A SYNCHRONIZATION SCHEME FOR DELIVERING MULTIMEDIA DATA STREAMS

By

AFTAB ACHMAD LUBIS

Bachelor of Engineering Science

Bandung Institute of Technology

Bandung, Indonesia

1988

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the degree of
MASTER OF SCIENCE
May, 1997

A SYNCHRONIZATION SCHEME FOR DELIVERING MULTIMEDIA DATA STREAMS

Thesis Approved:

[illegible]
Thesis Advisor

J. Chandler

[illegible]

Thomas C. Collins
Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my thesis advisor, Dr. K.M. George for his intelligent supervision, constructive guidance, inspiration, and help. My sincere appreciation extends to Dr. J.P. Chandler and Dr. J. Hatcliff for serving on my graduate committees, and providing me with some feedback to improve my thesis.

I would like to give my special appreciation to my wife, Dewi Gunawati, for her encouragement at times of difficulty, love and understanding throughout this whole process. My respectful and very special thanks also go to my parents, Mr. and Mrs. Syarif Lubis, and parents-in-law, Mr. and Mrs. Bastari Halik, for their support, love, and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

B	Bi-directional predicted frame
DCT	Discrete Cousine Transform
I	Intra frame
ISO	International Standard Organization
kHz	kiloHertz
MDU	Multimedia Data Unit
MPEG	Motion Picture Expert Group
P	Predicted frame
PCM	Pulse Code Modulation
RGB	Red Green Blue
SNR	Signal Noise Ration
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

CHAPTER 1

INTRODUCTION

When computers were introduced, they were merely intended as tools for doing computations. As software and hardware have become further developed, computers are more widely used as tools for other purposes such as entertainment, communications, learning, record keeping, and so on. Eventually, computer data types also become more diverse. Multimedia systems are examples of computers that gather and process various media or data types, such as text, voice, audio, video, graphics, and images. Nowadays, research and development directions are leading toward multimedia systems connected to computer networks, in which processing and representing multimedia data streams introduce a number of new technical problems. These multimedia systems are often called distributed multimedia systems. Technical problems associated with distributed multimedia systems are as follows [CLSWCO95]:

- *synchronization*: deals with synchronizing multiple media streams at a presentation site.
- *media transmission*: deals with the transmission of multimedia streams between sites.

In this thesis work, we are proposing a synchronization scheme in delivering multimedia data streams. The basic ideas of our synchronization scheme are as follows:

- Generating timestamps (markers) for inter media synchronization purposes.
- Discarding partially lower priority multimedia data stream based on network traffic.

Generating timestamps seems to be a standard for delivering multimedia over computer networks. Ravindaran [Rav93] and Sheperd *et al* [SS90] are among researchers who have been studying and implementing timestamps. So far, discarding partially lower priority multimedia data stream has never been done. The idea came up since some of multimedia data streams might be generated based on their past and future data streams. Thereafter, we first have to define and choose which multimedia data streams fall into the category of lower priority data streams.

As stated previously, information sources can be graphics, animation, images, audio, and full motion video. However, for simplicity of this thesis work, we are only assuming that information sources are *full motion video* and *audio*. Then, we classify streams generated from those sources as follows:

- High Priority: audio stream is considered the higher priority stream, since losing some parts of the stream may convey wrong messages.
- Low Priority: video stream is considered the lower priority stream, since losing some parts of the stream may not give a wrong interpretation as long as the associated audio stream is still fully correct.

For sending a video stream over a computer network in this synchronization scheme, a standard for storing and retrieving a video stream is used. This standard is known as MPEG-1. As described later in chapter 2, the video stream is compressed such that the stream is consisting of I, B, and P frames. From these three types of frames, I frames are the frames that contain most information, since B and P frames are generated based on I frames. Hence, when network congestion occurs, discarding B and/or P frames, we can expect to get a good interpretation of the video stream.

CHAPTER 2

MPEG STANDARD

In multimedia systems, audio, images, and video data captured in analog signals need to be digitized before they are processed. These analog signals produce a vast amount of digital signals that might be stored either in computers' memory or secondary storage. These digital signals might also be shared and accessed by distributed multimedia systems. Hence, a standard governing formatting and storing these digital signals is obviously needed.

An international organization, called the International Standard Organization (ISO), has developed a standard format of video and audio stored in digital storage devices. The standard is known as MPEG, named after its experts group [Gal91]. The MPEG standard is divided into MPEG-Video, MPEG-Audio, and MPEG-System standards.

2.1 MPEG-Video

Video data is basically a continuous stream. Digitizing of video data takes place by sampling the data stream at a certain rate. Usually the sampling rate is about 30 images per second (the standard television rate). The result of sampled video data is a set of sequential images, I1,I2,...,In, which can be represented as a two-dimensional array of RGB triplets, where an RGB triplet is a set of three values that give the red, green, and blue levels in the image [PSR93]. The RGB values are represented in 24 bits of information, in which each color level is represented as 8 bits of information.

Since most of these sequential images do not change much within a small time interval, MPEG encoding is done by exploiting temporal locality among these images. It means that some of images can be generated by referencing other images close to them, and redundancy among images might be removed in which data (images) are compressed. This data compression could be implemented either in software or in hardware, and can be classified into **lossless** and **lossy** techniques [Fur94]. In a lossless technique, the original representation can be **perfectly** recovered, while in a lossy technique, the original representation still can be recovered with some loss of accuracy. Since the lossy technique gives a higher compression ratio than the lossless technique, it is often used in image and video compression.

MPEG-Video coding uses three techniques [SRD93] to compress video data. The first technique is called *transform coding* and takes advantage of two facts :

1. high frequency of visual information is more likely unrecognized by human eyes
2. using a mathematical transformation, the concentrated energy of an image can be represented in fewer values.

In MPEG transform coding, each RGB triplet in an image is transformed into a YCrCb triplet. The Y value represents the luminance (black and white) level and Cr/Cb values represent chrominance (color information). Since the human eyes are less sensitive to chrominance than luminance, the Cr and Cb planes are half-sampled. In other words, all luminance information (Y), which contains 8 bits per pixel of luminance information, is transformed, while in the case of chrominance information (Cr and Cb), half of it is removed by cutting in half the width and height of the Cr and Cb planes in the horizontal and vertical directions. This transformation reduces by 4:1 the ratio of Cr and CB information, and generates 2 bits per pixel of Cr information and 2 bits per pixel of Cb information. This process is a lossy compression, since the 24 bits per pixel of RGB information is reduced to 12 bits of YCrCb information, which results in a 2:1 compression ratio. The process continues by dividing the image into macroblocks. In the original image, the macroblock is a 16-by-16 pixel area. A transform coded macroblock is composed of a set of six 8-by-8 pixel blocks, four from the Y plane and one from each of the subsampled Cr and Cb planes. These six 8-by-8 pixel blocks then is transformed using Discrete Cosine Transform (DCT), which decomposes them into frequencies (see Appendix A).

The second technique MPEG used is called motion compensation. It exploits the fact that a frame I2 is likely to be similar to its predecessor I1, and I2 can be nearly constructed from I1. Many of the macroblocks in frame I2 can be approximated by pieces of frame I1, which is called a reference frame. Similarly, many macroblocks in I3 can be approximated by pieces of either I2 and/or I1. The appropriate piece of the reference frame requires fewer bits to encode than original pixels. This coding results in significant data compression. Further compression can be obtained if, at the time I2 is coded both I1 and I3 are available as reference frames, then I2 can then be built using both I1 and I3. When a large pool of reference frames is available, motion compensation can be used to construct more of the frame being encoded, reducing the number of bits required to encode the frame.

The final technique used by MPEG to compress video data is entropy coding. After motion compensation and transform coding, a final pass is made over the data using Huffman encoding.

Based on a coding procedure, we might classify frames into three types which are given below:

1. Intra (I) frame. This frame is encoded as a single image, with no reference to any past or future frames (see Figure 2.1). The block is first transformed from the spatial domain into a frequency domain using Discrete Cousine Transform (DCT), which separates the signal into independent frequency bands. Most frequency information is in the upper left corner of the resulting block. At this point, the data is quantized. Quantization can be thought of as ignoring lower order bits, and is the only lossy part of the compression scheme other than subsampling. The resulting data is then run-length encoded in a zig-zag ordering to optimize compression.
2. Predicted (P) frame. This frame is encoded relative to the past reference frame that can be either a P or I frame (see Figure 2.2). The past reference frame is the closest preceding reference frame. More formally, the past reference frame for a frame Ii is the reference frame Ij such that Ij comes before Ii in display order, and there are no other reference frames between Ij and Ii in display order. Each macroblock in a P frame can be encoded either as an I macroblock or as a P macroblock. An I macroblock is coded just like a macroblock in an I frame. A P-macroblock is some 16x16 area of the past reference frame, plus an error term. To specify the 16x16 area of the reference frame, a motion vector is transmitted. The error term is generated using the DCT, quantization, and run-length encoding.
3. Bi-directional predicted (B) frame. This frame is encoded relative to the past reference frame, the future reference frame, or both frames. The future reference frame is the closest following reference frame. It can be either an I or P frame. More

formally, the future reference frame for a frame Ii is the reference frame Ik such that Ik comes after Ii in display order, and there are no other reference frames between Ii and Ik in display order. The encoding for B frames is similar to that for P frames, except that motion vectors may refer to areas in the future reference frame.

A typical IPB sequence is shown in Figure 2.3. The arrows represent the inter-frame dependencies.

