significant amount of system capacity. It is expected that this will be mainly used to offer new premium TV channels, and new services such as movies on demand, while allocating a relatively much smaller amount of bandwidth for IP traffic, including streaming media. Our work questions this TV versus IP distinction and proposes instead an all-IP model. This approach has several potential benefits, including the ability to use multicast delivery to efficiently scale from large broadcast-style distribution to smaller, more select audiences. Also, the Internet provides access to content which is globally accessible, unlike the broadcast TV model, where access to content depends on the channels offered by your local TV operator. Integration with the web also becomes much more convenient and promotes interactive services.

We believe this approach has the potential to revolutionize the TV viewing experience by moving away from a broadcast model to a hybrid on-demand model. In that context, several difficult technical challenges arise, including the design of on-demand services, storage management, efficient use of multicast, caching models, and protocol interactions for live and on-demand viewing. In Section 1 we propose an architecture of network-based portals and sketch the types of services we envisage it will enable. In Section 2 we describe the design and implementation of a replay portal that operates within this architecture to offer a new on-demand TV-viewing experience. Section 3 provides a summary of related work, and Section 4 concludes the paper.

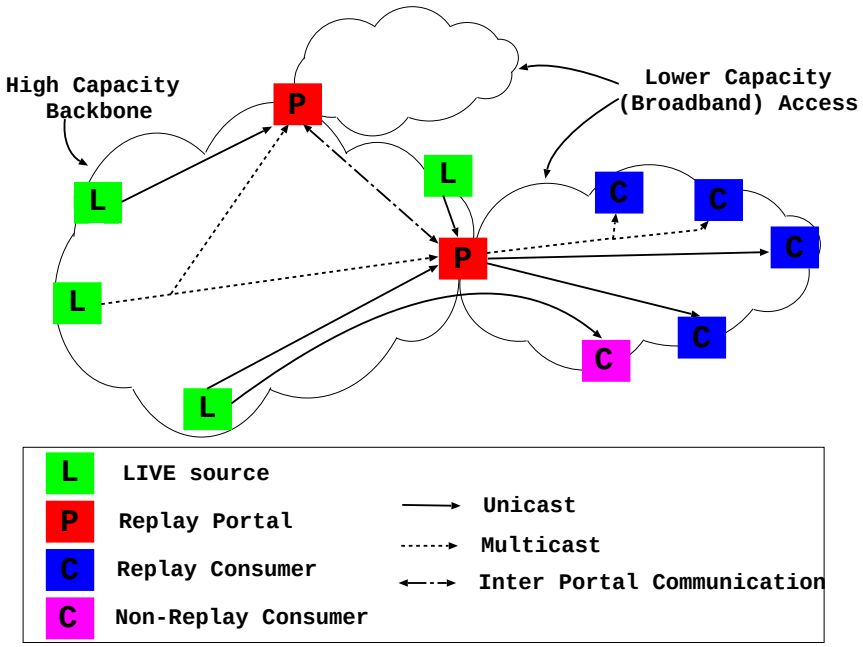

Figure 1 Network Based Replay Service

1. NETWORK BASED REPLAY SERVICES

Our basic assumption is that we are dealing with an environment where high quality live video is being distributed across an IP network. This live content might enter the IP network "locally" (e.g. from a satellite feed at the head-end of a local access plant) or might indeed be carried on IP from the remote live source. This world essentially duplicates or emulates the "pure broadcasting" (or more correctly for IP, multicasting) model of current TV networks. In our architecture we add the notion of a **Replay Portal** to this basic infrastructure to change the broadcasting model to a hybrid on-demand model. This arrangement is depicted in Figure 1 which shows a high level view of our architecture, its different components and variations.

A Replay Portal becomes the local video access point for customers and provides the following functions:

- Access to live content (subscribed to by portal on behalf of users or based on the content offering of the replay service).
- Moving window recording of recent (e.g. the last 24 hours worth of) live content, enabling on-demand viewing of such content.
- "Pause" and "Replay" functionality of live content.

- Personal recording facilities.
- Subscription to non-local live content, (obtained on subscription or on-demand basis from other Replay portals).
- Indexing and search functionality providing access to the video content of multiple cooperating portals.

As shown in Figure 1, content is delivered to the portal either by unicast or multicast, or indeed by a special content distribution mechanism. Similarly, from the portal downstream, content can be delivered by means of multicast, as would be the case when live content is watched through the portal, or unicast when previously stored content is watched on-demand. Customers that do not make use of this service, but are located on the same access network can connect directly to the original live sources (assuming that this is allowed by the live content provider) as they would do in the absence of the replay server. (Such customers will of course not be able to make use of the replay portal functionality.)

The high quality service we envisage will only be realizable on broadband access networks. While these access networks will increase the access capacity with an order of magnitude or more [Eldering et al., 1999], access capacity is expected to lag behind backbone capacity for a considerable time to come. We position the replay portal at this capacity discontinuity. We view our architecture as enabling the replacement of current TV offerings and as such schedule based streaming of content from the headend is still the basic service offering. However, in our architecture, video is only streamed downstream of the portal if there are actually consumers of a particular stream downstream of the portal. This can easily be achieved in an IP environment, where streams are delivered via multicast rather than broadcast, and is expected to result in bandwidth savings across the access network.

