

**IMPLEMENTATION OF VIDEOCONFERENCING SOFTWARE
FINAL REPORT**

ECE 496Y - Design Project
Department of Electrical and Computer Engineering

Submitted By

Wong Chujoy, Jaime - 940377330
Trinh, Shing - 941464590
Ng, Harfrey - 951547410
(Group B)

Submitted To

Professor Irene Katzela

University of Toronto
Faculty of Applied Science and Engineering

April 9, 1999

EXECUTIVE SUMMARY

This report is intended to describe the design and implementation of the Video Conferencing Software project. The objective of the project is to implement a bi-directional video conferencing tool. The project consists of designing and implementing software to encode, and send video samples to another system where it is decoded and played back in real time. The successful completion of the project will produce software that will enable two users located at different terminals to communicate with each other through video over the Internet. The project commenced in September 1998 and terminated in April 1999.

TEAM MEMBER CONTRIBUTIONS

Jaime Wong Chujoy

The implementation of the videoconferencing tool was made possible by the combined efforts of my team members and myself. However, the project was divided into three major groups: Communication protocol, Video encoding and display, and High level integration of parts. My contribution to the project consisted of researching, designing and implementing the video compression and decompression utility MPEG-1

My first task to the development of MPEG-1 consisted of researching video compression technology to become familiar with the features and functionality of MPEG-1. This included studying the different system calls by MPEG-1 to the X-Image Library (XIL). In order to assist the design of the video section of the project, I was responsible in analyzing the previous project by Yasseer Rasheed. This consisted of dissecting and studying the components of Rasheed's software. My contribution to the video section consisted of creating a high-level design for the initialization, compression, and decompression of MPEG-1. Once my design was finished and assured to be at optimum, I was responsible for implementing the design by programming using the "C" language. The decompression portion of MPEG-1 was developed along with Harfrey Ng. In addition, we encountered serious problems with the Solaris operating system. Along with Wing-Chung and Harfrey Ng, I was involved in solving problems with the XIL Library.

For the preparation of this final report, my duty was to describe *the "Methods and Materials or Design"* with regards to MPEG-1 video (chapter 2.4). I was responsible for composing the *Cover Page*, *Executive Summary*, *Table of Contents*, and *Introduction* (chapter 2.1). In addition, I gathered all the written work from the team member's compiled into this final report (this included editing Harfrey Ng's written work).

Shing Trinh

We all came up with the system design through several meetings between ourselves and with our supervisor. Before I started coding I reviewed the previous videoconferencing project done at the Network Architecture Lab. I also looked at the example programs provided with the XIL library. From this I wrote a minimal decoder program that would read in an MPEG-1 file, decode it, and display it in a window. After examining the program I divided up the program into general functions that encapsulated the initialization and display of video in a window. These functions were used by Harfrey to add the ability to view the video being encoded in the encoder application written by Jaime. I modified our decoder to read from a socket instead of a file and changed Jaime and Harfrey's encoder to write to a socket also. At this point we had a unidirectional video link. I rewrote the separate encoder and decoder programs as a series of modular functions that could be combined into a single program. The encoder and decoder were rewritten in such a way that both could run concurrently in their own threads. Finally the encoder and decoder functions were both combined into the same program creating a fully bi-directional video link. This involved writing code to initialize, run and, synchronize the two threads.

For the final report I wrote High Level Design (chapter 2.1), Video Capture and Display (chapter 2.2), the Suggestions for Future Work of the Conclusion (chapter 3.1), Code Module Descriptions (Appendix A), and the Manual Page in (Appendix B). I along with Jaime Wong and Harfrey Ng were also responsible putting together the final report and editing it.

Harfrey Ng

I was responsible for implementing the MPEG-1 Decoding System, Playback System and the User Display. In this Final Report, I was responsible for writing the topics of Implementation of Decode and Display, Problem Encountered, High Level Design of Video Conferencing Tool and Conclusion of Final Report.

