1 Introduction

The growing desire for multimedia communication and the popularity of the Internet have sparked a great deal of research into the delivery of time-sensitive data streams over IP-based networks. Technologies such as RealAudio [1] attempt to provide robust presentation of multimedia documents, while the MBone and multicast IP enable multimedia conferences where live audio and video are shared amongst the participants [2].

Tools that support conferencing are typically designed to deliver only live media. This works well for discussion sessions but is problematic for applications such as distance education. A lecturer may wish to use supplementary material, such as audio or video clips, to enhance his or her lecture. In this paper we present an application which includes extensions to the common MBone tools vic and vat to allow the inclusion of pre-recorded media streams in a MBone conference. These tools are integrated into the multimedia presentational technologies developed by the Canadian Institute for Telecommunications Research, which support the retrieval of documents stored in a multimedia database and quality of service negotiation [3].

A salient feature of our design is that extended software is not required at the machine of every user who wishes to take part in a distance education session. For student machines, off-the-shelf versions of vic and vat are used to transmit live audio and to receive audio and video from anyone in the conference. Only the lecturer machine requires the extended software because of the need to transmit pre-recorded documents.

This paper is organised as follows. Section 2 describes our distance education application and its requirements. Section 3 discusses sources of time-sensitive data streams and examines some of the issues inherent in their delivery. Section 4 provides details on how the various modules within our presentational system are integrated with the MBone tools. Section 5 presents some preliminary observations about the complete system and Section 6 contains our conclusions and a discussion of some open issues.

2 Application Architecture and Requirements

The distance education application under consideration involves a single lecturer and a group of students. Students are able to see and hear the lecturer, and are able to transmit audio as allowed by the lecturer. Students at any location may take part in the lecture provided that they have the necessary network connectivity and the lecturer's permission.

Lectures are often more valuable and interesting if they are augmented by supplementary information, which may take the form of pictures, audio recordings, video clips, and so on. In essence, any multimedia document may be used to enhance the lecture. To this end the system should support, alongside the conferencing facilities, a document storage and retrieval system. This system can act as a library. Facilities should be provided to search for and retrieve documents and to view them on a user workstation. Facilities should also be provided to allow a lecturer to include the contents of selected documents in his or her broadcast.
In summary, the users of the system may be grouped into two classes:

- **Lecturer**: The lecturer is the main focus of the discussion, having full audio and video broadcast capability to the students. The lecturer's machine is capable of network communication, video and audio capture, video and audio display, and search/query operations. The lecturer is responsible for session control.

- **Student**: The student's workstation has requirements similar to those of the lecturer's machine with the exception of video capture. Audio capture is the only multimedia input method required.

### 2.1 User-Level Capabilities

We now examine the users' requirements in more detail. The capabilities available to the lecturer and students include the following:

- The lecturer may broadcast the video recorded by a camera and the audio received by a microphone to the students. The camera will typically be trained on the lecturer's face and the microphone will be used to record his or her voice. In this application, visual aids such as overheads will be supported by capturing them with the camera.

- The lecturer may retrieve a document from the multimedia database. This includes performing a search for the document, viewing descriptive information about the document such as title and author, and so on.

- A retrieved document may be presented locally, so that it is only seen on the lecturer's machine. The lecturer may also broadcast the document to the students. Such a broadcast will replace the audio and/or video of the lecturer.

- When a student submits a request to speak, his or her name is presented to the lecturer. The lecturer may give any student permission to speak; that student's speech will replace the lecturer's audio and will be broadcast to the other participants. The lecturer's video signal will continue to be broadcast. Once the student has finished speaking the lecturer may regain control of the audio channel and may respond appropriately. In addition to asking questions the student may request the instructor to broadcast a multimedia document.

### 2.2 Hardware Requirements

The configuration of a student machine depends on what capabilities are desired. At a minimum the machine must be capable of supporting the core conferencing tools on which the system is built. Many such tools split the audio and video components of conferencing into separate programs. If such tools are used, the student machine may be configured to receive the audio or video component as desired. The machine must also have network connectivity, operating system support for multicast IP, and may have a hardware video decoding device suitable for enhancing the performance of the video conferencing tool. These requirements are quite flexible, allowing student machines to be custom-configured according to need, cost constraints, and resource availability.