For example, in a cable based access network the portal is expected to be located in the cable headend. As access capacities increase, the capacity discontinuity, and the portal location, will move downstream towards the home. Eventually, in a fiber to the home scenario, the discontinuity will disappear or move into the home. When this happens, the bandwidth savings envisaged by our architecture will become less important but the service interaction and integration aspects discussed in Section 1.1 will be as important only at a much larger scale.

1.1. SERVICE SCENARIOS

In this section we motivate our approach by considering a variety of services or service features enabled by our IP-based architecture. We require the basic functionality to be equivalent to current TV and as

such the basic service offering is still schedule driven access to a variety of live channels. These live channels are transmitted downstream by means of multicast delivery. The services and features considered below provide some enhancement to this basic service.

Moving window replay of subscribed channels: For a predetermined set of channels the portal stores a moving window of the most recent N hours worth of content for each channel. This stored content is made available to subscribers for on-demand viewing. Different ways of indexing can be provided to this stored content ranging from simple time based schemes to indexing that is content aware.

Library archive of popular programs: A complementary way of providing access to previously recorded content is to maintain a library of certain popular programs in the replay portal. For example, all programs in a certain series such as the X-Files or Star Trek can be archived for subscribers to the portal service.

Replay and pause of non-portal-subscribed channels: Customers of the portal service can also watch other live content, i.e. content not subscribed to by the portal, through the portal. In this case the portal will contact the actual live upstream server. Content is streamed to the downstream customer via the portal which stores a small moving window (say M minutes) of the stream. This small stored window allows customers to request a replay of a recent part of the stream, or pause the live stream (causing the window size to increase) and to then join the live stream again.

Network-based personal recording: A small extension of the above services allows customers to make their own personal recordings which are stored in a personal account on the portal. Reliable recordings can be initiated in a variety of ways and the content can be from either subscribed or non-subscribed channels. Note that combining this functionality with the moving window store allows a user to record content after it has been "aired". In this manner a user can "retro-actively record" something from the replay store thereby adding it to the user's personal collection.

Friends access to personal library: Since the portal is located in the network and on a high capacity backbone it is possible for users to allow access to their personal library of recordings to friends. A user might for example see something that he/she knows is of interest to a friend and start recording it. Having finished the recording the user can simply mail a pointer to his/her friend who can then stream (or transfer) the content from the portal where it was recorded.

A very powerful extension of the above service offerings, enabled by the fact that the portal is network based, is to allow interportal exchange

of content. The novelty, over current video-on-demand offerings, is that the user has control over the source of the material.

Subscription based exchanges: The simplest form of interportal exchange will be interportal-subscription based. In this case content stored by a remote portal is transferred to the local portal for on-demand viewing by local customers. Certain non-local channels (stored by a remote portal) might be of sufficient interest to the local community to warrant it forming part of the local portals regular offerings rather than having customers access it from the remote portal directly. An example might be regular (or seasonal) European sporting events made available in the U.S.

Personalized content aware exchanges: A much more sophisticated means of content exchange between portals might involve users specifying a profile of interest, with portals exchanging content based on the profiles of its local users. In this way users are ensured of receiving up to date streaming content on the topics they find interesting.

In the above examples the assumption was that content produced by some live sources was stored, indexed and made available for retransmission by the portals. Such actions will clearly require some agreements between the portal operator and the producers of live content, and a number of business arrangements are possible between content providers, portal service providers, customers and advertisers. However, in that scenario the mode of operation of content providers remains unchanged from the current cable TV scenario. A logical extension of this involves negotiation between the portal operator (or a third party that use its infrastructure) to directly negotiate with the content provider for specific content:

Local target audience: In this case the portal becomes the access point for a particular mix of programs targeted at a specific audience. At one end of the spectrum this might resemble the service currently offered by a local TV station. The key point however is that basing this architecture on a packet network allows this type of service to scale to arbitrary small target audiences. For example the the target audience might be the local bird-watchers club that obtains (and adds to its library) programming information of interest provided by a variety of content providers.