During the first term, my members and I met regularly with Professor Irene Katzela and came up with the verified design of the Video Conferencing tool. Also, I examined the previous Video Conferencing source code and wrote down a synopsis and a description of the useful system functions in a document to serve as my personal summary and four group's reference.

During the implementation of our project, there were problems from the XIL 1.3 library and SUN Capturing Device Driver in Solaris 2.6 system. I worked together with the NAL UNIX System Administrator, Wing-Chung Hung, to solve these system problems. Solving these problems have benefited not only our project, but also other projects at NAL that require XIL 1.3 library and the Capturing Device.

Regarding the contribution related to the implementation, besides the responsibilities described in the Interim Report; I have worked on some additional tasks. In the Interim Report, I wrote that I would implement the video rendering system that receives and playbacks the decoded video frames and would implement the User Interface. Since the rendering system is closely connected to the MPEG-1 Decoding System, I become also responsible for implementing the MPEG-1 Decoding System along with Jaime Wong. In conclusion, my contributions to the project are to implement the MPEG-1 Decoding System, the Playback System, the User Display, and to solve System Problems. I also integrate these components with Jaime Wong's components, so that we can decode and display the captured images at local workstation.

Based on my experience in the implementation of the Video Conferencing Tool, I will write the topics of Implementation of Decode and Display (chapter 2.2, 2.3), System Problem Encountered, High Level Design of Video Conferencing Tool (chapter 2.1), and Conclusion (chapter 3) in this Final Report.

MILESTONES

Problems Encountered

Based on the new timeline, the implementation of MPEG-1 retrieval/encoding, MPEG decoding/playback and User Display were supposed to take place concurrently in January. However, delays on the system occurred due to recent upgrades at NAL. The upgrades from Solaris 2.5 to Solaris 2.6 and the image library from XIL 1.2 to XIL 1.3 caused compilation and linking problems that prohibited us from compiling the Video Conferencing Tool. As a result, we spent approximately three weeks solving these system problems with the valuable help from the UNIX administrator, Hung Wing Chung. In effect, the schedule of our timeline was postponed for about 3 weeks.

There were two major system problems with the XIL and video capturing. At that time, two versions of XIL were installed in two different directories. Specifically, XIL 1.3 was installed in the operating system while the previous XIL 1.2 was still present. As a result, the operating system had the tendency to link with the old XIL 1.2 library when programs were compiled. To solve this problem, both XIL 1.2 and previous copies of 1.3 were uninstalled, and a new copy of XIL 1.3 was installed. For the video capturing problem, the images captured by the Sun Video Device could not be displayed on the screen. The problem was fixed by reinstalling the Sun Video Device Driver.

MILESTONES (September 1998)

Activity	October			November		January		February		March	
	Week 06	Week 12	Week 19	Week 02	Week 16	Week 04	Week 11	Week 01	Week 08	Week 08	Week 22
Background	6 days										
Examine Prev. Project	7 days										
Update Prev. Project			14 days**								
Requirement/ Specifications				14 days**							
Initial Design					21 days						
Verify Design						7 days					
Design Revision							7 days				
Implementation											
-MPEG retrieval/encoding							14 days**				
-RTP encoding/decoding											
-UDP/IP transport								28 days**			
-MPEG decoding/playback											
-GUI									7 days		
Testing and Debugging										14 days	
Performance Testing											7 days
Project Dead											

** Midterm season - delays may be expected

- Written Reports and Oral Presentations not included in this chart.


- Delays may be expected around


Oct. 27	Written Proposal Due
Jan. 05	Interim Report Due
Mar.	Design Fair
Apr. 09	Final Report

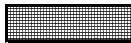
- No or little work anticipated during Christmas Break

REVISED MILESTONES

Activity	October			November		January		February		March	
	Week 06	Week 12	Week 19	Week 02	Week 16	Week 04	Week 11	Week 01	Week 08	Week 08	Week 22
Backgroun	6 days										
Examine Prev. Project	7 days										
Update Prev. Project			14 days**								
Requierment/ Specifications				14 days**							
Initial Design					21 days						
Verify Design						7 days					
Design Revision							7 days				
Implementation											
-MPEG retrieval/encoding							14 days**				
-MPEG decoding/playback											
-GUI											
-Test for MPEG playback									7 days		
-UDP/IP transport								28 days **			
-RTP encoding/decoding											
Testing and Debugging										14 days	
Performance Testing											7 days
Project Deadline											

 represents completed milestones

 represents uncompleted milestones

 represents cancelled milestones

** Midterm season - delays may be expected

- Written Reports and Oral Presentations not included in this chart.