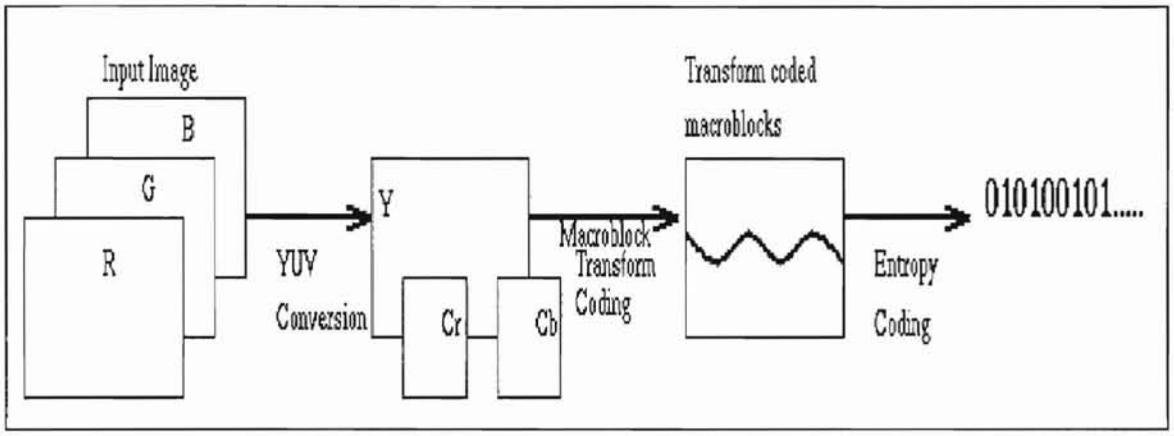

Figure 2.1 Intra Frame Coding [PSR93]

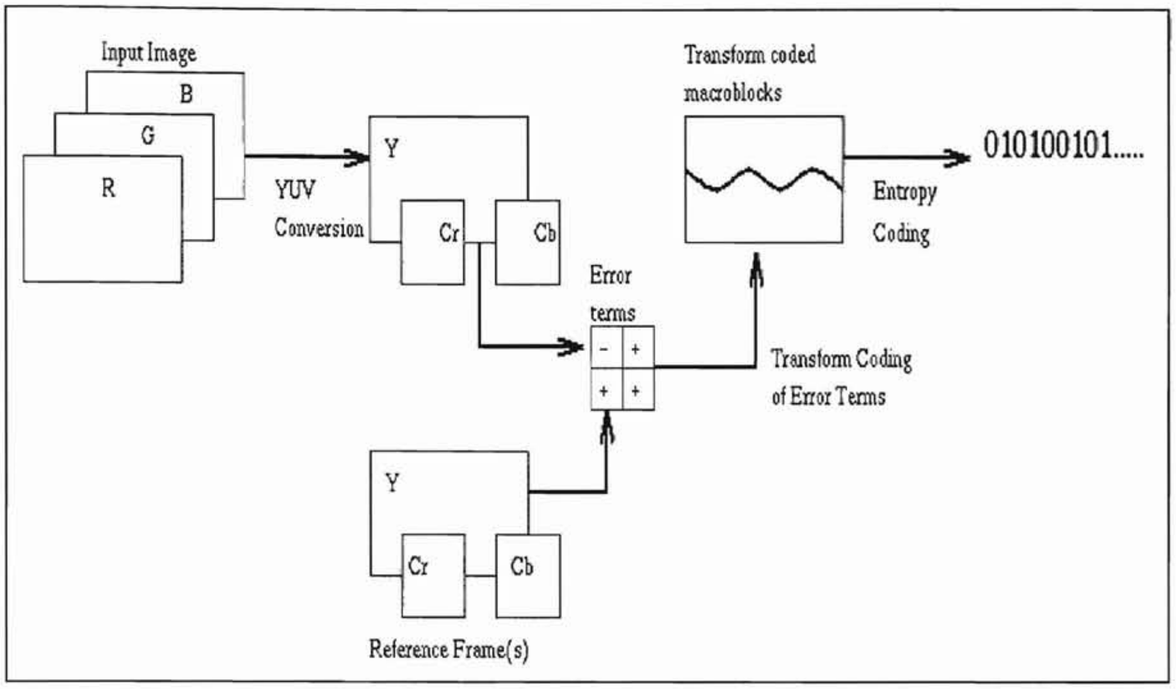

Figure 2.2 B/P Frame Coding [PSR93]

To rebuild YCrCb frame, the following operations are needed [PSR93]:

1. the entropy coding must be inverted,
2. for P and B frames, the motion vectors must be reconstructed and the appropriate parts of the reference frame copied, and
3. the error terms must be decoded and incorporated.

Once the YCrCb frame has been built, the frame is converted to an appropriate representation for display. This step is called dithering.

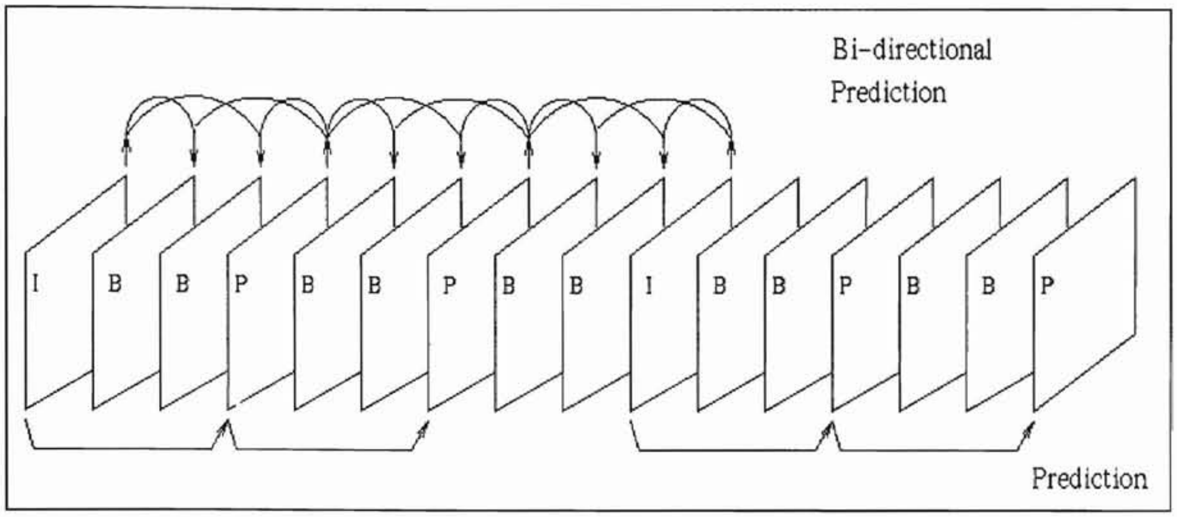

Figure 2.3 Configuration of IBP Frames [Fur94]

2.2 MPEG-Audio

Digital audio is a numerical representation of the pressure wave from an audio source. The usual representation is called PCM, for Pulse Code Modulation [Pan95]. The analog waveform is encoded by a series of numbers or samples, typically several thousands per second. The encoding rate is called the sampling rate and is directly proportional to the sound quality; the more samples per second, the higher the (theoretical) fidelity of the sampled sound. Sampling rates vary from 8 kHz for telephone-quality speech to the high-fidelity rates of 44.1 kHz (Compact Disc) [DPBF 96]. The higher the sampling rate, the more memory is needed for a given audio sample. ISO/MPEG has define a standard, called MPEG-Audio, for coding PCM audio signals with sampling rates of 32, 44.1, and 48 kHz at bit rates from 32 to 448 kbits/second. The standard defines three layers of coding algorithms which are Layer 1, 2, and 3. In all three layers the input PCM audio signal is converted from the time domain into a frequency domain. The Layer 1 coding scheme is known as sub-band coding which is based on resolving the audio signal into spectral components, or sub-bands [Pan95]. The PCM audio input is set to be simultaneously passed through a filter bank and a psychoacoustic model. The filter bank determines how the input stream should be divided as sub-bands, while the psychoacoustic model sets the ratio of the signal energy to the masking threshold for each sub-band. Using the signal to mask ratio, a bit/noise allocation determines how to share the available code bits for the quantization of sub-bands. Then, a bit stream formatting formats the quantization of sub-bands into a coded bitstream.

Layer 2 and 3 are improvement of Layer 1 coding [DPBF 96]. In Layer 2, redundancy and irrelevance on scale factors are removed. In Layer 3, a hybrid filter bank is used to get better frequency resolution.

CHAPTER 3

NETWORKED MULTIMEDIA APPLICATIONS

Multimedia systems gather various information sources, such as text, audio, video, graphics, and images, which are processed by a wide range of applications such as remote learning, multimedia mailing system, collaborative work systems, multimedia communication systems (video phone, conference system and information on demand system [Fur94]. Having various information sources, new technical problems need to be taken care of. These problems basically result from the different features among different information sources. More problems arise when multimedia systems are attached to computer networks. Synchronization and media transmission problems are interesting research areas in networked multimedia systems.

3.1 Synchronization

Synchronization problem arises when several related media are to be played back or displayed with respect to their temporal relationship constraints. It becomes more complex when these related media are also sent over computer networks, because of data rates networks cannot be predicted. Two requirements [KK94] exist on presenting multimedia data streams. The first requirement is **Intra Media**, in which the continuity of multimedia data streams needs to be maintained in order to provide smooth presentation. The second requirement is **Inter Media** which requires multimedia data to have temporal relationship in order to be presented synchronously.