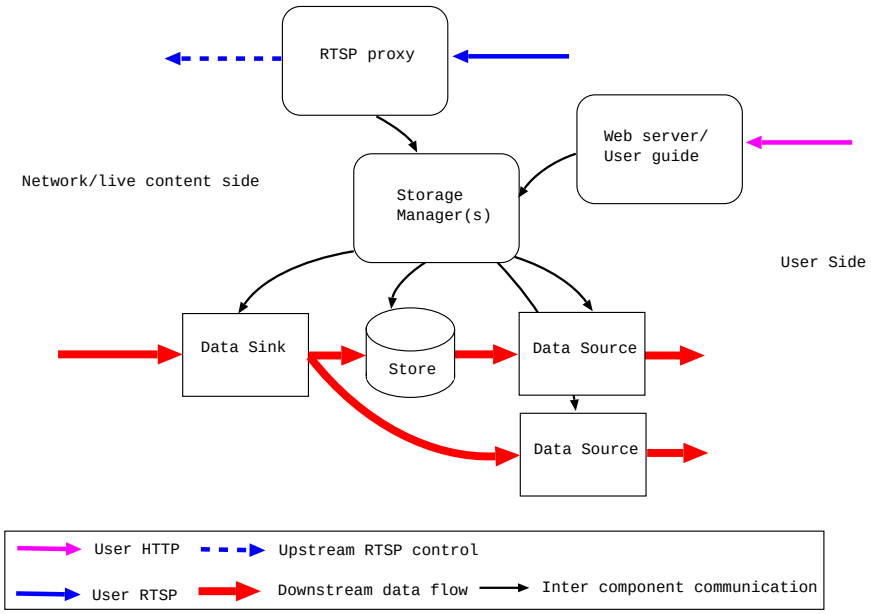

Figure 2 Replay Portal Architecture

2. REPLAY PORTAL DESIGN AND IMPLEMENTATION

2.1. PORTAL ARCHITECTURE

The main architectural components of a Replay Portal are shown in Figure 2. Our architecture is built around standard IETF protocols namely the Real Time Streaming Protocol - RTSP [Schulzrinne et al., 1998], the Real Time Transport Protocol - RTP [Schulzrinne et al., 1996] and the Hypertext Transfer Protocol - HTTP [Fielding et al., 1999]. A user typically starts interaction with the portal by means of accessing a portal **Web-server/User-guide**. This interface provides the user with personalized access to and control of the portal content. Personalized portal content includes the portal-subscribed content (either live or on-demand) as well as any content stored in the user's personal store. The Web interface offers a number of ways of indexing the content that is of interest to the user and allows the user to initiate streaming of any such content. In the case of a personal store the user can also perform management functions such as removing previously stored content or setting up the recording of a future streaming event. When a user initiates streaming through the portal Web interface, a helper file is downloaded

to the user's browser. The Mime type of this file instructs the browser to start up a streaming client application on the user's PC or set-top box passing to the application the RTSP URL contained in the helper file.

A user might also make use of the portal for content that is not subscribed to by the portal. In this case the user would not typically make use of the portal web interface. Rather the user will go to a web interface associated with the content source and obtain an RTSP URL in similar fashion as described above. This also means that in this case the user's first interaction with the portal will be through the RTSP interface as described next.

The RTSP URL obtained by the streaming client through either of the above approaches is presented to the **RTSP proxy** on the replay portal. The proxy establishes whether the URL represents content currently stored in the proxy or whether it is necessary to establish a connection to an upstream server. If the requested content is available on the server, either live or stored, the proxy initiates delivery of the content to the client. (In these cases the proxy would have contacted the relevant upstream servers beforehand.) If the content is not locally available, the proxy will contact the upstream server and on success will initiate local handling of the content as well as delivery to the client.

The actual manipulation of content on the portal is performed by a set of **storage managers**. Each storage manager is in control of a specific physical data store and controls the way content is added to and removed from the store. The storage manager provides the Web interface with information about the contents of a particular store, for example to create an RTSP URL to pass back to the client. Similarly the storage manager can tell the RTSP proxy whether a particular URL is currently to be found in the local store. The storage manager(s) manipulate the store(s) under control of the RTSP proxy. For example, in the case of live content being viewed through the portal, the RTSP proxy will instruct the storage manager to create a **data sink** and a **data source** for the data path handling of the stream. The data sink receives the content from upstream and writes it to the store, while also making the content available to the data source for immediate delivery to the client.

The portal architecture lends itself to a number of implementation options depending on the required scalability. In the simplest case a small replay portal can have all the components executing on a single physical machine. This is the nature of the prototype implementation which is discussed in more detail in the remainder of this section. A more scalable realization could involve **frontend web and RTSP servers** which hand off processing of streaming content to a farm of **backend**

RTSP servers and storage managers. This arrangement is depicted in Figure 3. In this case access to the portal through the web interface will result in one of the backend servers being chosen based on server load, the content to be accessed or some other policy. Similarly direct accesses to the portal RTSP interface, handled by the RTSP frontend, will be handed off to one of the backend servers.