The main task of the lecturer's machine is to prepare the lecture contents for broadcast to the student machines. This consists of delivering the currently relevant audio and video (either from the lecturer's camera/microphone or those representing stored documents) in a format appropriate to the conferencing tools.

### 2.3 Distribution of Capabilities

Our goal is to minimise the complexity of hardware required and to reduce the amount of specialised software and administration effort needed to operate the system. To achieve this goal we place the bulk of the system's complexity on the lecturer's machine. There are two reasons for this:

- **Cost**: In a typical educational setting there are many more student machines than lecturer machines. Although it could be argued that some students are in possession of powerful workstations, any simplifications we can make would translate into lower cost and greater system reliability.

- **Universality**: Minimising the requirements of the student machines also increases the number of machines that will meet these requirements. Using off-the-shelf software and commodity hardware allows individuals to take part in lectures without having to install costly and complex custom components.

![Figure 1: The streams that make up a multimedia conference.](image-url)
These considerations have led to our choice of the vic and vat MBone tools as our core conferencing software. They are widely available, exist for many platforms (including many flavours of Unix and Win32), are portable, have source available, and are commonly used for current Internet-based conferences [2]. After making this decision, the task becomes one of integrating the conferencing tools into our presentational technology.

3 Media Streams

Continuous-media objects, whether live conferences or stored documents, can be thought of as being made up of streams. Each stream represents a logical unit of data that is to be presented over a period of time. For example, consider a conference between two participants where each participant supplies video and audio to the other (see Figure 1). The video capture hardware on each workstation emits a stream of video information; similarly for audio. Thus we have four streams, all originating from “outside” the software architecture — the data that make up the streams are captured from the real world. We will call such streams live streams.

Contrast this situation with that shown in Figure 2. Here we have a simple video-on-demand system where a server provides audio and video data to a group of clients. The audio and video being transmitted are retrieved from storage and transmitted separately, resulting in two streams that are received and processed by the client machine. We will call such streams pre-recorded streams.

The following two sections examine in detail how live streams and pre-recorded streams are generated.

3.1 Live Media Streams

The production of live media streams is a relatively simple process. In the case of video an external video source prepares image frames at regular intervals which are then captured by digitising hardware. The digitising hardware may also support encoding (e.g. motion JPEG, MPEG, etc.). Audio stream production involves a similar approach, whereby individual audio samples are grouped into “frames”. These frames are presented by the digitising hardware as they become available.

An important aspect of these media streams is that an application has no control over their content or production. New data are presented to the application through some capture device on a regular schedule. The application must retrieve and act upon the streams in a timely manner if the flow of data (and thus the appearance of the media presentation to the user) is to be uninterrupted.

3.2 Pre-recorded Streams

Pre-recorded streams are often stored in a specialised file server referred to as a continuous media file server (CMFS). The motivation for designing such a server is well established. Data access patterns for continuous media, as well as the services provided to the clients by such a server, differ considerably from those of a conventional distributed service such as NFS. A CMFS typically transfers large volumes of sequential data, requiring large resource commitments from the system and from the network. To ensure continuous data transfer, the allocation of resources such as disk and network bandwidth, processor cycles, and memory at the server must be guaranteed for the duration of the stream's transmission. Our system is based on the CMFS developed at the University of British Columbia [4,5]. This system has a flexible architecture in the sense that more streams can be supported by adding server components.

3.3 Stream Synchronisation

Recall that our system design is based on the assumption that media streams may be transmitted separately. A mechanism is therefore needed to synchronise these streams for presentation at the user’s machine. The details of our synchronisation algorithm are reported in [6]. This algorithm is robust and does not require the use of a global clock.

4 Conferencing Architecture

We now examine the presentation of media streams to the user and the integration of this presentation with a real-time multimedia conference.

The MBone tools allow capture, processing, and broadcast of live media streams. Both vic and vat provide support for an extensive set of capture and playout hardware; they handle the conversion and delivery of the data received from such hardware. Our work therefore centres on adapting this model to pre-recorded media streams.
4.1 Data Preparation

For pre-recorded documents the media streams are simply retrieved and synchronised. The synchronisation module communicates with the relevant output device on the user's machine, such as a video decoder board or an audio interface.

