SOME SOFTWARE ISSUES OF A REAL-TIME MULTIMEDIA NETWORKING SYSTEM

Gabriel-Miro Muntean, Liam Murphy
Performance Engineering Laboratory, School of Electronic Engineering,
Dublin City University, Dublin 9, Ireland
munteang@eeng.dcu.ie, murphyl@eeng.dcu.ie

Abstract

The transmission of related multimedia data causes problems because of their very large size and continuous nature. Unlike the majority of the existing solutions for transmitting continuous media, which use connectionless protocols, we propose one that uses a connection-oriented protocol (TCP/IP). An object-oriented approach is used to build both server and client, allowing easier system debugging and expansion. We implemented a buffering mechanism which allows us to continue playing for a period when the network load increases. Multithreading is used to solve some problems which require concurrent solutions.

1. Introduction

The process of transmission of related audio, video, images, text and/or data among networked computers is known as multimedia networking. Both audio and video streams have timing requirements as well as a need to play out them in a continuous manner at the receiver. Besides these individual timing requirements, a combined multimedia stream requires synchronization at the receiver to be understandable.

Another problem with multimedia streams is their large filesize. A lot of work has already been done [1] to reduce the size of multimedia files by using different compression algorithms, thereby reducing the necessary bandwidth for transmission and thus the network load. For the system we have built, we have chosen MPEG format for compressing audio and video because of its high compression factor and its ability to adjust the size of the compressed stream depending on the requested quality.

The requested and expected quality of transmission depends on where the multimedia stream is used. In some applications a high quality of service (QoS) is important. In other cases the expectations for QoS are lower, but a smooth flow of the multimedia stream is desirable. There were some proposals to improve multimedia transmission quality like traffic control and dynamic bandwidth renegotiation at the server [2], dynamic bandwidth allocation and flow/congestion control [3], rate shaping [4] or a feedback mechanisms [5], [6]. The majority of them were either theoretical studies or simulations. Our client-server system implements a real-time multimedia streaming over the network. The current paper discusses some of the software solutions used.

2. The System Overview

Our client-server system has two main parts: the client and the server application (Fig 1). The implementation of the both client and server applications required the presence of a unit in charge of establishing and controlling the connections between them. Another one acquires information (e.g. video, audio) at the server and to play or display it at a client. A third unit is responsible for encoding (at the server) and decoding (at the client) of the multimedia stream while a fourth one implements the feedback mechanism at both sides (client and server).

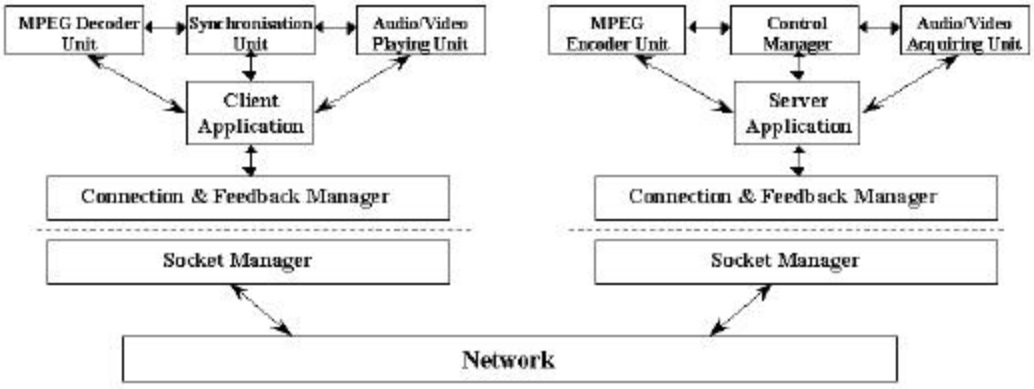

Figure 1: The structure of the Client-Server application

An object-oriented approach has been used in the design of the system, which has been decomposed into subsystems in order to meet the system goals. The implementation uses parallel processing threads which are in charge of particular tasks. When an event occurs (e.g. a connection request arrives, data is received, a user command is sent), the system is informed about it by a message (as in any other windows application) and a special thread is assigned to handle it.

In order to reduce the quantity of data to be sent, we have used MPEG compression for audio and video data [7]. Since it is computationally very complex [8], and we intend to handle real-time transmissions, we have chosen a hardware encoding solution. Because the feedback mechanism needs access to the MPEG stream components, we decided to implement our own software MPEG decoder for video, audio (layer 1, 2 and 3) and system streams.