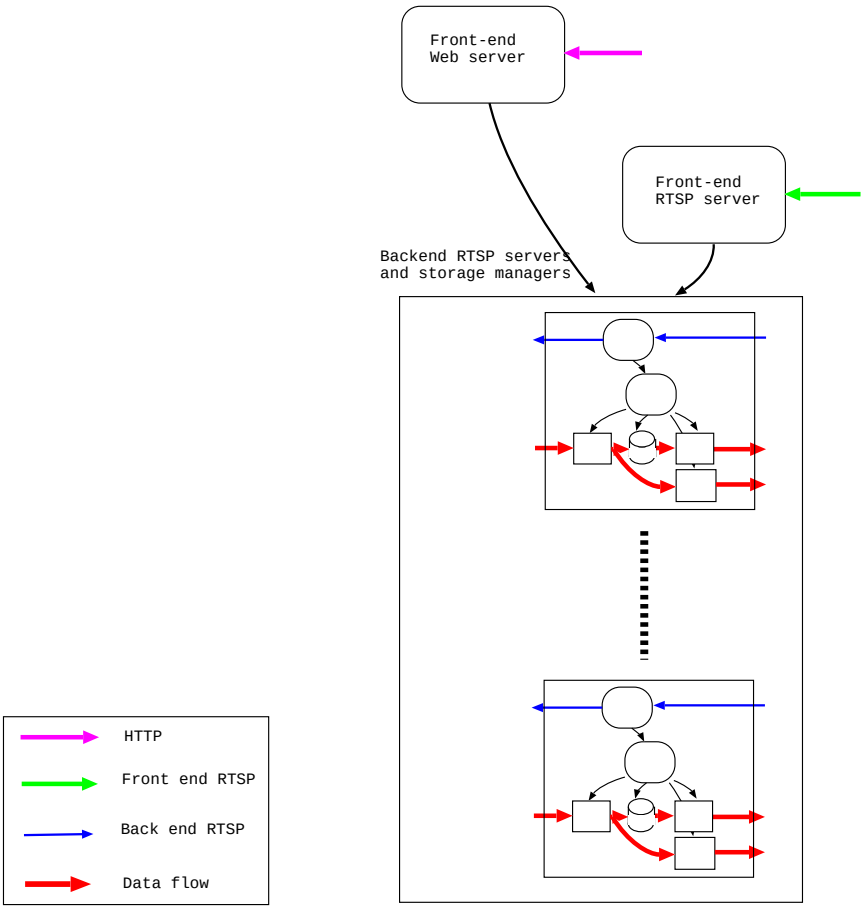

Figure 3 Replay portal with RTSP proxy/storage manager farm

The data path handling of streaming content can similarly be realized in a variety of implementations. Again in the simplest case an RTSP server, storage manager combination can simply execute on a single server machine potentially with two network interfaces. In such an implementation the server could however easily become a bottleneck, as

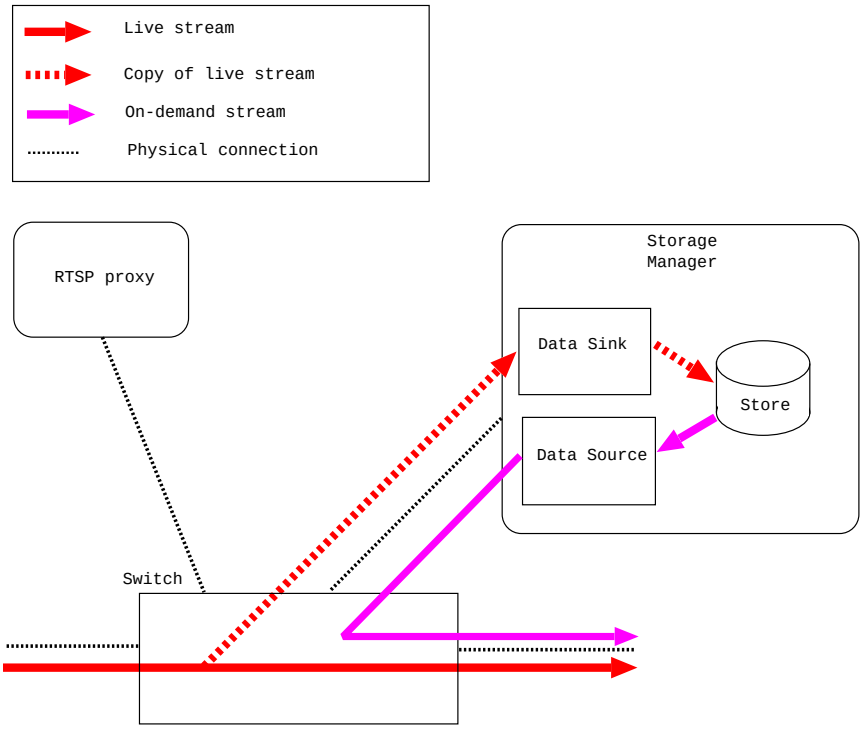

Figure 4 Scalable portal realization

it has to handle re-delivery of any live streams as well as any on-demand delivery of streams. An alternative realization is depicted in Figure 4. In this case an RTSP proxy and its associated storage manager is separated by means of a forwarding device such as a switch or a router. As before the storage manager is effectively controlled by the RTSP proxy based on user requests. The RTSP proxy also has some control over the forwarding device. In particular the RTSP proxy can instruct the switch to have a **copy** of a particular stream delivered on the switch interface connected to the storage manager. As before the RTSP proxy instructs the storage manager to expect and store this stream. In this case the storage manager does not handle the live stream at all and is only responsible for handling any on-demand requests.

While we do not expect any major obstacles in realizing the portal architecture in a scalable manor, many details need to be worked out and this is the subject of current and future work. In the remainder of this section we will concentrate the discussion on our current prototype implementation.

2.2. PROTOTYPE IMPLEMENTATION

In order to demonstrate the feasibility of our architecture we have developed a prototype system consisting of all the elements in our architecture:

- A live server
- A replay portal (consisting of web server, RTSP proxy and storage managers)
- A streaming client

Since we expect to provide a high quality service we use MPEG2 encoding for the video streams making use of hardware encoders and decoders we have used in earlier work [Basso et al., 1999]. We use hardware encoders from VisionTech [Vision Tech, 2000], while the decoders are from SigmaDesign using a Microsoft Windows environment [Sigma Designs, 2000]. RTSP is the control protocol that binds all our components together and we have developed an RTSP library (librtsp) which has been derived from an early public domain implementation from Real Networks [RealNetworks, 1999]. The portal was implemented on a Linux infrastructure and the web server is an unmodified Apache server [Apache Software Foundation, 2000]. Since we knew from the outset that we would be dealing with a diversity of platforms and operating systems, code portability was a major concern. We addressed this by developing a basic portability library (libcommon) that dealt with operating system specific issues and provided a common interface to other libraries and applications.

Each of these libraries and the applications built on them are discussed in more detail in the sections below.

2.2.1 Support libraries: libcommon and librtsp. The main functions provided by libcommon are an event scheduling mechanism and IO handling of both network and file systems across all supported platforms. The event scheduling mechanism allows specific functions to be called based on time, network or file events. This include the running of "background" tasks when the system is idle. Libcommon also contains a number of general mechanisms such as safe string handling and ring buffer and table manipulation.