Transmission of the pre-recorded media over the MBone is not as straightforward because of an important restriction: the data stream must be in a format suitable for consumption by the MBone tools installed on the student machines. There are two aspects to this restriction:

- The media data must be encoded using an appropriate method, and
- The media streams must be packetised and transmitted using protocols compatible with the MBone tools.

Given that our CMFS places no restrictions on data formats the first requirement can be met by encoding the data with the encoding scheme used by the MBone tools. As our test system contains Motion JPEG (MJPEG) boards and vic supports a software MJPEG decoder, MJPEG has been selected as the format for our implementation. Similarly, both vat and our audio hardware are capable of handling 8-bit PCM-encoded audio at 8000 samples per second, making this a suitable format for audio streams.

The second requirement is easily met by the modularity of the MBone tools. They already contain the necessary processing and packetisation functions; we simply inject our data at the appropriate place.

4.2 Data Delivery

We need to consider the delivery of the data comprising a multimedia document to the MBone tools. Any solution to this problem should take into account the following issues:

- Consistency with local playback: For simplicity we would like to use the same synchronisation system for the MBone tools as we use for playout to a local decoding device. Furthermore, the less knowledge that the system has of the data's destination the better. To the synchronisation module, playout to the MBone tools should look identical to playout to local display hardware.
- Media independence: The programming interface presented to the synchronisation module should hide the fact that different media types use different MBone tools (vic for video, vat for audio, and so on.)
- Expedient delivery: As frames are made available by the synchronisation module they should be dispatched to their relevant MBone tool quickly and with uniform delay. This will not only preserve the fluidity of each stream but will also aid in synchronisation. The MBone tools responsible for different media types do not communicate with each other, and their architecture does not support synchronisation of simultaneous streams; they rely on best-effort synchronisation. If we deliver data streams to the tools in synchrony we maximise the probability of synchronised playout at the student machines.

We achieve the above requirements by keeping the synchronisation system in its own process, using shared memory to transfer the media data to the processes that house the MBone tools. This has two benefits:

- The isolation afforded by different processes minimises the interaction between the real-time requirements of the synchronisation module and the event-driven process-blocking user interfaces employed by vic and vat.
- Transferring the bulk of the data through shared memory eliminates the costly data copying that might otherwise be required. Control information on the order of a few bytes per frame is copied, however, acting as a means to monitor access to the shared memory buffers.

![Figure 3: The data path for media streams.](image-url)
A schematic showing the flow of data through this architecture is shown in Figure 3.

"vic" and "vat" are designed such that frames are “pushed” by source objects as they are generated. The frame rate (and therefore the bit rate of the data stream) is determined by the input objects themselves. An input object can determine how often it should present a new frame using the data-rate and frame-rate control information supplied by the tool’s user interface. This scheme adapts well to our synchronisation module, which will not necessarily produce frames at regular intervals if the video stream must be delayed to match the audio.

### 4.3 API Structure

The API is designed to handle streams individually. It is the job of the API to move, over time, the data comprising each stream from a source application to destination applications that are stream-specific. The data are divided into segments that represent frames of media. The amount of time represented by each frame is defined by the user of the API. One or more buffers are created and shared between the source and destination applications, forming a circular queue into which frames may be deposited. The number of buffers is limited by the maximum number of per-process shared memory segments supported by the operating system. For example, under AIX, the limit is 10.

Once the buffers are created they are mapped to shared memory segments. The application refers to the buffers using a simple index, while the API internally refers to them by their shared memory ID or `shm id`. This ID is unique across any given system. It can therefore be used by a destination application as a handle to uniquely identify a buffer. When the synchronisation module wishes to display a frame it passes the index of the buffer containing the frame to the API. The API then sends the `shm id` of this buffer to the destination process via a Unix FIFO. This triggers the MBone tools to begin processing the frame. Note that the `shm id` corresponding to the new frame is the only data copied between the address spaces of the two processes. This is a small value (typically 32 bits) and is therefore inexpensive to transport. The message exchange involved in transmitting a stream is shown in Figure 4.