To solve the synchronization problem, many techniques have been proposed. Ravindran [Rav93] has proposed transport models for temporal synchronization scheme. At source site the data streams are segmented as Media Data Units (MDUs), and each data stream is sent on separate channel. At destination site, MDUs are buffered before they are presented. To build temporal relationship among data streams, we wait in an interval of length *K* for completely receiving all data streams (since one MDU might arrive earlier than others).

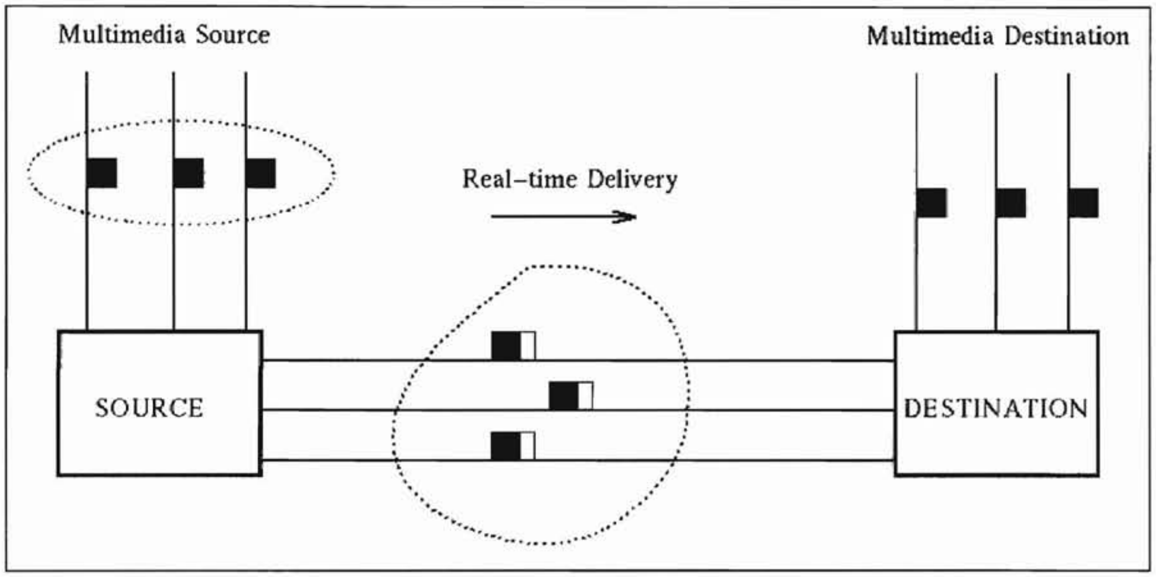

Figure 3.1 Synchronization Markers (SMs) Appended in Data Streams[KK94]

Shepherd *et al* [SS90] have proposed a mechanism using synchronization markers, denoted as SMs, which are integrated with application layer of ISO model. The synchronization markers are inserted into data streams at sender site. The destination site buffers the arriving marked data stream until all marked data streams are received. Then, using the temporal relationships included in SMs, the synchronization is performed. Figure 3.1 shows this mechanism. Black squares at multimedia source represent multimedia data streams. From multimedia source, each data stream is sent using separate channel, and appended with synchronization marker, which is denoted as a blank square adjacent to the black squares. Synchronization markers are removed at multimedia destination. These synchronized streams then are passed on to applications running at multimedia destination.

Basically, synchronization can be achieved by implementing the following algorithm [CLSWC95]:

```
loop{
        /* estimate the audio waiting time in advance */
        estimate_audio_waiting_time();
        /* show the related audio and video frame */
        play_audio_segment();
        play_video_frame();
        /* waiting for audio data consumed completed by audio device */
        sleep(audio_waiting_time);
} until end_of_playback;
```

where audio waiting time is calculated as

$$audio\_waiting\_time = \frac{audio\_segment\_size}{audio\_sampling\_rate} + overhead$$

$$over\_head = data\_access\_time + system\_overhead$$

Ideally, the audio waiting time is equal to the size of audio segment divided by the sampling rate of the audio device. In single process environments, the overhead includes interrupt service time and instruction execution time. The data access time and system overhead time is critical. If the estimated time interval is longer than the real one, it will be too late to load the next audio segment into the audio device in time. The buffer will be exhausted before the next audio segment arrives. It leads to no audio data to be played between these two audio segment. This results in discontinuous audio output. If the estimated time interval is shorter than the real one, it would be too early to load the next audio segment into the audio device. In this case, audio device buffer could be full and the phenomenon of out of synchronization occurs.

In the multi-process environment, process context switch time, which is difficult to predict, must also be taken into account. The synchronization is implemented using parent-child scheme, in which the following criteria must be satisfied [CLSWCO95]:

1. There will be child processes supporting their parent processes.
2. Each process is responsible for playing back one medium.
3. The parent process plays the role of monitoring its child processes and playing back the highest priority medium.
4. The child processes play back the lower priority media.
5. The responsibilities of the parent processes are
    - pre-calculate the vital synchronization information.
    - fork (generate) the child processes before the playback starts.
    - kill (terminate) the child processes after playback ends.
6. Synchronization mechanism among different media processes: Two approaches are implemented in this model.
    - ***relative synchronization***: Based on pre-calculated synchronization information (some synchronization points), media processes can synchronize with each other through some well-known internal process communication techniques such as shared memory, socket, and pipe.

- ***absolute synchronization***: Based on pre-calculated synchronization information (some time table), each medium process synchronizes with the global system clock.

3.2 Buffer Requirement

Most networked multimedia systems rely heavily on the availability of communication services. In todays computer networks, there are many types of network operating systems attached to the networks. Handling multimedia data streams over computer networks can be done by the network layers (protocols) and/or the multimedia applications themselves. Buffering is one technique in handling multimedia data streams, and it is usually be done at receiver sites (multimedia destinations). Two possible buffering approaches, which are traditional buffering and application oriented buffering, have been described by Velthuys *et al* [VLP95]. They are traditional buffering and application-oriented buffering.

3.2.1 Traditional Buffering

Figure 3.1 shows incoming network data units (frames) being stored in buffer space, somewhere in main memory. The buffers remain occupied until the application has retrieved the data. Then, the buffers are freed and can be reused for storing subsequent network frames.

One important function performed is segmentation and re-assembly. Often, application data frames are too large to be sent as a single network data frame. Hence, application data frames are segmented and transmitted as a series of network data frames. At the receiving site, the network frames are stored in buffer space, and re-assembled into application data frames.

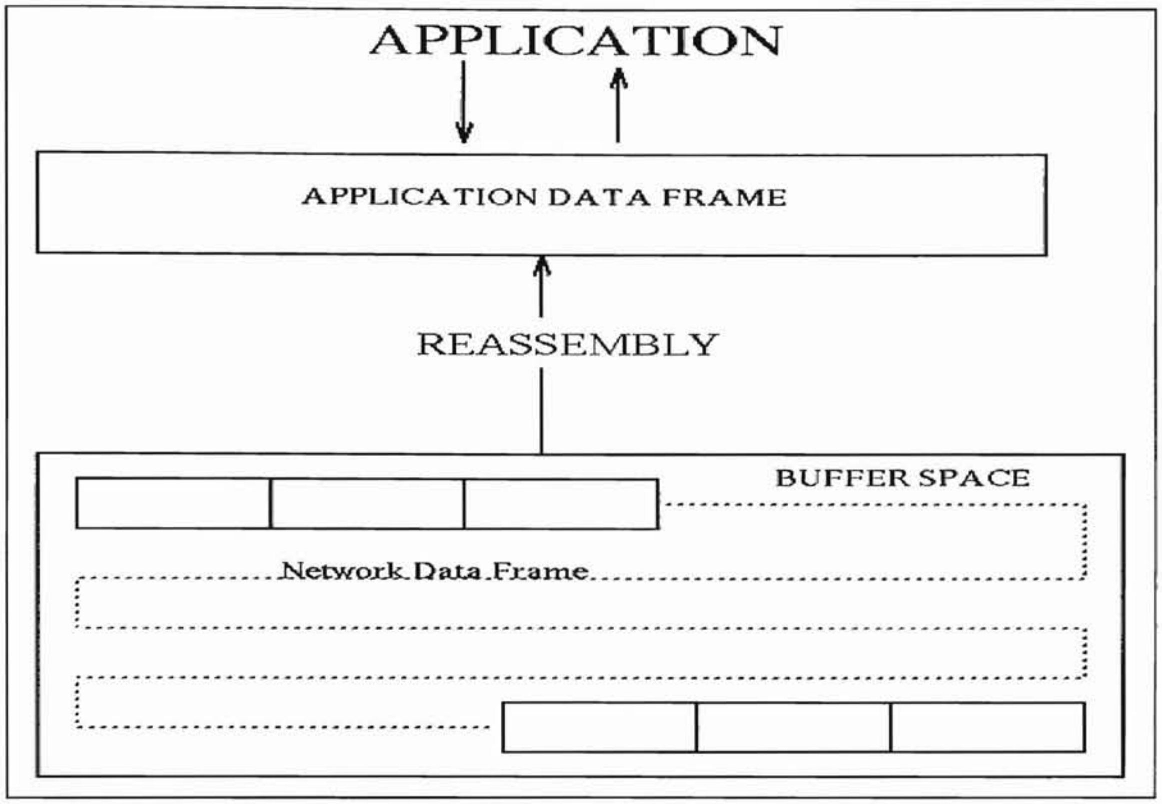

Figure 3.2 Traditional Buffering [VLP95]

In some multimedia systems, buffering management is adjusted to the network operating systems (network oriented). Since buffer space is needed for receiving data frames from the network, it seems appropriate to segment these buffers in data units that are equal to the size of the network data frames. Then, if there are some data frames received incorrectly (caused by some transmission error), network operating systems will correct or remove them..