- Delays may be expected around:

Oct. 27 Written Proposal Due

Jan. 05 Interim Report Due

Mar. Design Fair

Apr. 09 Final Report

- No or little work anticipated during Christmas Break

TABLE OF CONTENTS

♦ **Executive Summary**

♦ **Team Member Contributions**

- Jaime Wong Chujoy
- Shing Trinh
- Harfrey Ng

♦ **Milestones**

- Explanation for Delays
- Old Milestones
- Revised Milestones

♦ **Table of Contents**

♦ **Acknowledgements**

♦ **Chapter 1: Introduction**

- 1.1 Background Information
- 1.2 Motivation
- 1.3 Project Objective
- 1.4 Report Outline

♦ **Chapter 2: Methods and Materials or Design**

- 2.1 High Level Design
 - 2.1.1 Sender Design
 - 2.1.2 Receiver
 - 2.1.3 Sender-Receiver Synchronization
- 2.2 Video Capture and Display Implementation
- 2.3 Video Display
- 2.4 MPEG-1 Video
 - 2.4.1 MPEG-1 Implementation
 - 2.4.2 MPEG-1 Initialization
 - 2.4.3 MPEG-1 Transmission
 - 2.4.4 MPEG-1 Reception

♦ **Chapter3: Conclusion**

3.1 Concluding Remarks

3.2 Suggestions for Future Work

♦ **References**

♦ **Appendices**

- Appendix A: Module Descriptions
- Appendix B: Manual Page
- Appendix C: Source Code Listing

ACKNOWLEDGEMENTS

We would like to thank our supervisor Professor Irene Katzela for her guidance, commitment, friendly advice, and encouragement throughout the course of our design project. Her help has unquestionably allowed us to complete the project successfully.

We would also like to extend special thanks to Daniel Lopez for all his help. His time and effort put into ensuring our project ran as smoothly as possible was much appreciated. In addition, we like to express gratitude to Davy (Wing-Chung) Hung for his continuous technical assistance in the Networks Architecture Lab.

Chapter 1: INTRODUCTION

1.1 Background Information

Video Conferencing

Videoconferencing is the live communication of two or more people using a combination of video, audio, and data exchange. Over the past few years there has been an increase in demand for distant communications. Videoconferencing has found many applications in such diverse fields as personal communications, collaborative work, presentations, research, distance education, remote surveillance, and entertainment.

The implementation of a videoconferencing tool has some inherent difficulties due to the large amounts of data produced when video and audio are digitally produced. The vast amount of data causes the video CoDec (Compressor/ Decompressor), and data network connecting the computers to slow down. The delay in the CoDec and network will produce unwanted delays and de-synchronization between video frames. There are many ways to decrease the size of video and audio data. One of the best ways is by compression using the popular standards MPEG. However, the compressed information is still large enough to cause delays in the network. As a result, a protocol that allows real-time communication is needed.

User Datagram Packet (UDP)

UDP is a protocol layered on top of IP (Internet Protocol), the base protocol for transferring data over the Internet. UDP is a connectionless, unreliable, datagram protocol used to send data over IP networks such as local area networks or the Internet. When a UDP packet is sent there is no guarantee that it will reach its destination. In addition, there is no guarantee that packets are received in the same order they were sent, since packets may take different routes to its destination. Furthermore, duplication of packets is possible. However, if the packet does reach its destination it is guaranteed to be error free.

