Best-effort Support for a Virtual Seminar Room

Robin Sharp and Edward Todirica
Informatics and Mathematical Modelling
Technical University of Denmark
DK-2800 Kgs. Lyngby, Denmark.
{robin,edward}@imm.dtu.dk

ABSTRACT

This paper describes the RTMM Virtual Seminar Room, an interactive distributed multimedia application based on a platform with a simple middleware architecture, using best effort scheduling and a best effort network service. Emphasis has been placed on achieving low latency in all parts of the software system, so that as large a margin as possible is available for the transfer of data through the network. This approach gives good user acceptability for the transfer of audio and video over distances of several hundred kilometers within the high-bandwidth Danish Research Network. The design of central parts of the system is presented, and the performance offered by this approach is discussed.

Categories and Subject Descriptors

H.4.3 [**Information Systems Applications**]: Communications Applications—*Teleconferencing*; C.2.4 [**Computer Communication Networks**]: Distributed Sytems; D.4.8 [**Operating Systems**]: Performance

General Terms

Multimedia

Keywords

Multimedia, Virtual Seminar Room, Best-effort service

1. INTRODUCTION

At the Technical University of Denmark, work is in progress to develop a Virtual Seminar Room (VSR) as an example of an interactive distributed multimedia (DMM) system. This development work is part of the RTMM (Real-Time Multi Modal) project, a collaborative effort involving the Informatics and Mathematical Modelling department and the COM Center at the university. The VSR is intended for use in teaching situations, where it should offer participants facilities to see and hear one another and to have access to various pieces of shared virtual equipment, such as a virtual whiteboard, a virtual slide projector and virtual demonstrations. The layout of a typical site in the RTMM VSR is shown in Figure 1. One or more participants sit in front of a camera, screen, microphone (or microphone array, as in the figure) and a set of loudspeakers, with access to a writing tablet and keyboard, which are used for writing on the virtual whiteboard and controlling the running of the teaching session. To increase user acceptance, and promote natural reactions among the participants at each site, it has been our aim that each site should be set up in an ordinary teaching room, with ordinary furniture and a platform based on a standard PC, and that the only visible special equipment should be the large screen, the camera, and the loudspeakers and microphones needed for achieving the required quality for the video and audio.

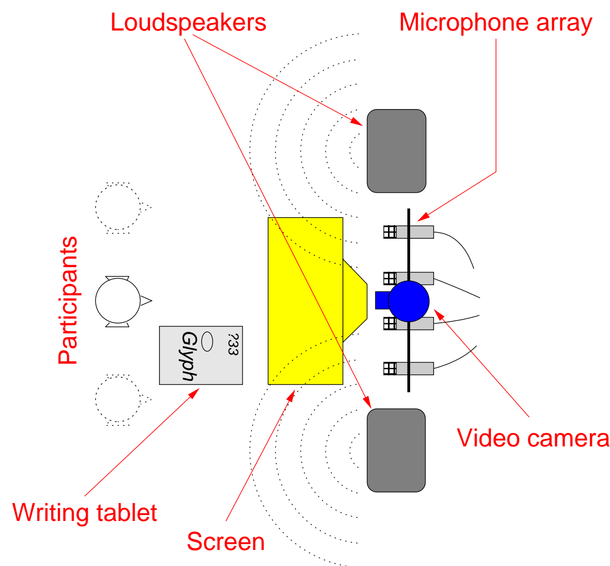

Figure 1: Layout of a typical site of the RTMM Virtual Seminar Room

MULTIMEDIA '02 Juan les Pins, France

Implementation of systems of this type involves a number of technical challenges, as it is necessary to ensure that data representing video pictures, still images, sound and other information are passed between the sites taking part in the seminar, and are presented in real time to the users at these sites, to give them the illusion that they are taking part in a discussion in the same room. To do this, it is necessary