3.2.2 Application-oriented buffering

Nowadays, reliability of modern networks is more sophisticated, only few network frames are lost, corrupted, or delivered out off order. Application data frames can be reassembled from incoming network data frames (figure 3.3) which are received consecutively in contiguous buffer space, e.g., network data frame *n+1* follows network data frame *n*, etc. Once all network data frames for an entire application data frame has been received (uncorrupted), the application data can be delivered to the application.

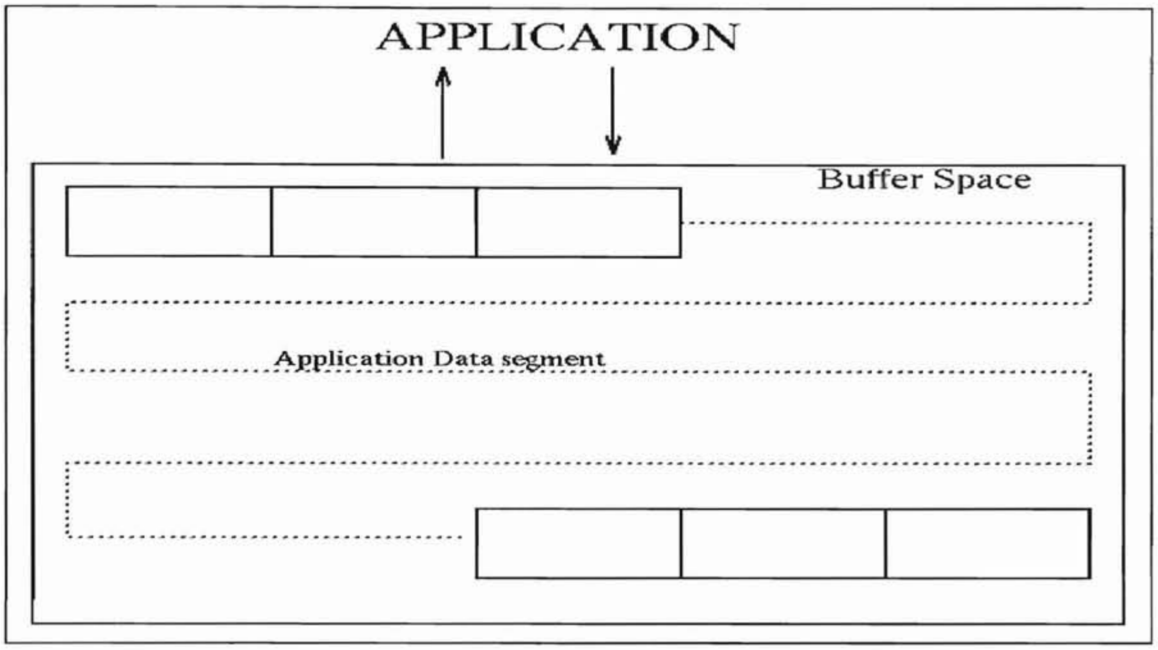

Figure 3.3 Application-oriented Buffering [VLP95]

In most cases, neither the communication protocols delivering the frames nor the buffering mechanism storing them understand the data received from the network. The data is not meaningful to the application until it is reassembled.

Meanwhile, audio and video applications need ongoing streams of incoming application data frames. We often end up with so little time left in detecting and recovering errors in reassembled data streams [VLP95]. However, this does not mean that the reassembled data frames cannot be corrected. Some of data streams might be tolerated in having corrupted data. A video stream is one of the examples of tolerated data. A corrupted network frame means that one out of a series of images is incorrect. In this case, copying the previous image can be a solution. In audio stream, the lost network data frame cannot be tolerated. Loosing one network frame may lead us to different perception. A solution for this problem is to retransmit the corrupted network frame, which may introduce an additional delay. This buffering approach is often called as application-oriented buffering.

3.2.3 Buffer Management Philosophies

Buffers can be rescheduled for allocation in one of two strategies [VLP95]:

- underestimating actual usage (or pessimistic): allocating more buffer space than likely to be needed. This comes with the risk of buffer conflicts because of overcommitment.
- overestimating actual usage (or optimistic): allocating fewer buffers than requested. This approach avoids buffer conflicts, it assumes that applications demand more resources than necessary.

The only situation which causes problems during reception and presentation is if fewer buffers are allocated then the actual need, since this means that more data is lost, causing a reduction in presentation quality.

CHAPTER 4

IMPLEMENTATION

4.1 Hardware

In this implementation we use Sun Sparc 20 as workstations, and Solaris 5.4 and Openwindows as their operating systems and X-Window systems. Solaris 5.4 is actually Unix System V operating system which has facilities such as semaphore for controlling mutual exclusion of processes, and TLI sockets for networking. These facilities are needed in our implementation.

4.2 Software

Berkeley Multimedia Research Center at University of California at Berkeley has a developed program called ***MPEG-1 Encoder and Decoder*** [RPSGH95] which implements MPEG-1 standard for encoding and decoding video stream. In this implementation, MPEG-1 Encoder can be called by Server process to encode incoming video stream. MPEG-1 Decoder must be run by Client process.

Tobias Bading from Berlin University of Technology, has developed ***MAPLAY*** [Bad94], an audio player. It decodes audio streams sampled at frequencies 32, 44.1, or 48 kHZ, into raw 16 bit PCM stream. This software is called at Client site at presentation time.

Phillip Lougher from Computing Department, Lancaster University, has developed a software called ***mpegUtil*** [Loug95] that can generate information on MPEG video files, and extract partial image sequences. This software is used at server site. It takes compressed video file, and can generate another compressed video file containing a certain sequence images.

TCP connection is set up to make Client and Server communicate. This is connection oriented, which means connection must first be established between Client and Server before they do transactions.

4.3 Method of Implementation

As stated in previous section, in multi-process environment such as Unix, the synchronization must use parent-child scheme. However, in our implementation, we made slight changes. Client server scheme is also used, since multimedia data streams are sent over computer network, in which multimedia source is the server, while multimedia destination is the client. To be able to accept multi requests from clients, concurrent server is a better strategy. Hence, the following scenario is implemented:

1. Relative synchronization approach is implemented. Inter Process Communication techniques used are as follows:
    - Socket: for communicating between server and client processes.
    - Shared Memory: letting appropriate processes to access the same information.
    - Semaphore: controlling processes' access to share memory.
    - Pipe: for communicating between parent server/client to their child processes.
2. There will be two child server processes supporting parent server processes.
3. There will be two child client processes supporting parent child processes.
4. One child server process is responsible for sending video segments, while another is responsible for sending audio segments.

5. One child client process is responsible for receiving, buffering and reassembling video segments, while another is responsible for buffering, receiving and reassembling audio segments.
6. Application-oriented and pessimistic buffering strategy is implemented.
7. The responsibilities of parent server are:
    - Fork off two server child processes.
    - Pre-calculate the synchronization information: parent server divides incoming video and audio streams into segments.
    - Create timestamp for each segment every time an I (intra) frame is read from compressed video stream generated by Berkeley's MPEG1 Encoder. Hence, the number of I frames is the number of segments generated. The timestamp information is stored at a shared memory which can be accessed by both audio and video server .
    - Use two semaphore variables to avoid overrunning and to control child server processes.
    - Detect when discarding B and P frames should be taking place, and also let video server process to send I frame only.
    - Kill (terminate) child processes after clients' connection are disconnected.
8. The responsibilities of parent client are:
    - Fork off two client child processes.
    - Check how many segments are already received in buffer space.
    - Inform parent server to send all frames, no B frames, or no B/P frames.

- Use also two semaphore variables to avoid child client processes from overrunning.
- Adjust frame rates based on what type of frames are sent from server processes.
- Once video and audio streams are ready reassembled, calculate audio waiting time, and inform video and audio client processes to play video and audio player.
- Kill (terminate) child client processes after child processes has finished displaying multimedia streams.