Librtsp builds on libcommon and provides a simple way for either client or server applications to use RTSP. For example a client application simply calls "rtsp_connect" to initiate communication with an RTSP server. On success the client obtains a handle with which all further interaction with the server (i.e. describe, play etc) is conducted

through a remote procedure call (RPC) like interface. The library deals with message formatting and parsing and presents the content of messages to the application in the form of well defined structures, or form RTSP messages out of structures provided by the application.

2.2.2 RTSP client and server. The structure of the client software is depicted in Figure 5. A graphical user interface (GUI) allows the user to specify the RTSP URL of interest and initiate streaming. (As explained earlier an alternative is for the URL to be supplied to the client software by means of a helper file downloaded by a web browser on the client device.) On successful RTSP interaction with the server, the client sets up a datasink and a ring buffer and initiate the MPEG hardware decoder. The datasink receives an RTP encapsulated MPEG stream from the network, strips off the RTP encapsulation and puts the MPEG packets in a ring buffer for asynchronous collection by the MPEG decoder hardware. The decoder driver performs an upcall into the application whenever its buffers are below a certain threshold at which point data is transferred from the ring buffer to the decoder.

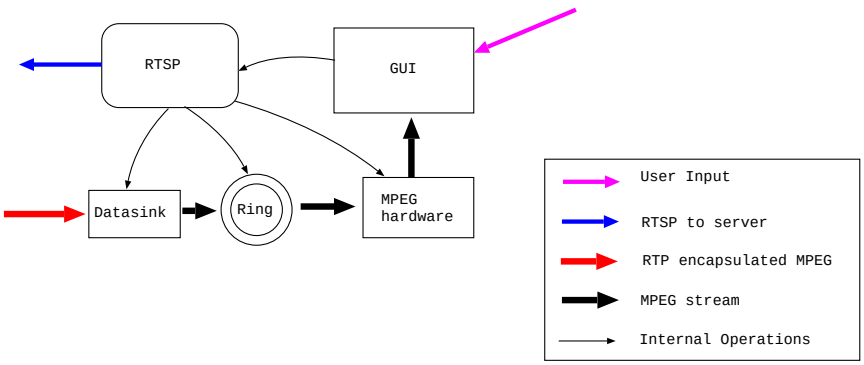

Figure 5 RTSP client

Figure 6 shows the main components of our RTSP server implementation. RTSP requests received by the RTSP library are passed to a **Media Manager** which determines if there is a media backend that can handle a request of this type. A number of media specific backends have been implemented namely backends for MPEG2 audio, MPEG2 transport and WAV streams. These backends deal with media specific issues such as the frame format of streams, the rate at which streams should be played out and how to encapsulate media frames in RTP. The content on which the backends operate can be either stored on disc or

be supplied in real time from an encoder. For example, our live server is implemented as an encoding thread which supplies an MPEG stream to an MPEG transport stream backend.[1]

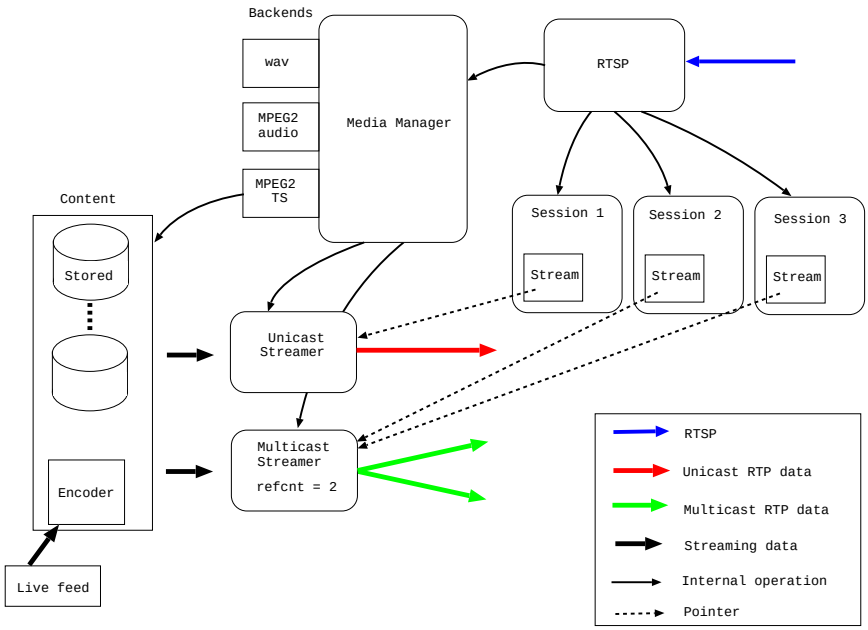

Figure 6 RTSP server

Once the Media Manager has determined that the requested content is available (i.e. a successful RTSP DESCRIBE interaction), the client application normally issues RTSP SETUP and PLAY requests. The SETUP request results in session state being created in the server and a **streamer** is initialized to deliver the stream. A PLAY request starts delivery of the stream. The session state contains stream information that is relevant **for this stream for this session** (e.g. the time a session joined a stream), whereas the streamer contains only session independent information about the stream. This separation is important in order to deal with multicast streams. The first client to request delivery of a multicast stream will result in a streamer being created. Subsequent sessions for the same stream will be served by the same streamer and

[1]Currently only the live server makes use of threads as the encoder hardware and SDK operates as a threaded application on the Solaris platform.

a reference count in the streamer ensures that the streamer does not disappear when the initial session is terminated.