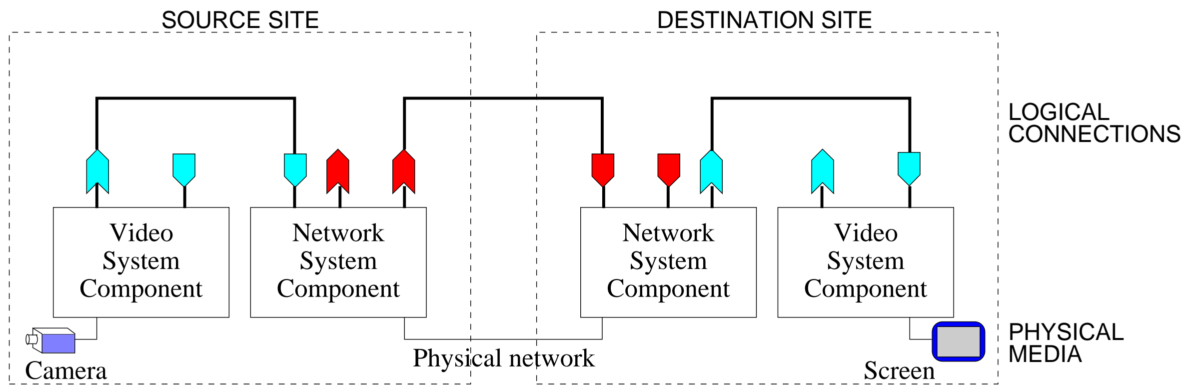

Figure 2: A system with two sites and four system components.

to have an efficient system for capturing, distributing and replaying high quality video, audio and other data in real time, in order to meet the necessary synchronisation requirements [8]. One of the aims of the RTMM project has been to investigate the extent to which this target can be achieved on the basis of a standard platform with a standard operating system and best-effort network connections. This paper discusses how this aim has been achieved.

2. THE STREAM LAYER ABSTRACTION

The approach to system construction taken in the RTMM project is to provide the DMM application implementor with middleware designed as a software toolbox based on a number of simple concepts. The toolbox offers facilities for connecting active entities, known as *system components*, via logical channels through which *streams* of data with various quality of service requirements can be passed. Typically, a system component is composed of a hardware device and adaptor card with its driver, but pure software components, such as stream transformers, also occur. The facilities are offered at the interface to a conceptual *Stream Layer*, which is the basis on which all applications are built. For portability, the Stream Layer software is implemented as a GNU C++ library, and the system runs on a standard Linux/PC platform.

The Stream Layer offers its users facilities for defining system components, for specifying how they are logically connected and for specifying which types of streams of data pass through these connections. Figure 2 shows a simple example of a system consisting of two physically separated sites, each containing two system components: a video system component and a network system component. A *stream* is a logically related sequence of data units which pass between two system components, starting at the *source* of the stream and finishing at its *destination*.

A *socket*, through which the stream passes, is used to identify the particular starting point of a stream in the source system component and the particular ending point in the destination system component. RTMM sockets permit unidirectional flow of data. The socket in the source from which the stream starts is denoted an *OutSocket* and the socket in the destination at which the stream ends an *InSocket*. For illustrative purposes, these are designated by graphical symbols as shown in Figure 3.

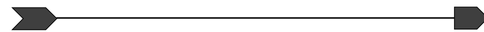

Figure 3: A connection from an OutSocket (left) to an InSocket (right)

To pass a stream between system components, a *connection* must be set up between these components. When initially created, a connection connects two sockets: an *OutSocket* and an *InSocket*, which may be associated with the same or different system components. By setting up a connection to one or more further *InSockets*, an *OutSocket* may become the source for a stream with multiple destinations, as illustrated in Figure 4. On the other hand, it is not possible for

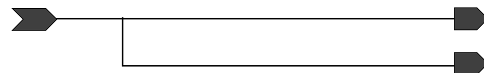

Figure 4: Connections for passing a stream to multiple InSockets

the same *InSocket* to be the destination for several streams. When initially created, an *InSocket* is *closed*; it must be put into the *open* state in order to pass data. Data arriving at the *InSocket* when it is closed will just be ignored.

A connection passing between two sockets offers a certain *Quality of Service (QoS)* to the stream which it carries. The parameters of the QoS include the bandwidth, delay, jitter and other properties of the connection, and are expected to be compatible with the QoS required for the stream.

A more complete description of the Stream Layer API is given in [7].