Despite the problems mentioned, UDP is used as the preferred protocol in applications that require speed but not necessarily require reliability. Non-critical means that speed and throughput are not crucial due to low overhead. Since UDP does not provide the acknowledgements, retransmission, and ordering features of TCP (Transmission Control Protocol), it provides a much faster end-to-end transmission time, and makes a more efficient use of the network bandwidth. UDP by itself is not suitable for real-time applications such as videoconferencing because it provides no facilities for synchronization, as well as provision of real-time delay constraints. What is needed is an additional protocol, layered on top of UDP to provide the additional real-time functionalities.

Real-Time Transport Protocol (RTP)

The Real Time Transport Protocol, introduced in 1996, provides delivery services for data with real-time characteristics, such as interactive audio and video. It provides procedures for timing reconstruction, loss detection, security, and content identification. RTP is designed to be independent of the underlying transport and network layers protocols.

The RTP protocol is essential for interactive multimedia applications, like videoconferencing, due to its real-time capabilities. There are a couple of reasons for this. Consider first the situation where IP packets are routed across a network, where individual packet transit times can vary significantly. In order to handle the variations in transit time delay, referred as “jitter”, a timestamp in the RTP header could be used for delay variation calculations. If a receiver detects, from the timestamp field of the RTP header, that the transit time for an RTP packet has an abnormally long time, the receiver may decide to discard the packet rather than to keep the packet and distort the audio/video information later on. RTP also records the sequence number for each packet sent so a receiver can detect lost packets, preserve sequence and request retransmission of the corrupted or lost packets.

In congested network condition the effect on non-real-time traffic is that the transfer takes longer to complete if RTP protocol is used. In contrast, real-time data becomes obsolete if it does not arrive within

certain time interval. As a result, the timestamp and sequence number fields become essential in alleviating the problem.

RTP has reached widespread acceptance, especially in network multimedia applications, because of its suitability to work with real-time traffic. It has found use in many large commercial applications such as Netscape's LiveMedia and Microsoft's NetMeeting conferencing software.

1.2 Motivation

A videoconferencing system has already been implemented at NAL by a current Ph.D. student Yasseer Rasheed as part of his Masters thesis. The system consists of a unidirectional link between two computers over IP and ATM connections. Images from a video capture board and camera are digitized, compressed using the MPEG-1 video standard, and transmitted over an Ethernet network through IP or ATM. The system uses a custom proprietary protocol layered on top of IP and ATM to accomplish real-time video transfers.

The system was originally implemented using Solaris 2.5 and the XIL 1.2. However, the operating software and its associated libraries have been upgraded to Solaris 2.6 and XIL 1.3. The upgrade has caused a number of incompatibilities between the unidirectional video conferencing tool and the system libraries and as a result the program no longer runs or compiles under the new operating system.

1.3 Project Objective

The purpose of the project is to implement a bi-directional videoconferencing tool. The project consists of designing and implementing software to encode, and send video samples from a video capture board to another system where it is decoded and played back in real time. The system will use the MPEG-1 video standard for compression and decompression. For communication purposes, we will be using the Real Time Protocol (RTP) together with the User Datagram Protocol / Internet Protocol (UDP/IP) standards.

As a first step, study and research of materials that are relevant to the project will be performed. Technologies such as video compression utilities, real-time protocols, and the Solaris operating system will be studied and familiarized. Next, to help accomplish and accelerate the project's objective, the development of this project will take advantage of the research done previously at NAL. This would facilitate the design of a new system capable of bi-directional communications using MPEG video compression.

Initially, a plan was arranged to update the previous project by Rasheed to help familiarize with video conferencing designs. However, the update of Rasheed's project will not be accomplished mainly due to time constraints and also due to the relative short amount of learning experience.

The design will be accomplished by dividing the project into three main categories, which each team member is responsible for design and implementing. The first member, *Jaime Wong Chujoy*, will be responsible for the video compression/decompression of MPEG-1. The second member, *Shing Trinh*, will be responsible for the communication protocols and synchronization. Last, *Harfrey Ng* will be responsible for the display and help implement the decompression of MPEG-1.

After the design is reviewed and assured to be correct, the software will be implemented using the "C" language and the XIL library. Once the implementation and debugging is finished, additional testing and debugging will be required. In addition, some performance testing will be done to optimize the execution of the software.