2.2.3 RTSP proxy and Storage Managers. The RTSP proxy functionality required by our replay portal is realized by having the proxy as another media backend. As is the case with other backends, the proxy backend determines whether a request can be satisfied from its local stored content. However in the case of the proxy, the server address of the RTSP URL is not the proxy address and if the request can not be satisfied locally the proxy backend can issue an upstream RTSP request to the server specified in the URL. (In our current implementation the client has to be configured to establish an RTSP connection to the proxy server rather than the real content server.)

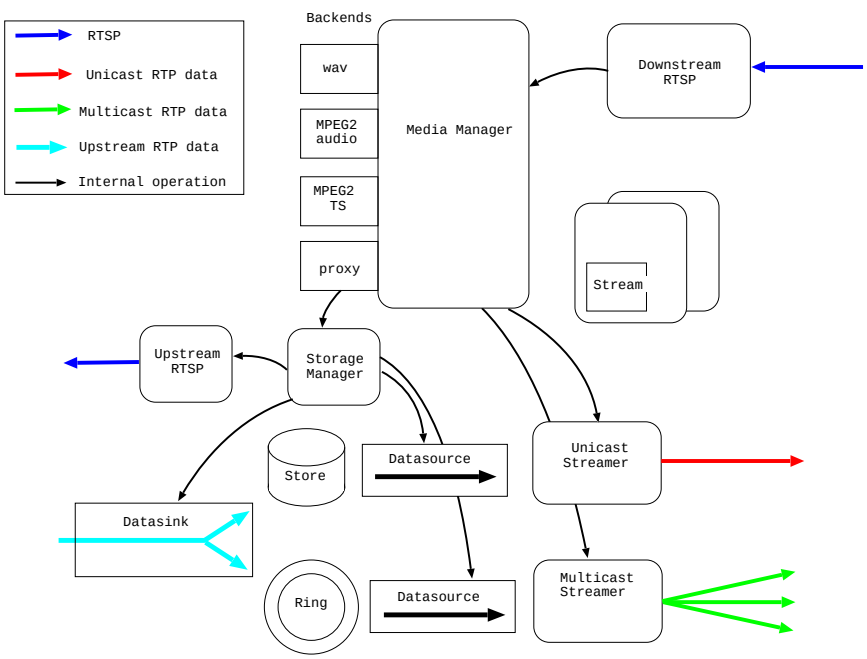

Figure 7 RTSP proxy and Storage manager

If the request can be satisfied from the local store, a streamer is set up as described above and the stream is delivered to the client. If content is received from upstream, a datasink will receive the packets writing them to disk and putting a copy in a ring buffer for delivery to live viewers of the stream (if any). In the case of the proxy backend, content is stored to disk with the RTP header it was received with intact. Sub-

sequent playout of stored content is then based on the RTP timestamp of the stored packets while the RTP sequence numbers are replaced for retransmitted downstream delivery. Storing content in the proxy with RTP headers intact has the desirable property that our proxy is media independent so long as the stream is delivered using RTP and the clock frequency used for RTP timestamps is known from RTSP interaction.

The storage manager(s) handles the manipulation of stored content. This includes the eviction policy associated with a particular stream. In the case of portal subscribed content which is made available for on-demand viewing the policy is simply to keep the last N hours worth of content. This is currently implemented as a logical circular set of files where the oldest file gets overwritten after N hours with new content. In order to have the N hour window move forward in time with a small granularity and to ease time based indexing into the stored content each of these files holds a relatively small amount of data, in the order of one or two minutes worth of content.

In the case of a user watching non portal subscribed content through the portal, the store manager in the proxy still stores some amount of the content to disk. This is needed to facilitate replay of very recent content (i.e. in the order of the last few minutes). However in this case the eviction policy of the storage manager is much more aggressive.

A key aspect of the replay service is that it provides access to arbitrary time offsets into the past of live streams. This requires each client and the proxy to agree on a certain reference point in time in the live stream. We call this reference point the **fixedpoint** relative to which all time sensitive interaction is performed. We rely on a mapping between global time (UTC) and the RTP timestamps in the media stream for this purpose as is described below.

Consider the interaction between a live server and a proxy: The server picks an RTP packet (the first for a unicast stream) that it considers the start of the stream for a particular user and records the absolute time that corresponds to this timestamp as the server fixed point for the session. This information, i.e. fixed time and RTP timestamp is relayed to the proxy via the control channel, e.g. in the PLAY response message. Given this information, the proxy, when it receives an RTP packet can work out the absolute time that the server would have associated with this packet (the clock frequency for the RTP time stamp is known from the RTSP interaction). This absolute time is stored with each RTP packet on the portal and is used for obtaining stored previous live content from the portal. For example, a client might obtain from the portal web interface a URL of the form

```
rtsp://pc-green:8554/live.m2t?pausepoint=utc:19991011T103400Z
```

based on the selection made from the schedule on the web interface. The absolute time in the URL (the pausepoint), represents for example the time when a particular program was aired live, and is used by the proxy to serve the appropriate content by comparing it with the absolute times stored with each RTP packet. The main point here is that all time offsets into the media stream is effectively based on the RTP time stamps of the live source which allows the indexing based on the time the content was aired, which is crucial to our approach.