3. Software Implementation Details

With the system architecture from the Fig. 1, we now discuss in more detail some of the software problems occurred and the solutions we propose.

Client-Server Communication

We have used the services offered by the Windows Sockets 2 API to bind to a particular port and IP address on a host, initiate and accept a connection, send and receive data, and close a connection.

A bi-directional connection, implemented and controlled by "Connection Managers", is needed for data transmission and the implementation of the feedback algorithm. Our

"Connection Managers" improve the standard communication scheme by using the Windows operating system message pump to call the correct function to build the connection or read incoming data, saving the time usually spent polling the sources for possible interruptions.

Ideally the encoding, the transmission over the network, decoding and playing or displaying multimedia should be done at the same rate. Unfortunately, under heavier loading, it is possible that the network could delay different audio or video packets by different amounts. A buffer is necessary at each receiver (i.e. client) to store the arrived data until it is processed, and again after decoding until it is played or displayed. Our feedback mechanism also makes use of receiver-side buffers and therefore a relatively complex buffering mechanism has been developed.

Buffering Mechanism

A circular system of buffers (Fig. 2) has been implemented at the client side, allowing data to be received and stored into a FREE buffer, which will then be marked as FULL. The Decoder Unit takes FULL buffers and decompresses the data, storing the ready-to-display or ready-to-play data into READY buffers.

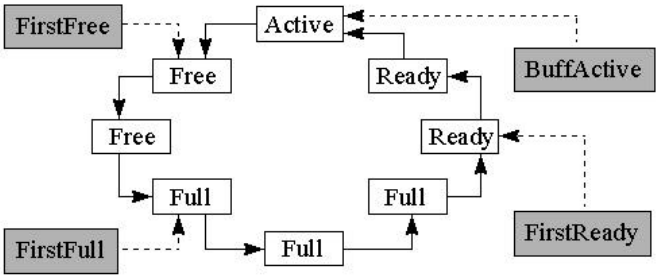

Figure 2: The circular buffer system at a client

Separate threads are playing or displaying data which is contained in buffers. These buffers are marked ACTIVE while playing and immediately after the process is over are marked FREE. Thus we allocate a number of buffers and reuse them, without losing the time to allocate and free them one-by-one as needed. One of the disadvantages of this scheme is the high memory space usage all the time, even if there is no data to store in all the buffers; but this is less of a problem with today's memory capacities. By introducing different pointers for the first FREE, FULL, READY and ACTIVE marked buffers respectively, we reduce to the minimum the time spent on sequential parsing of the circular buffer structure. Such a circular system of buffers is allocated to each type of MPEG stream (audio, video and system), both at server and at client side.

Multithreading

Our system uses multithreading for both server and client applications. At the server side, different threads are assigned to read data from a file and to send them to the client as is shown in Fig. 3. FillThread is in charge of filling the server buffer (SrvCircBuff) with the encoded data at the same time as they are being read and sent for decoding by a local decoder thread. TxThread transmits data from SrvCircBuff to the client every time a signal comes from a timer.

At the client side (Fig. 3), there are three threads for each instance of an audio or video stream. FillThread takes the encoded data received from the server and stores it in a

circular buffer (CliCircBuff). DecThread decodes data taken from the CliCircBuff and sends the decoded data to the audio or video driver, which puts it in the DriverBuff. PlayThread plays or displays the decoded data already sent to the Play/Display Driver, according to the stream's properties.

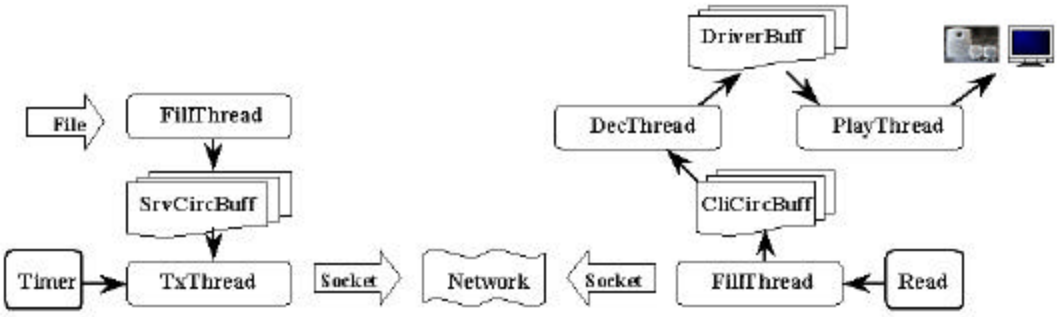

Figure 3: Data flow through threads and buffers both at server and at client sides

Critical sections are used in order to protect the shared resources (e.g. circular buffers) from being accessed by more than one thread at a time. Events are used to signal threads (e.g. a new buffer was filled and is waiting to be decoded). The Windows message system wakes up a thread if an important event occurs, such as receiving a packet with data from the server.