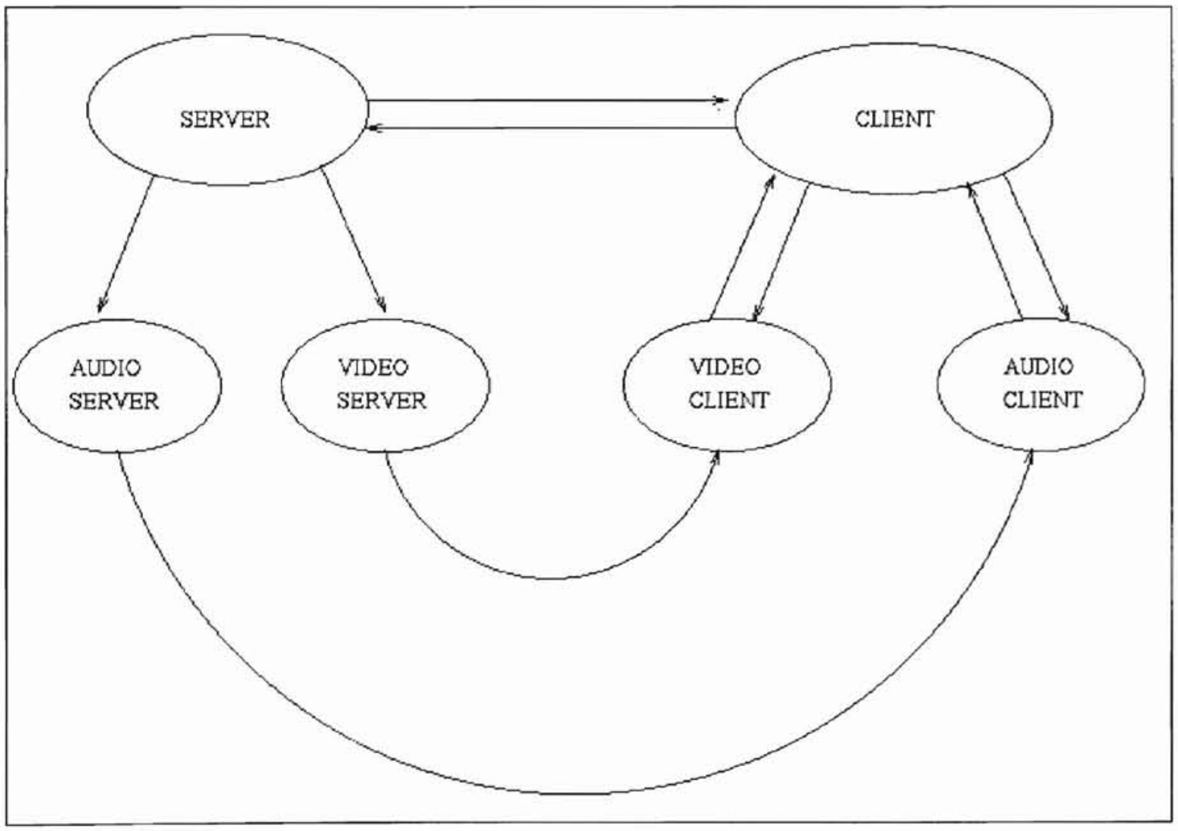

Figure 4.1 Client-Server and Parent-Child Scheme

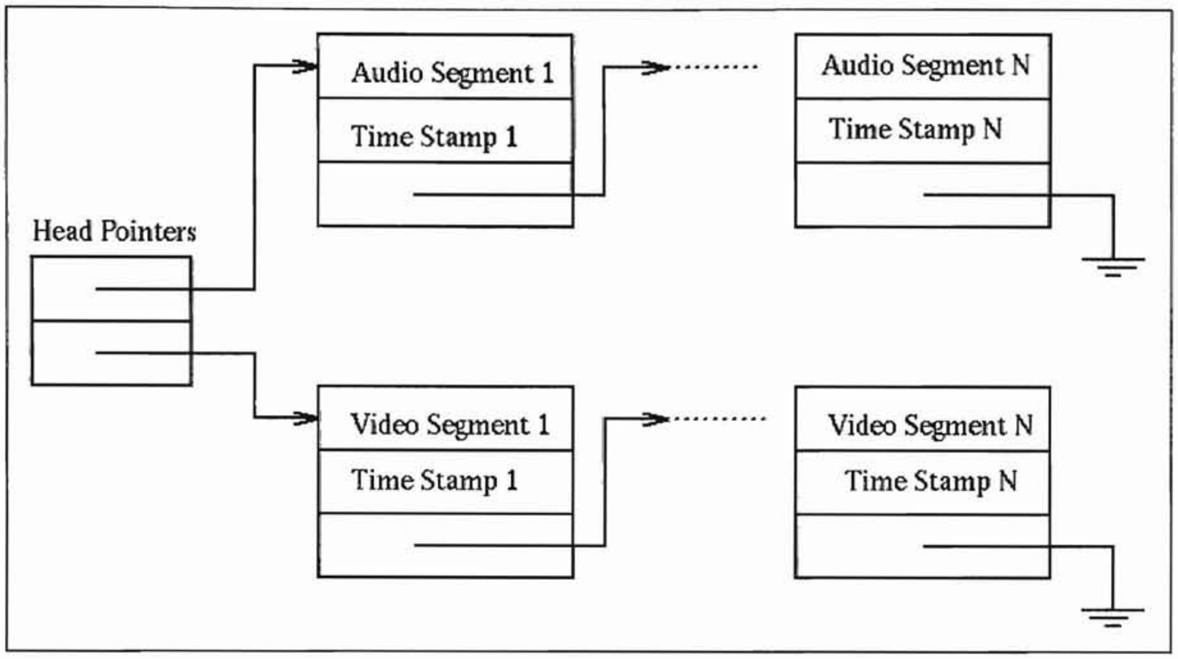

Figure 4.2 Data Structures Used at Server and Client Sites

4.4 Server Process

Since these workstations are not provided with video camera, we may use personal computers to capture multimedia data. Then, the data is loaded into Sun Sparc 20 workstation. Audio and video streams are encoded using Berkeley's MPEG Encoder, which generates compressed video files consisting of sequence of I, P, and B frames, and an audio files sampled at the rate of 44 kHz.

Before clients send requests for multimedia files, parent server has to be set up by binding parent server to a certain port number. The parent server can run in background. Since we implement concurrent server, for every request connection parent server forks off another parent server process which is exactly a duplicate of parent server. The duplicate parent server communicates to and serves the client, while the original parent server waits for other request connections. The following is algorithm used for parent server:

```
create_a_TCP_transport_end_point();
socket = bind_parent_server_address_into a certatin_port();
loop_forever {
    listen_for_any_request_connection();
    new_socket = accept_connection();
    fork_off_server_process(new_socket);  /* concurrent server */
    close(new_socket);
    }
```

For sending the multimedia files, the duplicate parent server then forks off two child processes which are video and audio server child processes. Before forking off child processes, duplicate parent server must set up pipe connections in order to communicate with its child processes.

Type of frames needed must be specified by client before sending or receiving multimedia data streams begin. Client can issue three options which are to send all frames, to discard B frames, or to discard P and B frames. Video server child, who is set up to be run first, calls mpegUtil software to fulfill client request.

For writing timestamp information to data structure, a semaphore variable which is also a shared memory, is used. Video child server process increments this timestamp information before it reads a video segment. Since timestamp information is in shared memory, audio server process can always reads this timestamp information when it needs.

In controlling parent and child server processes, two semaphore variables are used and initially set to zero. Control is first handed to duplicate parent server which reads message from client (using socket) and passes on the message to video child server (using pipe). Based on the message, video child server process reads video segment from source, and sends the video segment back to duplicate parent server using pipe. Next, this video segment is sent to client by duplicate parent server (using socket). Similarly, almost the same sequence steps are followed by audio child server. The difference is that audio child server does not increment the timestamp. Please notice that even all server processes get the same copy of algorithm, they do not execute every statement. For example, statement block for video process is only executed by video child server. The following is the algorithm used by duplicate parent server, video child server, and audio child server.

```
create_and_initialize_semaphore_variables( audio_server_child);
create_and_initialize_semaphore_variables( video_server_child);
create_and_initialize_semaphore_variables( time_stamp);
create_and_initialize_semaphore_variables( audio_server_parent);
create_and_initialize_semaphore_variables( video_server_parent);
create_pipe_for_communication_between( parent_and_audio_server_child);
create_pipe_for_communication_between( parent_and_video_server_child);

for_loop_index(0,..,1) {
  childpid = fork        /* fork off child process */
  if ( child process == video_process) { /* video child server */
    loop_forever {
      DOWN(video_server_child); /* video child gets blocked */
      increment time_stamp;
      read_message_from_pipe( parent_and_video_server_child);
      if( message == "open file") {
          translate_request( message, parameters);
          system_call( mpegUtil -parameters);
          set message = "continue");
      }
      if( message == "continue") {
          read_video_file(filename);
          write_message_into_pipe(parent_and_video_server_child);
          UP( video_server_parent); /* lets parent run */
          if( bytes_read <> default_read_data) /* end of file */
               exit(0);
        }
  }

  if ( child process == audio_process) { /* audio child server */
    loop_forever {
   DOWN(audio_server_child); /* audio child gets blocked */
```

```
        read time_stamp;
        read_message_from_pipe( parent_and_audio_server_child);
        if( message == "open file") {
            translate_request( message, parameters);
            set message = "continue");
        }
        if( message == "continue") {
            read_audio_file(filename);
            write_message_into_pipe(parent_and_audio_server_child);
            UP( audio_server_parent);
            if( bytes_read <> default_read_data) /* end of file */
                exit(0);
        }
    }

} /*  end for_loop_index(0,..,1) */

/* parent process */

while( VIDEO_DATA <> EOF or  AUDIO_DATA <> EOF) {
  if ( VIDEO_DATA <> EOF ) {
      read_message_from_ socket(to_client);
      write_message_into_pipe(parent_and_video_server_child);
      UP(video_server_child); /* lets video child run */
      DOWN(video_server_parent); /* parent gets blocked */
      read_message_from_pipe(parent_and_video_server_child);
      write_message_into_socket(to_client);
  }

  if ( AUDIO_DATA <> EOF ) {
      read_message_from_ socket(to_client);
      write_message_into_pipe(parent_and_audio_server_child);
      UP(audio_server_child); /* lets audio child run */
      DOWN(audio_server_parent); /* parent gets blocked */
      read_message_from_pipe(parent_and_audio_server_child);
      write_message_into_socket(to_client);
  }
}

parent_wait_until_all_child_process_exit();
remove_semaphore_variables_from_memory();
```

```
        create_a_TCP_transport_end-point_and_bind_it();
        connect_to_server();
        if ( connection not established) then
           send_error_message_and_quit();
        call_procedure_doit();
        display_result();

procedure doit():
        create_and_initialize_semaphore_variables( audio_client_child);
        create_and_initialize_semaphore_variables( video_client_child);
        create_and_initialize_semaphore_variables( audio_client_parent);
        create_and_initialize_semaphore_variables( video_client_parent);
        create_pipe_for_communication_between( parent_and_audio_client_child);
        create_pipe_for_communication_between( parent_and_video_client_child);

        for_loop_index(0,..,1) {
          childpid = fork()     /* fork off child process */
          if ( child process == video_process) { /* video child client */
            send_request_to_server();
            loop_forever {
              UP(video_client_parent);
              DOWN(video_client_child);
              read_message_from_pipe( parent_and_video_client_child);
              write_to_file();
              UP(video_client_parent);
              DOWN(video_client_child);
              send_message( "continue");
            }
          }

          if ( child process == audio_process) { /* video client child*/
            loop_forever {
              UP(audio_client_parent);
              DOWN(audio_client_child);
              read_message_from_pipe( parent_and_audi_client_child);
              write_to_file();
              UP(audio_client_parent);
              DOWN(audio_client_child);
              send_message( "continue");
            }
          }
        } /*  end for_loop_index(0,..,1) */
```

```
/* parent process */

while( VIDEO_DATA <> EOF or  AUDIO_DATA <> EOF) {
  if ( VIDEO_DATA <> EOF ) {
      DOWN(video_client_parent);
      read_message_from_video_client_child();
      send_message_to_server();
      read_message_from_ server();
      write_message_to_video_client_child();
      UP(video_client_child); /* lets video child run */
      DOWN(video_client_parent); /* parent gets blocked */
  }

  if ( AUDIO_DATA <> EOF ) {
      DOWN(audio_client_parent);
      read_message_from_audio_client_child();
      send_message_to_server();
      read_message_from_ server();
      write_message_to_audio_client_child();
      UP(audio_client_child); /* lets audio child run */
      DOWN(audio_client_parent); /* parent gets blocked */
  }
}

parent_wait_until_all_child_process_exit();
remove_semaphore_variables_from_memory();
```