1.4 Report Outline

In September 1998, a proposal was written to organize the development of the project. In addition, three interim reports were written in January 1999 to present the progress of the project at that

time. This report will describe the final design, implementation, and results of developing the video conferencing software.

In the following sections, the high-level design and solution to the present challenges will be presented. Furthermore, the section will describe the design of the communication protocols, video encoding/decoding, and display. The conclusion section will describe our achievements and results of our design and implementation. Finally, the appendices will present the module description, a user's manual on how to operate the software, and the source code.

2.1 High Level Design

In order for two computers to participate in a videoconference the videoconferencing software must be running on both computers (see figure 1). The videoconferencing software requires some sort of video capture device to be present. Data captured on one computer is encoded, packaged, and sent over the Internet to the other computer where it is decoded and played back through the video card; this must happen in both directions simultaneously.

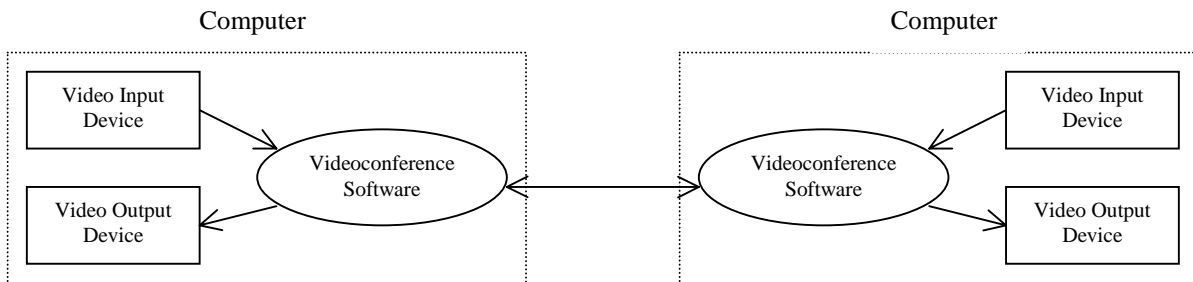


Figure 1. Video Conferencing System Level 0 Dataflow Diagram

The videoconferencing system consists of two separate subsystems: a sender and a receiver (see figure 2). The sender is responsible for acquiring video data and sending it to the receiver at the other end of the connection. The receiver is responsible for receiving the video data from sender and outputting the data.

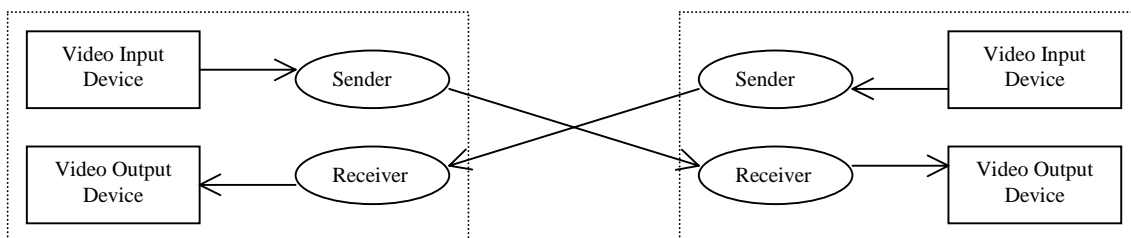


Figure 2. Video Conferencing System Level 1 Dataflow Diagram

2.1.1 Sender Design

Sending a video stream involves four stages (see figure 3):

1. Recording and Timestamping
2. Image Preview
3. MPEG compression
4. RTP Encoding and Delivery

The recording and timestamping stage is responsible for obtaining video samples. This involves sampling frames from a video capture device. Dividing the video into frames allows for the data to be sent in discrete packets and avoids overflowing the network with a continuous stream of data. At the time the sample is taken, a timestamp must be added to the clip before passing it down the pipeline to retain its timing information.

The image preview stage displays the captured image in a window. Allowing the sender to preview the video being sent over the network so that he/she can make changes to camera position.