3. SYSTEM COMPONENTS

Conceptually, each system component can have one or more InSockets through which incoming streams can be received by the component, and one or more OutSockets through

which outgoing streams can be sent out of the component. An incoming stream can be *consumed* by the component, in which case the data in the stream are typically directed to some ouput device associated with the component. Similarly, a component may be used to *generate* an outgoing stream, whose data typically originate from some input device. Finally, a component may pass an incoming stream on as an outgoing stream, possibly after *transforming* it into a stream with new properties. The Stream Layer API offers functions for associating generator (source) functions, consumer (sink) functions and transformer functions with the component, and for setting up internal connections within the component in order to pass incoming streams at InSockets to OutSockets via which the streams can leave the component.

A typical RTMM DMM system is composed of several different types of system component. Most of these are directly associated with particular hardware adapter cards based on specialised auxiliary processors that are exploited to reduce the load on the CPU. Examples used within the implementation of the VSR set up at DTU are:

Audio A: Based on a standard stereo sound card for encoding and decoding audio signals.

Audio B: Based on the Bittware Spinner ADSP-21065L dual-DSP board [1], which is used for executing more advanced signal processing algorithms for source location, beam forming and echo cancellation, using a 4-microphone array as input source and a two channel audio setup for output [3]. The positional information about the source of the sound provided by this component enables us to control the orientation of the video camera, so as to follow the current speaker.

Video A: Based on the Matrox Marvel G400 graphics card [5] for MJPEG encoding and a standard, freely available software library (IJG JPEG [4]) for MJPEG decoding of video in SIF/CIF format. The received video can be displayed either in a separate window or in full-screen mode, using functions from the SDL library.

Video B: Based on the Equator MAP-CA video DSP platform [2] for more advanced video processing, including MJPEG encoding and decoding in full PAL resolution and MPEG en- and decoding.

Whiteboard: Based on the Wacom graphics tablet as input device.

UDP Network: Based on a standard network card and UDP/IP protocol stack.

As can be seen, some of the components are implemented in several versions, which can be selected according to the desired quality of the relevant stream and/or the target price of the complete system.

Figure 5 shows an example of the code needed to set up a small application involving a video component, an audio component and a network component on each site. Only the code related to the video streams is shown; the code for the audio streams is similar. The corresponding system structure is shown in Figure 7. Note that the threads, marked t1,t2, . . . ,t12 in the figure, are not explicitly activated by the code shown above, but are activated within the components concerned.

```
startStreamLayer();
//Declare components
VideoComp C1;           //Video component
AudioComp C2;           //Audio component
NetComp   C3;           //Network component

//Create sockets for the components.
//Socket id: first  number - component no.
//          second number - stream no.
//Initial "s": local socket, "n": network socket
OutSocket*    s11 = C1.createOutSocket("s11");
InSocket*     s12 = C1.createInSocket ("s12");
InSocket*     s31 = C3.createInSocket ("s31");
OutSocket*    s32 = C3.createOutSocket("s32");
OutNetSocket* n31 =
    C3.createListenSocket("Stream1","n31");
//Set up socket to receive on agreed local port
InNetSocket*  n32 = C3.createReceiver("45678");

//Create connections between components.
//Connection id: first  number - source comp.
//                second number - dest.  comp.
Connection *c13,*c31;
try{ c13= new Connection("Stream1",s31,s11);
     c31= new Connection("Stream2",s12,s32);
} catch (Connection_error e){e.showError();};

//Create internal connections in C3.
C3.connectInToOut("",0,0,s31,n31);
C3.connectInToOut("",0,0,n32,s32);

//Create network connection to remote port
n31->connect("foo.bar.com 45678");

//Open connections to allow stream to flow
c13->openConnState();
c31->openConnState();

//Associate source + sink functions with sockets
try{ C1.source(1,"Stream1",0,s11);
     C1.sink(1,"Stream2",0,s12);
}catch (Component_error e){e.showError();}

//Activate the generated stream
try{C1.activateStream("Stream1");
}catch (Component_error e){e.showError();}
  ...
  ...
endStreamLayer();
```

Figure 5: Code for setting up a small application using the RTMM toolbox

4. PERFORMANCE

A series of timing measurements made on the system are summarised in Table 1. These timings were observed over a two-way link between DTU and Aarhus in Denmark, a dis-

	Min.	Max.	Avg.	Std.dev.
Stream layer	0.04	0.13	0.05	0.03
Video decoding	15.6	29.0	19.5	1.57
Video presentation	11.2	27.2	13.1	1.50
Network transmission	8.9	20.1	10.5	0.96
End-to-end video	41.3	58.7	44.4	2.33