The system stream decoding needs to be able to decode both audio and video encoded data. Thus an extra number of threads have to be launched. Unfortunately, the CPU time slice given by the operating system to all the threads was not enough for a smooth play (especially of the audio stream). To keep up with the playing, the decoding thread had to be granted a higher priority than the other threads. This in turn meant that the system decoding thread also had to be higher priority.

Streams Synchronization

The Synchronization Unit is not active unless a system stream is transmitted. The MPEG system decoder sends data to the video and audio circular buffer systems at irregular intervals. With the help of the decoding time stamps (DTS) and the presentation time stamps (PTS), the system, audio and video stream decoders are able to decode received data and play/display them at the appropriate times.

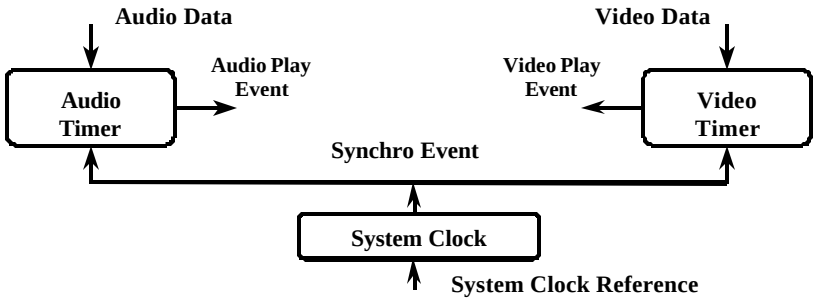

Figure 4: Synchronization Unit

A System Clock is created by the Synchronization Unit at the client side (Fig. 4) when the server sends the first part of the system file requested. The initialization and adjusting is done when the client decodes the System Clock Reference (SCR) components of the system

stream. DTS and PTS are sent along with the decoded data to the play/display drivers. The System Clock repeatedly sends synchronization events (Synchro Event) to the audio playing thread and to the video displaying thread. If the two streams are not played or displayed simultaneously, by using DTS, PTS and the Synchro Event sent by the System Clock, the skew can be reduced to zero by adjusting the displaying rate of the video. We didn't interfere with the audio playing thread because from experience we noticed that any change in the rate the audio data are played badly affects the general quality.

4. Preliminary Results

The server was built using multi-template and multi-document approach, so it can open more than one type of document template and, for each document template, more than one document at a time. Thus the server application can accept client requests for connections while playing one or more MPEG or AVI audio, video and system files.

The following charts show the buffers occupancy for both audio and video, in both cases: elementary streams and system stream. Next we will discuss the results obtained allocating a Circular Buffer of the type described earlier, with 500 equal size buffers. The size of the buffers depends on the case: for audio we have used 4096 bytes large buffers and for video 16384 bytes large buffers.

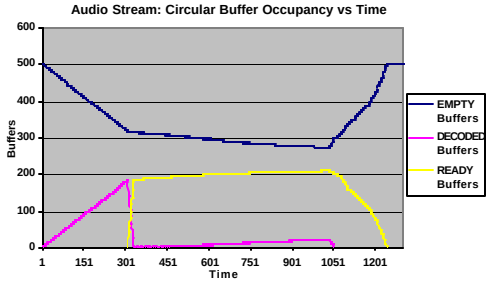

Figure 5: Audio Stream: Buffer Occupancy vs. Time

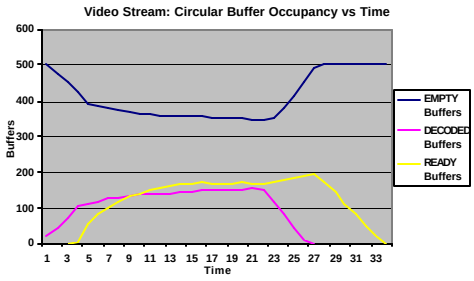

Figure 6: Video Stream: Buffer Occupancy vs. Time

Looking at figures 5 and 6, in which the number of different marked buffers against the time were plotted, we can see that, when the stream the user wants to be played has been chosen, the incoming data is decoded and some of the EMPTY marked buffers are filled. Thus an increasing trend of the DECODED marked buffers can be remarked. When the streams started to be played, the number of buffers sent to the playing driver increases. In the case of audio stream whose decoding is not so CPU time-consuming, the increase is sharper than the one observed in the case of video elementary stream.