CHAPTER 5

RESULTS

Quality of Service is a key parameter to determine the effectiveness of this synchronization scheme. Vogel *et al* [VKBG95] have determined five categories (see Table 5.1) to be used to measure Quality of Service.

Table 5.1 The Five Categories of QOS Parameters [VKBG95]

Category	Example Parameters
Performance-oriented	End-to-end delay and bit rate
Format-oriented	Video resolution, frame rate, storage format, and compression scheme.
Synchronization-oriented	Skew between the beginning of audio and video sequences
Cost-oriented	Connection and data transmission charges and copyright fees
User-oriented	Subjective image and sound quality

Some of the parameters given in table 5.1 are applicable to this work, others are not applicable. End-to-end delay and bit/rate parameters depend on system load which vary from time to time, and therefore is not applicable in our simulation. Connection and data transmission charges cannot be used, since we put both server and client in the same system. Video resolution, compression scheme, or storage format cannot also be used, since we do not alter video resolution, compression scheme, nor storage format. Sound quality cannot be used since we do not alter the sound stream. Hence, parameters to be measured are as follows:

- *frame rates:* measures the number of encoded frames per second. In addition to frame rates, we also measure number of I, P, or B frames sent.
- *skew*: measures the synchronization between video and audio streams.
- *video quality*: measures the smoothness video stream presentation. It is subject to opinion of viewers.

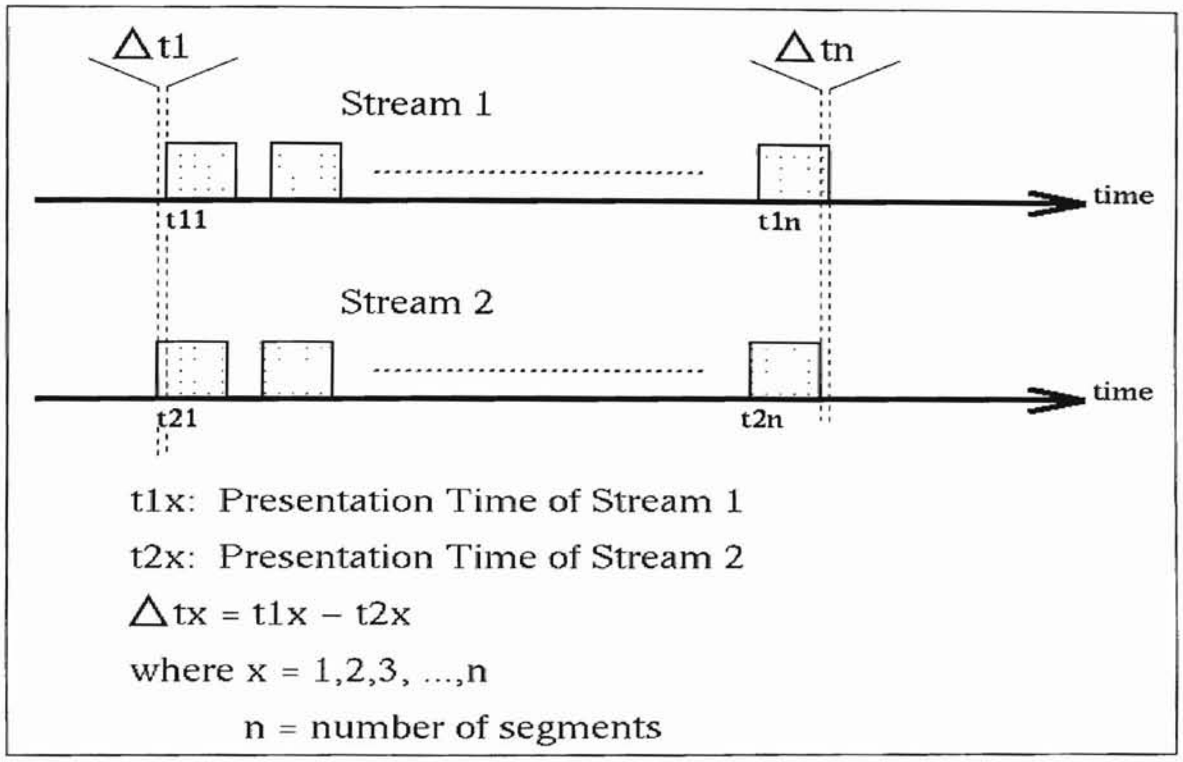

Figure 5.1 Presentation Time of Two Multimedia Data Streams

As we mentioned earlier, multimedia data streams are segmented and given timestamps by server processes. Client processes reassemble and present them after receiving the streams. Figure 5.1 is a picture of presentation time line of two multimedia streams. Each segment is expected to be presented at their designated time. The different presentation times between the audio and video segments can be denoted as Δtx, where x = 1,2,...,n, and n is the number of video and video segments. If audio and video streams start and end at the very same time, then $\Delta t1 = \Delta t2 = \Delta t3 = ... = \Delta tn = 0$. This is rated as a good skew. A moderate skew is rated when Δtx is less than 1 sec. It means that either audio or video streams might be started and/or ended at most 1 second earlier than another. A poor skew is rated when Δtx is greater than 1 sec. An unknown skew is rated when multimedia files have only one data stream, either audio or video stream.

Object movements in video stream play important role in compressed video quality. Losing some image sequences in multimedia files that have fast moving objects might not generate smooth object movements. The pixel areas, where the object movements are captured, are the most significant parts generating the level of video quality. Some of object movements' information might be kept in B and P frames. Losing these frames might cause distortion on these pixel areas. A good video quality is rated when no distortion occurred in any parts of image sequences. A moderate video quality is rated when distortion occasionally occurred in some part of image sequences. In other words, object movements are rarely captured. However, if distortion occurs in multimedia files, which object movements are frequently captured and the pixel areas of object movements are relatively small portion compared to their frames' size, their video qualities still can be considered as moderate. A poor video quality rating is given when frequent distortion occurred in most of the image sequences. The rating of skew and video quality parameters can be described in the following table:

Table 5.2 Rating of Some QOS Parameters

Rating	Skew Parameter	Video Quality
Good	All presentation time of Audio and video segments are simultaneous.	No distortion occurred in any parts of image sequences.
Moderate	All presentation time of audio and video segments differ from one another by at most 1 second.	Distortion occasionally occurred in some of the image sequences.
Poor	Presentation time of audio and video segments differ from one another by more than 1 second.	Frequent distortion occurred in most of image sequences.
Unknown	Multimedia files contain only one multimedia data stream.	Multimedia files contain only audio stream.

5.1 Experiments

Five multimedia files were used to test the synchronization scheme. For each multimedia file, (if applicable) we had conducted three simulations which were as follows:

- First simulation was set up such that a client requested a multimedia file from server with all I, B, and P frames being sent.
- The second simulation was set up such that a client requested a multimedia file from server with I and P frames being sent.
- The third simulation was set up such that a client requested a multimedia file from server with I frames being sent.

The first multimedia file was about self-introduction of three researchers from Multimedia Communication Labs at Boston University. During the presentation, they rarely moved, and the background of the scene was a still object. Hence, just a few movements were captured in this file. The file had both video and audio streams. The video stream was encoded in IPB frame configuration. The frame size was 80x64 pixels area. The result of measured parameters are tabulated in table 5.3.

Table 5.3 Result of Measurement on Multimedia File Containing Relatively Still Objects.

Simulation	Number of I frames	Number of P frames	Number of B frames	frames/sec	skew	video quality
I	741	741	2222	60	Moderate	Moderate
II	741	741	0	24	Moderate	Moderate
III	741	0	0	12	Moderate	Moderate

The second multimedia file was about a person speaking in German language. During the presentation, the person occasionally moved, and the background of the scene was launching of a rocket. Hence, a lot of movements were captured in this file. The file had both video and audio streams. The video stream was encoded in IPB frame configuration. The frame size was 320x240 pixels area. The result of measured parameters are tabulated in table 5.4.