Table 1: Video processing times in the system (milliseconds)

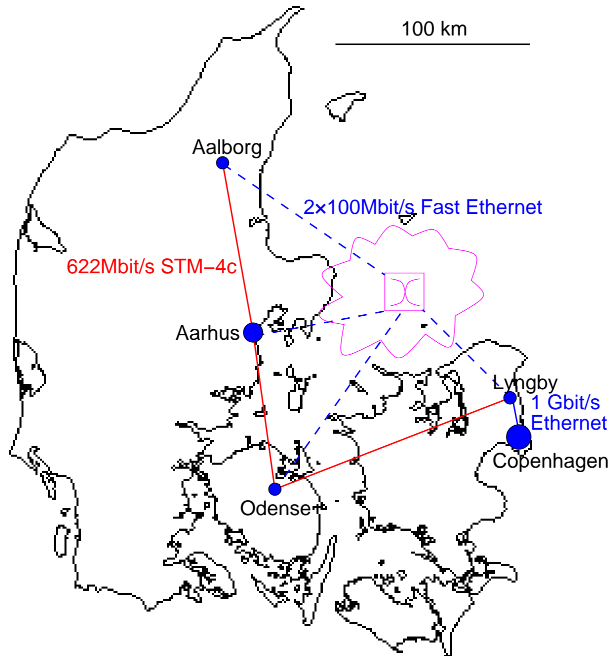

Figure 6: The National Backbone of the Danish Research Network

tance of about 400 km., using UDP/IP over the Danish Research Network. The National Backbone of this network is illustrated diagramatically in Figure 6. DTU is situated on the same campus as the main network node in Lyngby, which is connected to the node in Aarhus both by a 622 Mbit/s STM-4c link passing though Odense, and an IP network based on double 100 Mbit/s Fast Ethernet links.

The computers at both ends are based on a 933 MHz Pentium III CPU with 256 Mbytes of main memory, runnning RedHat Linux version 6.2 with the Linux 2.3.99 kernel. For the purposes of the measurement, the RTMM application running in Aarhus was set up to act as a reflector which returned all the received audio and video data to the Lyngby site. In all the measurements, the Video A video system component, operating in 352x288 pixel format at 25 frames/s, and the Audio A audio system component were used. The minimum, maximum and average bandwidth requirements for video were 3.15, 3.50 and 3.30 Mbit/s, with a standard deviation of 0.04 Mbit/s. For audio, the bandwidth was a constant 670 kbit/s for uncompressed mono CD quality. All measurements refer to a period of about 1 minute (1500 frames). For network transmission, one IP packet was used to contain each JPEG encoded video frame (about 16.5 kbytes) or a mono audio packet of about 1 kbyte, so no IP segmentation was required.

The times referred to in the table are measured as follows:

- Stream layer: On the source site, the time from the start of transmission of a frame from the source video system component until the end of receipt by the source network system component. This is a measure of the delays introduced by the Stream Layer software.
- Video decoding: In the destination system component, the time from the instant when the frame is completely received by the component until the software decoding algorithm is completed.
- Video presentation: In the destination system component, the time from the instant when the frame has been received and decoded until the SDL library function returns after storing the frame in the video frame buffer.
- Network transmission: The time from the start of transmission of a frame from the source network system component until the entire frame has been received by the destination network component. (This is measured as half the time for sending the frame to the remote site and back again, where the remote site acts as a reflector.)
- End-to-end video: Measured as the time from the instant when the frame is completely captured until the presentation function returns, where the video frames are sent to the remote site and back again, with the remote system functioning as a reflector. The network transmission time is here halved, so the value in the table is a measure of the true end-to-end delay for video between two sites.

The measurements all refer to the processing of the video stream. For the audio stream, the processing times required are, of course, smaller. All times were measured by recording timestamps captured by using the `gettimeofday` system call, which returns a value with a relative accuracy of 1 μs. To limit the overhead involved in recording the timestamps, they were stored in memory in a buffer dimensioned to hold results from one minute's operation of the system. At the end of the chosen measuring period, the application was stopped and the data were copied into non-volatile storage. Not unexpectedly, the observed times are completely dominated by the video processing and network transmission times. The overhead introduced by the Stream Layer for transfer of data between system components on the same site is very small.