For per stream (or VCR-like) functions between the proxy and the client we employ a similar approach. Again the server (or more correctly the proxy in this case) gets the absolute time of the first RTP packet it is about to send (i.e. the proxy fixed point), to the client and sends this to the client in a control message. The absolute time for this packet is the absolute time the packet was first sent by the live source not the absolute time when the proxy is about to send it. When the client is about to **play out** this packet it takes a local time stamp which becomes the client fixed point. Now when the user performs a per stream operation, e.g., a PAUSE, the client works out the time difference between the time at which the operation was performed and the client fixed point and adds this to the known proxy fixed point which is specified in the request sent to the proxy as an absolute pausepoint as before.

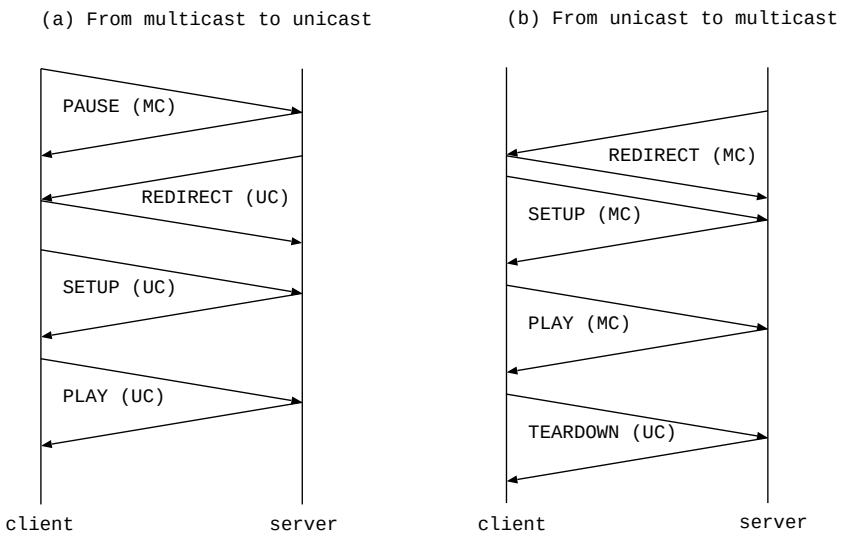

Figure 8 (a) Moving from multicast to unicast and (b) moving from unicast to multicast

A user watching a live event that performs a PAUSE, will have to be switched from (typically) a multicast stream to a unicast (per-user) stream on which these operations can be performed. This can be done by having the proxy send the client an REDIRECT request with the transport parameters for the new unicast stream after the per-stream operation, followed by the client doing a SETUP and PLAY with the new transport parameters as per normal. Figure 8 (a) shows this arrangement. At some later time the proxy might realize that the client's unicast playout point has moved close to the playout point for the live stream. This might for example happen as a result of the user fast-forwarding through the stream. The proxy might then sent another REDIRECT message to the client as an invitation to rejoin the live (multicast) stream. At its discretion the client might then rejoin the live stream by performing a SETUP and PLAY with the multicast transport parameters and eventually tearing down the unicast connection. This is depicted in Figure 8 (b). An alternative is for the client to explicitly rejoin the live (multicast) stream because of for example the user clicking on a "joinlive" button.

3. RELATED WORK

If we focus the discussion on a single replay portal then its functionality is similar to that of consumer electronic devices such as those provided by TiVo [TiVo, 2000] and ReplayTV [ReplayTV, 2000]. The products of these companies are very similar – both sell a combination of a hardware device and a TV listings service. These devices provide an in-home replay service, and do not allow the benefits of content sharing provided by a networked replay portal. A networked solution can also offer higher degrees of reliability, more sophisticated search and indexing, and relieves the consumer of the burden of having to keep pace with advances in technology. Also, as indicated in Section 1 the Replay Portal architecture avoids the blanket broadcasting of content across access networks which is not possible with regular consumer electronic devices, which only deals with the video signal once it has already been delivered to the home. Having a network based service also implies a managed service freeing users from the chore of managing their own content unless they so wish. Finally, in the Replay Portal architecture a user is not limited in the number of simultaneous recordings that can be performed by tuner limitations in a home device. Since all storing of content happens inside the network, on shared service provider infrastructure, a user might be simultaneously recording multiple streams (at the portal) while watching any one of these (or indeed any other stream)

live. Prior work on network-based services for processing TV content includes Agora [Hyden and Sreenan, 1996] from Bell Labs. Agora allows users to have personalized access to TV newsfeeds, its main focus being techniques for efficient content extraction and event notification.

Addressing some of the same issues from a different angle are the activities of the Advanced Television Enhancement Forum [Advanced Television Enhancement Forum, 2000]. This type of interactive television aims to add HTML data as overlay information on TV signals. This approach does not change the fundamental broadcasting-everything model and is therefore unlikely to succeed in an environment where users are demanding personalized services.