Table 5.4 Result of Measurement on Multimedia File Containing Relatively Moderate Moving Objects.

Simulation	Number of I frames	Number of P frames	Number of B frames	frames/sec	skew	video quality
I	41	80	359	25	Moderate	Moderate
II	41	80	0	10	Moderate	Poor
III	41	0	0	5	Moderate	Poor

The third multimedia file was about an animation from IRIX Inc. The animation contained moving and resizing letters. The background of the scene was a black (frame) object. Hence, relatively a lot of movements were captured in this file. The file had both video and audio streams. The video stream was encoded in I frame configuration. The frame size was 160x120 pixels area. The result of measured parameters are tabulated in table 5.5.

Table 5.5 Result of Measurement on Multimedia File Containing Relatively Moderate Moving Animation.

Simulation	Number of I frames	Number of P frames	Number of B frames	frames/sec	skew	video quality
I	901	80	359	25	Moderate	Moderate

The fourth multimedia file was a basket ball game. Many objects were captured in this file. The objects were basket ball, players, referees, and observers. Basket ball and players were the most focused objects, and were moving frequently. Hence, a lot movements were captured in this file. The file had only video stream, which was encoded in IPB frame configuration. The frame size was 320x240 pixels area. The result of measured parameters are tabulated in table 5.6.

Table 5.6 Result of Measurement on Multimedia File Containing Moving Objects.

Simulation	Number of I frames	Number of P frames	Number of B frames	frames/sec	skew	video quality
I	786	2355	6278	30	Unknown	Moderate
II	786	2355	0	12	Unknown	Poor
III	786	0	0	6	Unknown	Poor

The fifth multimedia file was a scene of a bus circling a park. A rotating camera was placed at the center of the park. During the presentation, background was changing frequently. Hence, it seemed that we had a still object (bus) with a lot of movements in the background. The file had only video stream, which was encoded in IPB frame configuration. The frame size was 352x240 pixels area. The result of measured parameters are tabulated in table 5.7.

Table 5.7 Result of Measurement on Multimedia File Containing A Moving Object.

Simulation	Number of I frames	Number of P frames	Number of B frames	frames/sec	skew	video quality
I	786	2355	6278	30	Unknown	Moderate
II	786	2355	0	12	Unknown	Poor
III	786	0	0	6	Unknown	Poor

5.2 Discussion

There are three factors determining skew parameters of multimedia files. They are encoding (sampling) process of audio and video streams, segmenting and assigning timestamps on multimedia data streams, and adjusting frame rates of video streams. Multimedia files used in this study are pre-sampled. Audio files are sampled at 44.1 kHz. This audio sampling rate is sufficient enough since it generates about 44000 samples per second. While video files are sampled in varying rate. The MPEG-video standard suggests 30 frames per second to be used. Higher sampling rates on video stream give moderate skew parameter (refer to tables 5.3, 5.4, and 5.5).

Segmenting and assigning timestamps on multimedia data streams is one of the factors that skew parameter depends on. In this study, video and audio streams are segmented based on I frames found in video stream. When B and/or P frames are discarded, audio segments associated with them are either assigned to be associated to the I frame preceding or following the discarded frames. This might generate inappropriate segmenting and assigning timestamps on multimedia data streams. Furthermore, it might not generate good skew parameter.

Another factor in determining skew parameter is frame rate of video stream. When B and/or P frames are discarded from original multimedia files, the frame rates of video stream need to be adjusted. This adjustment is calculated as frame rate of original video stream times the ratio of total frame number of modified video stream over total frame number of original video stream. The experiment shows that multimedia files that originally sampled at higher sampling rate have moderate skew parameter in simulations which are discarding B and/or P frames (see tables 5.3 and 5.4).

Discarding some sequence of images and lowering rate of frame sampling might cause poor video quality. When some sequences of images are discarded, the frame rate of the video stream needs to be adjusted in order to have synchronized presentation of audio and video streams. Choking effect is the impact of discarding some sequences of images, and it decreases quality of video streams. The appearance of this effect depends on the sampling rates of original video streams. If original video streams are sampled at higher rates, the choking effect may not be recognized during presentation of video streams. As an example, the original first multimedia file had sampling rate at 60 frames per second (fps). When B frames were discarded, the frame rate was adjusted to 24 fps, which was close to the sampling rate of MPEG-Video standard (30 fps). Its video quality was rated as moderate.

CHAPTER 6

CONCLUSION

6.1 Summary

In this thesis work, we have described a method of synchronization which discards parts of a compressed video stream at the sender site. This synchronization scheme also implements relative synchronization, and application-oriented and pessimistic buffering strategy.

An advantage of this synchronization is that flow data rate sent can be adjusted by client according to the network traffic. However, some disadvantages occur which are as follows:

- Choking effect may appear when the multimedia data are played back, since B and/or P frames might be discarded when it is necessary.
- A TCP connection must be made up before transactions begins. This connection might not be reliable when multimedia source and multimedia destination are far apart. If network congestion problem often occurs, a choking effect may appear frequently.

Experiments suggest that original video streams need to be sampled at higher sampling rate in order to reduce choking effect. This suggestion does not seem to be encouraging since having higher sampling rate on video streams requires large amount of storage to store compressed video stream.

6.2 Future Work

Following are suggested as future work:

- *UDP connection*: to have more reliable connection, we need to change the type of connection to be a UDP connection. Using this type of connection we might be experiencing less "choking" effect. However, a major modification of client and server codes need to be done, since we have to provide a mechanism that makes sure that all video and audio segments are well received, and arrange them in the correct order.
- *Self-adjustment Mechanism:* implement a self-adjustment flow mechanism such that flow data rate can be adjusted by both client and server dynamically by evaluating current network traffic without the user's intervention.

BIBLIOGRAPHY

[AS94] Frank Adelstein and Mukesh Singal. "Priority Ethernet". *In Proceedings of the IASTED/ISMM: Distributed Multimedia Systems and Applications*, pages 45-48, Honolulu, Hawaii, August 1994.

[Bad94] Tobias Bading. "MPEG Audio Player MAPLAY 1.2". *Archived software ftp://ftp.crs4.it/mpeg/programs/maplay1_2.tar.Z.* Berlin University of Technology, June 1994.

[CLSWCO95] Herng-Yow Chen, Nien-Bao Liu, Chee-Wen Shiah, Ja-Ling Wu, Wen-Chin Chen, and Ming Ouhyoung. "A Novel Multimedia Synchronization Model and Its Applications in Multimedia Systems". Communication and Multimedia Laboratory, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.

[Cri93] S.M. Crimmings. "Analysis of Video Conferencing on A Token Ring Local Area Network". *In Proceedings ACM Multimedia 93*, pages 301-310, ACM Press, New York, 1993.

[DGJJ94] D. Davcev, S. Gievska, S. Jordanoski, and Lj Josifovski. "Real-Time Multimedia Tele-Teaching System". *In Proceedings of the IASTED/ISMM: Distributed Multimedia Systems and Applications*, pages 131-134, Honolulu, Hawaii, August 1994.

[DPBF 96] Martin Dietz, Harald Popp, Karlheinz Brandenburg, and Robert Friedrich. "Audio Compression for Network Transmission". *Journal of the Audio Engineering Society,* vol. 44, pages: 58-60, January/February 1996.

[Fur94] Borko Furht." Multimedia Systems: An Overview". *IEEE Multimedia*, 1 (1): 47-49, Spring 1994.

[Gal91] Didier Le Gall. "MPEG: A Video Compression Standard for Multimedia Applications". *Communications of the ACM*, 34 (4) :47-58, April 1991.

[Hung91] Andy C. Hung, "PVRG-MPEG Codec", Archived file MPEGv1.1.tar.Z, Portable Video Research Group, Standford University, 1991.

[JSN95] Spaul W. Jardetzky, Cormac J. Sreenan, Roger M. Needham. "Storage and Synchronization for Distributed Continuos Media". *Multimedia Systems*, vol. 3, pages 151-161, 1995.

[KK94] Cheeha Kim and Sang Wook Kang. "A Media Synchronization Scheme for Distributed Multimedia Systems". *In Proceedings of the IASTED/ISMM: Distributed Multimedia Systems and Applications*, pages 163-166, Honolulu, Hawaii, August 1994.

[Loug95] Phillip Lougher. "mpegUtil". *Archived software mpegUtil.tar.gz.* Computing Department, Lancaster University, Lancaster, LA1 4YR, United Kingdom, March 1995.

[OWCLL95] Ye-Jen Oyang, Chun-HungWen, Chih-Yuan Cheng, Meng-Huang Lee and Jian-Tian Li. *IEEE Transactions on Computer Electronics*, 41 (1), February 1995.

[Pan95] Davis Pan. "A Tutorial on MPEG/Audio Compression". *IEEE Multimedia*, 1(1): 47-49, Summer 1995

[PSR93] Ketan Patel, Brian C. Smith, and Lawrence A. Rowe. "Performance of a Software MPEG Video Decoder". *Proceedings ACM Multimedia 93*, Anaheim, California, August 1993.

[Rav93] K Ravindran. "Transport Models for Synchronization of Multimedia Data Streams". *Technical Report TR:93-8*, Department of Computing and Information Science, Kansas State University, January 1993.

[RPSGH95] Lawrence A. Rowe, Ketan Patel, Brian Smith, Kevin Gong, Eugene Hung, Steve Smoot, Doug Banks, Sam Tze-San Fung, Darryl Brown, and Dan Wallach. "Berkeley MPEG Tools". *Archived software ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/mpeg/bmt1r1.tar.gz*. Computer Science Division-EECS University of California at Berkeley, February 1995.