4.1 Performance Analysis

The performance of the system depends strongly on the implementation of the Stream Layer software, and in particular on the way in which the various activities are partitioned on threads. The current implementation associates a thread with each socket. On each site, data from a particular stream are passed from the source system component through one or more connections to a network OutSocket or from a network InSocket through one or more connections to the sink system component. For the system set up by the code of Figure 5, this is illustrated in Figure 7.

Each connection contains a queue with a limited size (default size 10 elements). The thread corresponding to the OutSocket puts data units on the queue, and the thread corresponding to the InSocket gets data units from the queue. The queue operations are blocking, so the OutSocket thread is blocked when the queue is full and the InSocket thread is blocked when the queue is empty. In this way, even if there are many threads in the VSR application, only a few of them are active at any given time; most of them are blocked waiting on empty queues. This means that the scheduler has the

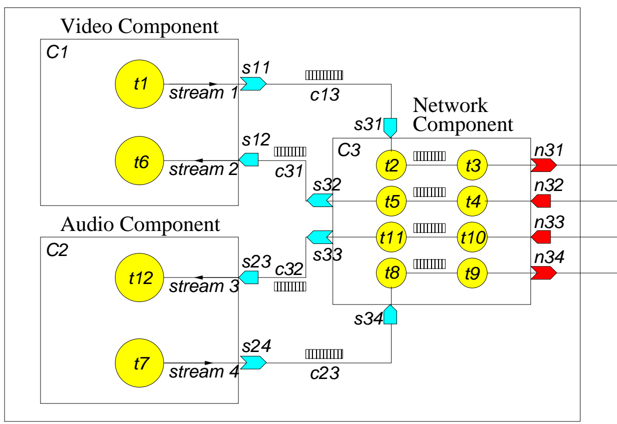

Figure 7: The software architecture of a simple site

Yellow circles symbolise threads and striped rectangular boxes buffers (queues). The component, connection and socket identifiers correspond to the code example given in Figure 5.

relatively simple task of handling only a few ready to run threads. We can say that in the Stream Layer the activity follows the data units: threads become active only if they have a data unit to transfer. Thus the threads corresponding to the Stream Layer have a very short execution time, and most of the time is spent in the threads which correspond to the sinks, where relatively much processing has to take place.

Within the video and audio sink threads, it is possible that, due to delays in the system, buffers corresponding to several frames will be available at the same time. The strategy adopted here is always to present the newest buffer and throw away older ones which are waiting to be dealt with. During periods where the system is busy with other activities, or in cases where some buffers take an exceptionally long time to be processed, frames will be lost.

The behaviour of the system was investigated in detail by running the Linux Trace Toolkit (LTT) utility [6] together with the VSR application. LTT makes it possible, amongst other things, to record the times at which selected system events, such as interrupts and context switches between threads, take place. Some typical results from this investigation are shown graphically in Figures 8 and 9, both of which illustrate the activity of the system as a function of time. In each of the figures, the red curve shows the active thread at consecutive instants of time, and the green crosses mark the instants at which interrupts from the various peripheral units took place. The labels on the vertical axis of each plot identify the activities, where the thread names correspond to the names used in Figure 7, and the names of the interrupts are derived from the devices concerned in the obvious manner.

Close inspection of the tracing data reveals a number of characteristic patterns of activity. For example, with reference to Figure 8:

1. Samples are generated by the audio source and are processed on average every 10 ms., in the figure at times approximately 567000, 580000, 589000, 601000, 610000, 622000 and 632000. A corresponding pattern is seen in the threads which process incoming audio and deliver it to the audio sink, at times 569000, 581000, 592000, ...
2. Samples are generated by the video source and are processed on average every 40 ms., in the figure at times 560000 and 600000. A corresponding pattern is seen in the threads which process incoming video and deliver it to the video sink, starting at times 581000 and 621000.
3. The video sink thread (t6) uses a large amount of CPU time to decode and present each incoming frame. Each period during which the thread processes an incoming frame ends with a very short period of activity (at times 574000 and 613500 in the figure), when the thread releases resources.
4. Network activity, indicated by the presence of network interrupts, shows regular heavy bursts (in the figure around 578000 and 620000). Each of these bursts is due to the arrival of a group of Ethernet packets carrying an encoded video frame. (In the setup considered here, 11 Ethernet packets are needed to carry a coded frame of about 16.5 kbytes.) Similarly, pairs of network interrupts appear at regular intervals, corresponding to the arrival of a group of two Ethernet packets carrying an encoded audio frame.
5. "Other processes" use about 10% of the CPU time. Most of this is activity within the X server.