To be able to play the stream correctly to the end, we have to decode, in a timely fashion, enough data to submit to the play or display driver. When the decoding threads finish their jobs, the playing or displaying ones continue their jobs until the data they have received is finished. We notice that the number chosen for the available buffers (500) is big enough to cover the requirements of both streams. To save memory, 250 buffers for audio decoding and 200 buffers for video decoding solve the problem; even a smaller number of buffers may be enough.

The system stream has a much harder job to be done in the same time, because its streams have to be split before decoding is performed on each of them. Thus, looking at Fig. 7

and Fig. 8 which show the audio and video buffer occupancy respectively, we notice a more gradual increase of the decoded data filled buffers and of the ready-to-play or ready-to-display one. In the system case as well, the number of the used buffers does not tend to exceed the number allocated.

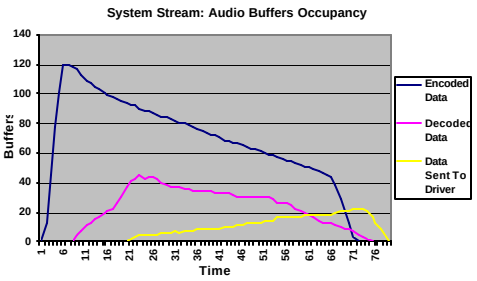

Figure 7: System Stream: Audio Buffer Occupancy vs. Time

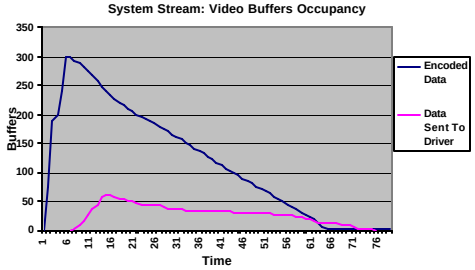

Figure 8: System Stream: Video Buffer Occupancy vs. Time

5. Conclusion and Further Development

The prototype system is still under development. The "Connection Managers" are running as required. They have been tested by sending files of both MPEG and AVI types and have yielded satisfactory results. The MPEG Decoding Unit successfully decodes MPEG-1 system, audio and video streams, with the help of the buffering scheme implemented. Some improvements may be necessary regarding system performance. Streaming has been realized with the help of buffers. Thus a multimedia stream can be decoded even though the whole file is not available yet.

References

[1] W. T. Ooi, B. Smith, S. Mukhopadhyay, H. H. Chan, S. Weiss and M. Chiu, ”The Dalí Multimedia Software Library”, SPIE Multimedia Computing and Networking 1999, San Jose, CA, January 25-27, http://www.cs.cornell.edu/zeno/Papers/#Dali

[2] B. Zheng and M. Atiquzzaman, “Traffic Management of Multimedia over ATM Networks”, *IEEE Communications Magazine*, vol. 37, no. 1, Jan. 1999, pp. 33-38

[3] M. Krunz, “Bandwidth Allocation Strategies for Transporting Variable-Bit-Rate Video Traffic”, *IEEE Communications Magazine*, vol. 37, no. 1, Jan. 1999, pp. 40-46

[4] B. Sheu and A. Ortega, "Microsystems Technology for Multimedia Applications", *IEEE Press*, May, 1995

[5] P. Bindal and L. Murphy, “Multimedia Feedback Control in ATM Local Area Networks”, *Proceedings of the 35th ACM Southeast Conference*, Murfreesboro, TN, April 2-4, 1997, pp. 243-250

[6] G. Muntean, L. Murphy, "An Object-Oriented Prototype System for Feedback Controlled Multimedia Networking", *ISSC*, University College of Dublin, Ireland, June 29-30, 2000

[7] ISO/IEC International Standard 11172, “MPEG-1 - Coding of Moving Pictures and Associated Audio for Digital Storage Media up to 1.5 Mbits/s”, Nov. 1993.

[8] D. LeGall, “MPEG: A Video Compression standard for Multimedia Applications”, *Communications of the ACM*, vol. 34, no. 4, April 1991, pp. 46-58