[SRD93] Ke Shen, Lawrance A. Rowe, and Edward J. Delp. "A Parallel Implementation of an MPEG1 Encoder: Faster Than Real-Time". *Technical Report MM93*, University of California at Berkeley, 1993.

[SS90] D. Sheperd and M. Salmony. "Extending OSI to Support Synchronization Required by Multimedia Applications". *Computer Communication*, 7(13), page: 399-406, 1990.

[VKBG95] Andreas Vogel, Brigitte Kerherve, Gregor von Bochman, and Jan Gescei. "Distributed Multimedia and QOS: A Survey". *IEEE Multimedia*, 1 (1): 10-17, Summer 1995

[VLP95] Rolf Velthuys, Kelly Lyons, and Ian Parsons. "Multireception Service for a Multimedia News Application". *Fourth International Conference on Computer Communications and Networks (ICCCN'95)*. IEEE Computer Society Press, Los Alamitos, California, September 1995.

APPENDIXES

APPENDIX A

DISCRETE COUSINE TRANSFORM ON IMAGE COMPRESSION

The following paragraphs are taken verbatim from PVRG-MPEG Codec [Hung91]:

"For each separate color component, the image is broken into 8 x 8 blocks that cover the entire image. These blocks are the input to the DCT. Typically, in the 8 x 8 blocks, the pixel values vary slowly. Therefore, the energy is of low spatial frequency. A transform that can be used to concentrate the energy into a few coefficients is the two-dimensional, 8 x 8 DCT which is extremely efficient for highly correlated data. Conceptually, a one-dimensional DCT can be thought of as taking the Fourier transform and retaining only the real (cosine) part. The two-dimensional DCT can be obtained by performing a one-dimensional DCT on the columns and then, a one-dimensional DCT on the rows. Formula for the two dimensional 8 by 8 DCT can be written in terms of the pixel values *f(i,j)*, and the frequency domain transform cofficients,

$$F(u,v) = (1/4)C(u)C(v)\sum_{i=0}^{7}\sum_{j=0}^{7} f(i,j)\cos((2i+1)u\pi/16)\cos((2j+i)v\pi/16)$$

where

$$C(x) = \begin{cases} 1/\sqrt{2} & x = 0 \\ 1 & otherwise \end{cases}$$

The transformed output from the two-dimensional DCT is ordered so that the mean value (the DC coefficient) is in the upper left corner of the 8 x 8 coefficient block, and the higher frequency coefficients progress by distance from the DC coefficient. Higher vertical frequencies are represented by higher row numbers, and higher horizontal frequencies are represented by higher column numbers.

The inverse of the two-dimensional DCT is written as

$$f(i,j) = (1/4)\sum\sum C(u)C(v)F(u,v)\cos((2i+1)u\pi/16)\cos((2j+1)v\pi/16)$$

".

APPENDIX B

QUANTIZATION ON IMAGE COMPRESSION

The following paragraphs are taken verbatim from PVRG-MPEG Codec [Hung91]:

"The coefficients of the DCT are quantized to reduce their magnitude and to increase the number of zero value coefficients. The uniform quantizer is used for the MPEG method, with a different stepsize per DCT coefficient position.

The intraframe blocks are quantized with DC and the AC terms separately; the AC and DC quantization are

$$C(0,0) = \lfloor (F(0,0) \pm 4) / 8 \rfloor$$

$$A(u,v) = \lfloor ((F(u,v) * 16) \pm Q(u,v) / 2) / Q(u,v) \rfloor$$

$$C(u,v) = \lfloor ((A(u,v) \pm Q_F) / 2Q_F \rfloor$$

where *C(u,v)* is the quantized coefficient, *F(u,v)* is the DCT frequency coefficient, *Q(u,v)* is the quantizer stepsize, Q_F is the quantizing parameter (for rate control); and $\pm$ is positive for *F(u,v)* positive, negative otherwise.

The inverse intra quantize is

$$F(0,0) = 8C(0,0)$$

$$F(u,v) = C(u,v) Q_F Q(u,v) / 8$$

For forward predicted and interpolated blocks, the qunatisizer has a dead-band around zero, and is the same for both AC and DC components as

$$A(u,v) = \lfloor((F9u,v)*16) \pm Q(u,v) / 2Q(u,v)\rfloor$$

Then if Q_F is odd

$$C(u,v) = A(u,v) / 2Q_F$$

otherwise

$$C(u,v) = (A(u,v) \pm 1) / 2Q_F)$$

where $\pm$ is positive for *A(u,v)*>0, otherwise is negative.

The inverse quantizer is then

$$F(u,v) = (2F(u,v) \pm 1)Q_F Q(u,v) / 16)$$

where $\pm$ is positive for *F(u,v)>0,* negative otherwise.

Quantization is the lossy stage in the MPEG coding scheme. If we quantize too coarse, we may end up with images that look "blocky", but if we quantize too fine, we may spend useless bits coding.".

APPENDIX C

CODING MODEL AND ENTROPY CODING

The following paragraphs are taken verbatim from PVRG-MPEG Codec [Hung91]:

"The coding model rearranges the quantisized DCT coefficients into a zig-zag pattern, with the lowest frequencies first and the highest frequencies last. The zig-zag pattern is used to increase the run-length of zero coefficients found in the block. The assumption is that the lowest frequencies tend to have larger coefficients and the highest frequencies predominantly by zero.

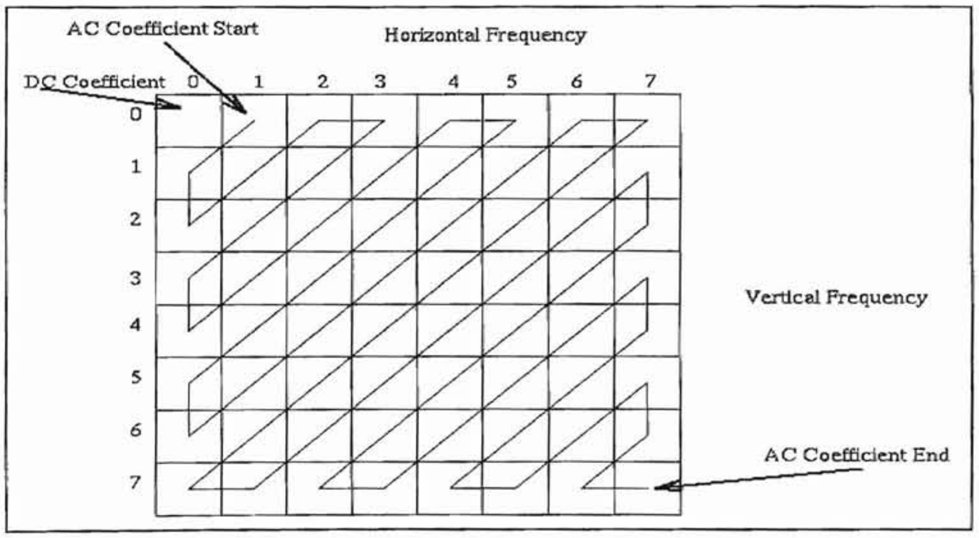

Figure C.1 zig-zag pattern [Hung91]

The first coefficient (0,0) is called the DC coefficient, and the rest of coefficients are called AC coefficients (see figure C.1). The AC coefficients are traversed by the zig-zag pattern from the (0,1) location to the (7,7) location.

The quantisized DC coefficients are encoded by the number of significant bits, followed by the bits themselves. The quantisized AC coefficients usually contain runs of consecutive zeros. Therefore, a coding advantage can be obtained by using a run-length technique. The AC coefficients are encoded based on the number of zeroes before the next non-zero coefficient. For frequently occurring combinations of zero-run-length/nonzero-coefficient, a unique variable length code is used. For the other codes, an ESCAPE variable length code allows the definition of run-length of zeroes, and the level of the coefficients, as is. The inverse of run-length coder translates the input coded stream into an output array if AC coefficients. Based on the run-length code, it takes the current position in the output array and appends a number of zeroes followed by the next non-zero coefficient.

The block codes from run-length models can be further compressed using entropy coding. For the MPEG method, the Huffman coder is used to compress the data closer to symbol entropy. To compress data symbols, the Huffman coder creates shorter codes for frequently occurring symbols and longer codes for occasionally occurring symbols. The first step in creating Huffman codes is to create a table assigning a frequency count to each symbol. Then, initially designate all symbols as leaf nodes for a tree. Starting from two the two least weight nodes, aggregate the pair into a new node. Repeat this process for the new set until the entire symbol set is represented by a single node. Attach a binary digit to each branch and assign a 0 and 1 to left and right branches. The symbol code is generated by following the path of branches from the top node to the symbol leaf node.".

VITA

Aftab Achmad Lubis

Candidate for the Degree of

Master of Science

Thesis: A SYNCHRONIZATION SCHEME FOR DELIVERING MULTIMEDIA DATA STREAMS.

Major Field: Computer Science

Biographical:

Personal Data: Born in Jakarta, Indonesia, On May 21, 1965, the son of Syarif Ahmad Lubis and Organi Semiarti Siregar.

Education: Received Insinyur Fisika Teknik from Institut Teknologi Bandung, Indonesia in October 1988. Completed the requirements for the Master of Science degree with major in Computer Science at Oklahoma State University in May 1997.

Experience: Software Quality Controller at Computing Center, Nusantara Aircraft Industries, Indonesia from June 1989 to July 1992.