The figures clearly reveal the systematic manner in which the consecutive threads involved in processing the video and audio streams are activated. It is also clearly seen that the video sink thread is the thread which occupies most of the available CPU time, in agreement with the timestamp-based measurements given in Table 1. LTT itself introduces a small overhead into the system, partly due to the execution of the thread marked LTT, and partly as a (not explictly visible) overhead to the running of the kernel. This overhead is typically estimated to account for at most 2.5% of the available CPU time [9], and is thus expected to cause very little distortion in the behaviour of the system.

Neither the Linux OS nor the UDP/IP network offer QoS guarantees, so the resources available for the VSR application may be strongly influenced by other activities in the system. In such a best-effort system, it is obviously important not to have too many competing tasks running together with the VSR application. In particular, the software video decoding technique is very CPU intensive and competition with other high-CPU activities can cause loss of frames due to failure to complete decoding before the arrival of the next frame. However, it is, as we see, in practice possible to achieve good results, with low ($\sim$40 ms.) end-to-end latency, full 25 frames/s frame rate and low ($< 1 \times 10^{-3}$) rate of frame loss, in a system with a moderate load. The traces and statistical measurements reported here have been made in a system running version 4.0.2 of the XFree86 X server, the KDE2 window manager and two open console windows, in addition to the RTMM VSR application and the tracing tool. Apart from the tracing tool, this is probably typical of

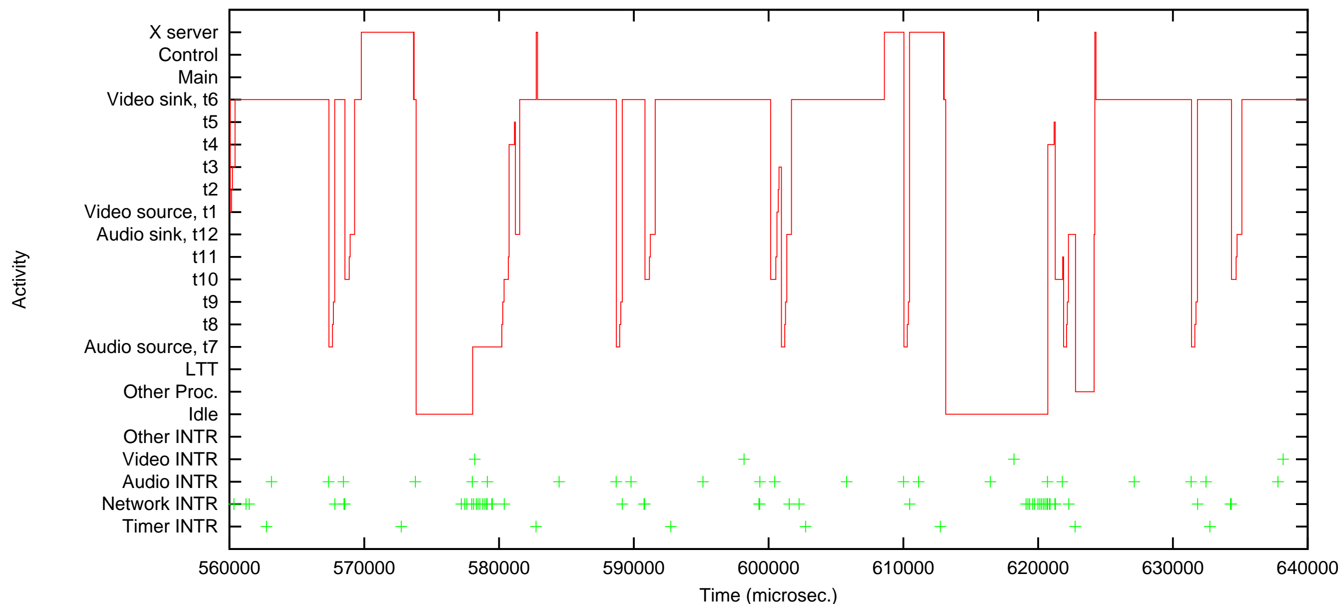

Figure 8: Activity in the system during an 80 msec. interval in which two video frames were processed