The MPEG compression stage compresses the raw video frame into MPEG-1 encoded frames. The timestamp provides the timing information required to string the MPEG frames together to recover the original video stream.

The RTP encoding stage adds an RTP header to the MPEG data that encodes the timing information. A sequence number is also included in the RTP header that will allow the receiver to detect lost frames. After an RTP header is added to the data, it is sent to the other client over the Internet via UDP/IP and the computer's network interface card.

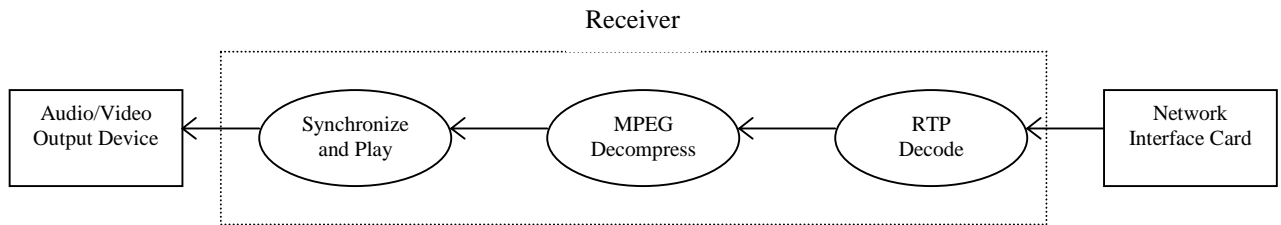


Figure 3. Sender Level 1 Dataflow Diagram

2.1.2 Receiver

Playing back the data on the receiver involves three similar stages but in the reverse order (see figure 4):

1. RTP Decoding
2. MPEG Decompression
3. Synchronization and Playback

In the RTP Decoding stage the receiver waits for packets to arrive. When a new packet arrives, the RTP header is stripped and the timestamp is extracted. The output from this stage is a timestamped MPEG-1 frame. The RTP Decoder may also drop “late” packets (explained further on) at this point to avoid decompressing packets that will eventually be dropped.

The MPEG Decompression stage takes the MPEG-1 frames passed to it from the previous stage and decompresses it to a raw image that can be displayed on the computer screen.

The Synchronization and Playback stage plays back all the video frames in the correct order and at the right time. In other words, it must playback the video with the original time gaps in between successive clips.

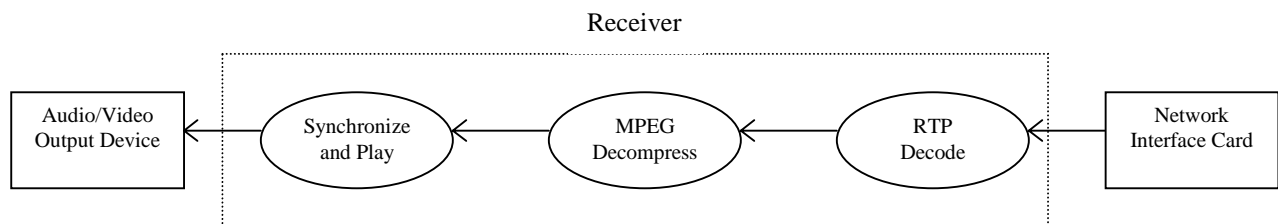


Figure 4. Receiver Level 1 Dataflow Diagram

To play back the packets in real-time the Synchronize and Play stage does one of two things:

1. Drops the packet if it has arrived “late” since the real-time data is no longer valid. If the timestamp of the last packet to be played is T_{i-1} , and the current packet’s timestamp is T_i , and the current time is T_c , then it is considered “late” if $(T_i - T_{i-1}) > (T_c - T_{i-1})$ or $\Delta T_i > \Delta T_c$ (see figure 5). In other words, it is late if the current packet’s timestamp (relative to the timestamp of previously played packet) is greater than the current time (relative to the timestamp of previously played packet).