Another important area of related work is Internet based content distribution. The replay portal architecture presented in this paper will be a value added service to a "basic" streaming content distribution network. Our architecture will make use of a content distribution network in order to get content to portals and to exchange content between portals and will then add the replay and related functions in a service offering. Current product and service offerings in this space mainly cater to Web traffic but support for streaming content is becoming available from both the vendor and research communities [Francis, 1999, SightPath, 2000, RealNetworks, 2000b, Fast Forward Networks, 2000]. One part of the problem solved by these offerings resolves around on-demand streaming of fairly short (low quality) clips where the objective and solution is very similar to that of Web content, i.e. to get content closer to users and to make intelligent choices as to what server will serve a particular request. The problem is generally addressed by creating an overlay network of cooperating content distribution servers which interact with each other and the actual content servers to offer load balancing, redundancy and reduced latency. In the domain of live streaming content the overlay network can also provide efficient application level distribution trees between the content distribution servers and offer retransmission facilities in the overlay network to compensate for the lack of such mechanisms in streaming protocols. Indeed one of the major problems with current streaming offerings [RealNetworks, 2000a, Microsoft, 2000] is the lack of standard protocols on which to transfer streaming content. This means that content distribution server vendors are required to support a number of proprietary protocols in order to realize their goals thus increasing the price and complexity of their products. More seriously though is the fact that these proprietary protocols are not subjected to the same amount of scrutiny TCP has undergone and its impact on the stability of the Internet is therefore unknown.

The final substantial area of related work is that of video-on-demand (VOD). The work presented here is not VOD in the "traditional" sense, where video content is somehow uploaded to a server and then made available for on-demand viewing. Rather in our architecture, live schedule-based content is made available for on-demand viewing as soon as it has been "aired". Nonetheless, as soon as content is viewed on-demand, we expect that many of the techniques and methods developed for VOD will be applicable in our architecture. For example, access to popular content might well benefit from batching [Dan et al., 1994] or patching [Sen et al., 1999] techniques. Batching involve slightly delaying a particular request for content in the hope that other requests for the same content will arrive soon so that all requests can be served with a single response and content delivery. Patching on the other hand tries to exploit the buffering capabilities of endpoints by allowing a client to receive (and buffer) part of a clip from an existing stream, and the server then only has to send the missing initial part of the stream.

4. CONCLUSION

We presented a hybrid IP-based architecture which explores the space between broadcasting and personalized on-demand access to streaming media. Our solution maintains the current schedule driven approach of present day TV, while making previously "aired" content available for on-demand viewing in a variety of ways. The architecture presents an attractive means for service providers to gradually introduce a variety of services, on a common IP transport infrastructure, which enables the possibility of rich interaction between different packet based service offerings.

References

[Advanced Television Enhancement Forum, 2000] Advanced Television Enhancement Forum (2000). `http://www.atvef.com`.

[Apache Software Foundation, 2000] Apache Software Foundation (2000). `http://www.apache.org`.

[Basso et al., 1999] Basso, A., Cash, G., and Civanlar, M. (1999). Implementation of a real-time MPEG-2 delivery system based on RTP. Packet Video 99, NY.

[Dan et al., 1994] Dan, A., Sitaram, D., and Shahabuddin, P. (1994). Scheduling policies for an on-demand video server with batching. Proceedings of the second ACM international conference on Multimedia.

[Eldering et al., 1999] Eldering, C. A., Sylla, M. L., and Eisenach, J. A. (1999). Is There a Moore's Law for Bandwidth ? *IEEE Communications Magazine*, 37(10):117–121.

[Fast Forward Networks, 2000] Fast Forward Networks (2000). `http://www.ffnet.com`.

[Fielding et al., 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. RFC 2616.

[Francis, 1999] Francis, P. (1999). Yallcast: Extending the Internet Multicast Architecture. `http://www.yallcast.com`.

[Hyden and Sreenan, 1996] Hyden, E. and Sreenan, C. J. (1996). Agora - a personalized digital newsfeed. NOSSDAV.

[Microsoft, 2000] Microsoft (2000). `http://www.microsoft.com`.

[RealNetworks, 1999] RealNetworks (1999). RTSP: Reference Implementation. `http://www.real.com/devzone/library/fireprot/rtsp/index.html`.

[RealNetworks, 2000a] RealNetworks (2000a). `http://www.real.com`.

[RealNetworks, 2000b] RealNetworks (2000b). Real Broadcast Network. `http://www.rbn.com`.

[ReplayTV, 2000] ReplayTV (2000). `http://www.replaytv.com`.

[Schulzrinne et al., 1996] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. (1996). RTP: A Transport Protocol for Real-Time Applications. RFC 1889.

[Schulzrinne et al., 1998] Schulzrinne, H., Rao, A., and Lanphier, R. (1998). Real time streaming protocol (rtsp). RFC 2326.

[Sen et al., 1999] Sen, S., Gao, L., Rexford, J., and Towsley, D. (1999). Optimal patching schemes for efficient multimedia streaming. NOSSDAV'99, Basking Ridge, NJ. Available form:`http://www.nossdav.org`.

[SightPath, 2000] SightPath (2000). `http://www.sightpath.com`.

[Sigma Designs, 2000] Sigma Designs (2000). `http://www.sigmadesigns.com/`.

[TiVo, 2000] TiVo (2000). `http://www.tivo.com`.

[van der Merwe et al., 1999] van der Merwe, J., Chu, Y.-H., Caceres, R., and Sreenan, C. (1999). mmdump: A tool for monitoring internet multimedia traffic. AT&T TR 00.2.1, available from `http:\\www.research.att.com/resources/trs`.

[Vision Tech, 2000] Vision Tech (2000). `http://www.visiontech-dml.com/`.