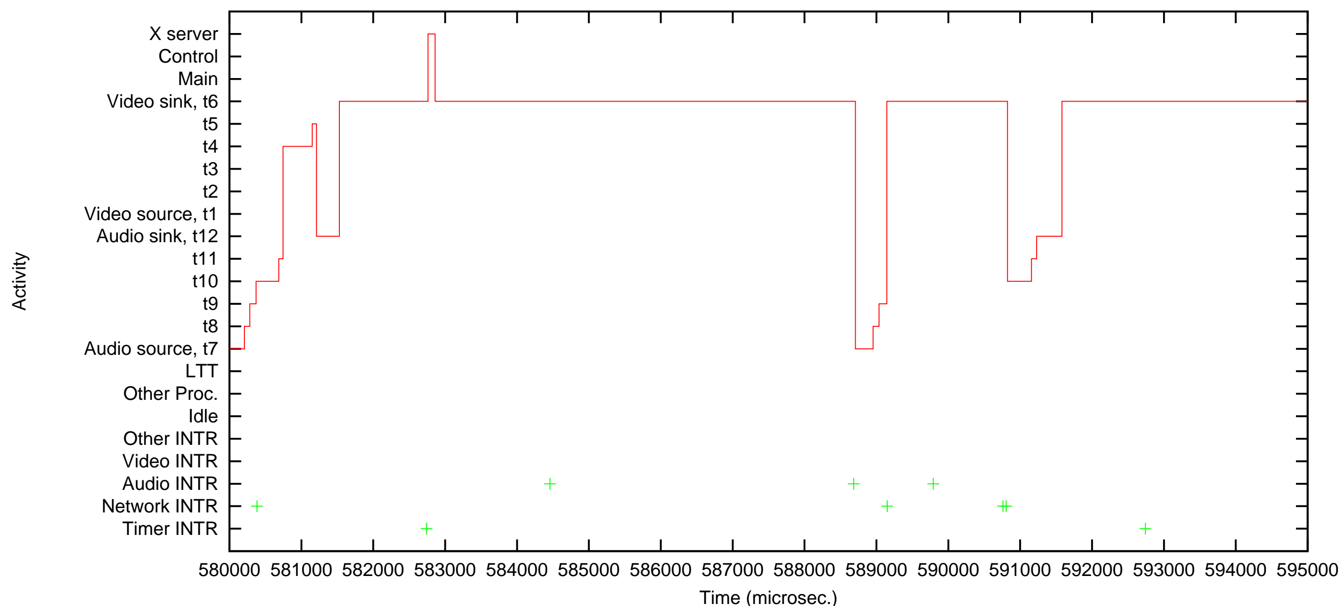

Figure 9: A close-up of activity during a smaller time interval in Figure 8

the kind of activity which will need to be supported while the VSR is in operation in a genuine teaching situation.

5. USER EXPERIENCES

Figure 10 illustrates the type of display experienced in a simple case where two users communicate over the Lyngby-Aarhus link using the full-screen video display mode. For use when multiple sites are connected, the decoded video from each site can be presented in a separate window. Users of the system express great satisfaction with the quality of the displayed video, which is experienced as being markedly superior to the more traditional videoconference quality offered by systems based on the use of H.323. Factors which contribute to this experience include the inherently higher quality of the MJPEG encoding and the low round-trip time achieved within the system over the distances concerned. The large amount of surplus bandwidth in the Danish Research Network helps to ensure that IP packets can be delivered with moderate jitter and a very low rate of packet loss in the network. Packet re-ordering has not been observed at all. The small rate of frame loss which is observed in the system is, as remarked previously, almost exclusively due to missed display deadlines due to delays in decoding, and is not considered significant (in most cases it is not even noticed) by participants.

Figure 10: A scene in the VSR with just two sites.