### 4.4 Tool Specifics

As discussed in [7], "vic"s architecture is designed with device-independence in mind. Video data from an external source enters through a combination of an InputDevice object and an associated Grabber object. The purpose of these objects is to control the video capture hardware and to manage the grabbing of single frames from the video input stream, respectively. The Grabber object is connected to a consumer that manages the delivery of video data to "vic"s network fabric. "vic" contains an extensive array of these consumers, allowing Grabber objects to supply raw video data, MJPEG-encoded video, and so on.

The lecturer’s machine runs a modified copy of "vic" that includes, in addition to the objects suitable for its video capture hardware, objects which retrieve frames from the shared memory buffers when instructed to by the synchronisation module. If pre-recorded video is selected (by choosing the appropriate input device from "vic"s menu) the Grabber object opens the Unix FIFO and reads the initialisation information placed there by the API. After making the shared memory areas available in its address space it waits for activity on the FIFO that will signal the readiness of a frame. Once a frame arrives the address of the buffer containing the frame is passed to the rest of "vic"s control fabric. This frame is packetised, processed, and transmitted; as far as the rest of "vic" is concerned the frame looks the same as if it had come from a hardware encoder.

Although the detailed object structure differs somewhat, we use a similar approach for "vat". "vat" supports audio devices with multiple input ports, intended for hardware with connections for line- and microphone-level inputs. Our input object adds another input port corresponding to pre-recorded audio. When the user selects this input port the object goes through the same procedure as outlined above for "vic": the `shm ids` are retrieved and "vat" waits for delivery of the first frame.
5 Results

We have completed a proof-of-concept implementation and can conclude that our approach to conferencing is viable. Our system consists of an IBM RS/6000 acting as the CMFS, another RS/6000 providing lecturer services, and two PCs and a Sun SPARCstation 10 acting as student machines. These machines are interconnected by a local 10 MBit/s Ethernet. An ATM switch also connects the RS/6000s.

Our experiments are concerned with a multicast conference. Live MJPEG video is captured at 5 frames per second and is broadcast to the student machines where the video is decoded in software. Audio consists of 8-bit PCM at 8000 samples per second. The lecturer may retrieve a document on demand from the CMFS and broadcast it to the student machines. We have tested the broadcast with MJPEG video stored at 15 frames per second and 320 x 240 resolution; this stream requires approximately 2 MBit/s of network bandwidth. The student machines can receive and present this video and audio with very few noticeable artefacts. Further tests were performed using 30 frames per second MJPEG video, but the Ethernet did not have sufficient bandwidth for their transmission to other machines. The lecturer's copy of vic, however, displayed the data without incident.

Further testing is planned, beginning with off-site student machines. This requires high-bandwidth wide area connectivity, which has been installed recently. As low-cost hardware MPEG encoders become available the system can also be tested with higher frame-rate live video.

6 Conclusions and Open Issues

The addition of pre-recorded content to a multimedia conference greatly expands the utility of such a conference. Applications such as distance education become viable, as do other applications requiring consultation and reference to stored material. The melding of presentational and conferencing systems will increase the utility of the technology, resulting in greater user interest and more opportunities for improvement.

Further research in several areas is required before conferencing systems over an IP-based network can be deployed on a large scale. Quality-of-service is an important and difficult issue. The system we have presented here involves the delivery of a document to many clients — each may have vastly different capabilities. For example, it is not obvious how to deliver a document to two student machines, one a fast workstation containing video decoding hardware and the other a small portable machine with low CPU power and little memory. Similar problems present themselves when one considers buffer allocation, network use, and so on.

Synchronisation is also very difficult in a multicasting environment. The MBone tools we have used do not support any form of synchronisation amongst parallel media streams. If every recipient is to be guaranteed a polished presentation a distributed synchronisation algorithm must be developed that can adapt to varying synchronisation requirements amongst the clients.

Further performance analysis of the application should also be carried out. Our experiments so far have shown that 15 frames per second video may be delivered over a moderately loaded Ethernet and that 30 frames per second video may be successfully prepared for broadcast. However the exact performance of the transmission tools under heavy load is not well understood nor has the API been extensively tested for scalability. Both of these will have to be done if the application is to be used in a production environment.

7 Acknowledgements

This research was supported by a grant from the Canadian Institute for Telecommunications Research under the NCE program of the Government of Canada.

References