The simple audio system, based on a single microphone and a pair of loudspeakers at each site, gives adequate sound quality, but requires a carefully chosen near-field microphone in order to achieve an appropriate ambience in an ordinary room. The more advanced audio system with beam forming and source location, using an array of far-field microphones, is experienced as being extremely effective. The advanced audio system offers the added advantage of being able to control the orientation of a rotatable camera, so that it follows the current speaker. This is extremely useful in cases where there are several participants in the same room, as it enables participants at other sites more exactly to identify who is speaking. Participants consider this a very attractive feature.

6. CONCLUSIONS

As part of a current research program into interactive, distributed multi-modal systems, we have at DTU implemented a prototype Virtual Seminar Room based on an operating system with best-effort scheduling and a network with a best-effort IP service. The VSR is built up on the basis of middleware, the RTMM Stream Layer, which offers a convenient and efficient toolbox for constructing a variety of DMM systems. Measurements on the system confirm that the run-time penalty introduced by using this approach is very small compared to the time required for decoding and network transmission in the system. This is important, in order to ensure that suitably large timing margins are available for the activities which may be affected by network and OS delays introduced by the best-effort service. Observations confirm that the rate of frame loss due to missed deadlines is considerably less than 1×10^{-3}.

Creation of the necessary middleware, the RTMM Stream Layer toolbox, has not involved modification of the Linux operating system kernel. This makes the middleware simple to deploy, as it becomes possible to avoid kernel patches and problems caused by incompatibilities amongst various kernel versions. It also makes the solution readily portable. Although built for Linux, a version for Windows could easily be produced by using the Cygwin environment. For use in the RTMM VSR project, a GUI has been developed which enables the user to set up the system in a simple manner by "plugging together" system components, both locally and remotely.

The measurements reported here have been based on a system which uses MJPEG video encoding over IP, a technique which can be implemented relatively cheaply using standard off-the-shelf components. We intend to continue to experiment with other techniques for video encoding and are currently developing further system components, including a virtual slide projector and an ATM network system component, for use in the VSR and other multimedia applications. Users have expressed satisfaction with the current system with site separations of up to a few hundred kilometers, as described here. We intend shortly to start experiments with systems with much larger extents, to investigate the technical problems which this involves and to gauge the user acceptibility of ssuch systems for practical interactive working.

7. ACKNOWLEDGMENTS

The authors would like to thank the Danish Research Councils and the Danish Center for IT Research, who have partially supported the RTMM project as part of the Center for Multimedia in Denmark. We should also like to thank our former colleague Peter Kirk Hansen, who was largely responsible for the audio side of the system, and our partners at the COM Center at DTU for their excellent collaboration during the project.

8. REFERENCES

[1] Bittware, Inc. Spinner ADSP-21065L Audio OEM Board. WWW document, 2002. Available from URL `http://www.bittware.com/products/PCI/SPNR/-spnr_desc.stm`.

[2] Equator Technologies, Inc. *MAP-CA DSP Datasheet*, June 2001. Document HWR.CA.DS.2001.6.20, available from URL `http://www.equator.com/products/-library/ca_datasheet_010620.pdf`.

[3] S. Forchhammer, A. Fosgerau, P. Hansen, R. Sharp, E. Todirica, and A. Zsigri. Video conferencing for a virtual seminar room. In *Proc. 4th International Conference on Digital Signal Processing and its Applications, Moscow*, pages 382–385, Feb. 2002.

[4] Independent JPEG Group. IJG Files. Web document, 2001. Available from URL `http://www.ijg.org/files/`.

[5] Matrox Graphics, Inc. Marvel G400. Web document, 2002. Available from URL `http://www.matrox.com/-mga/products/tech_info/marv_g400.cfm`.

[6] Opersys, Inc. *Linux Trace Toolkit Reference Manual, version 0.8*, Jan. 2002. Available from URL `http://www.opersys.com/LTT/Help/index.html`.

[7] R. Sharp, H.-H. Løvengreen, and E. Todirica. Streams and Sockets in DTU-RTMM, version 1.4. Technical report, Department of Information Technology, DTU, Sept. 2000.

[8] R. Steinmetz. Human perception of jitter and media synchronization. *IEEE Journal on Selected Areas in Communications*, 14(1):61–72, Jan. 1996.

[9] K. Yaghmour and M. R. Dagenais. Measuring and characterizing system behaviour using kernel-level event logging. In *Proceedings of the 2000 USENIX Annual Technical Conference, San Diego*. USENIX Association, June 2000.