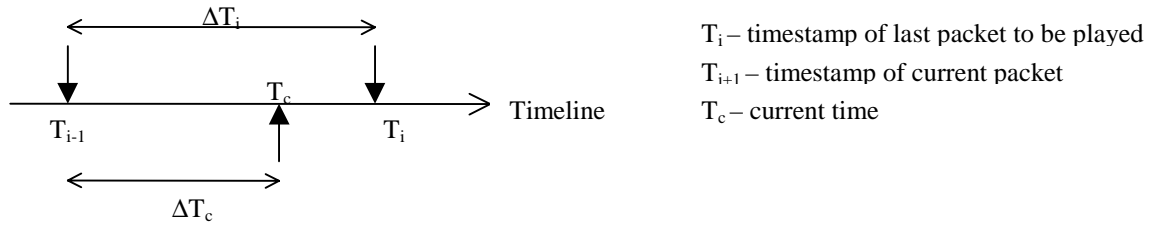


Figure 5. A “Late” Packet

2. Delaying playback of the packet if it has arrived early since it should be played at a later time. If the timestamp of the last packet to be played is T_{i-1} , and the current packet’s timestamp is T_i , and the current time is T_c , then it is considered “early” if $(T_i - T_{i-1}) < (T_c - T_{i-1})$ or $\Delta T_i < \Delta T_c$ (see figure 8). In other words, a packet is considered “early” if the packet’s timestamp relative to the timestamp of previously played packet is less than the current time relative to the timestamp of previously played packet. In this case the packet must be delayed for an interval ΔT_d equal to the difference between the current time and it’s own timestamp.

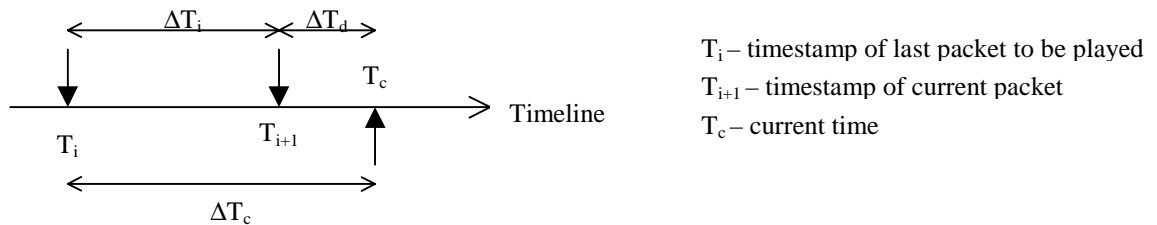


Figure 6. An “Early” Packet

All timestamps are represented in milliseconds. The receiver will not be concerned with the absolute value of the timestamp but rather the relative timestamp (or the time between two successive packets). The receiver uses the timestamp of the first packet as the basis for calculating timings of the rest of the packets. In this scheme no master clock is needed since all timestamps are relative.

2.1.3 Sender-Receiver Synchronization

Videoconferencing requires that both sender and receiver be running simultaneously. This can be accomplished by running the sender and receiver concurrently in separate threads. Both the sender and receiver are fairly independent once they are running but special care must be taken in the initialization and finalization of a video conference call.

The videoconferencing software is started in one of two modes: waiting for a call or placing a call. In placing a call, the receiver thread is started up first so that it is ready to receive and then the sender thread is started. In waiting for a call, the receiver thread can be started up immediately but the sender thread must wait until a connection is made before it can start sending data. The sender must wait until a connection is made since without a connection the sender does not know where to send data.

The program should exit when either the sender or receiver terminates. In other words, if one thread terminates then it must signal to the other thread to terminate in order for the program to terminate gracefully and free acquired resources. This can occur when the sender or receiver window is closed or when the remote client closes the connection.

The only element shared by the sender and receiver threads is the socket. Therefore synchronization of the sender and receiver can be done using the socket.

During initialization the receiver is started up before the sender to ensure that the system is always ready to receive video data (see figure 7). Once the receiver receives the first packet of data it can identify

the source address and port of the caller. At this point the receiver notifies the sender that a call has been received. It does so by issuing a socket connect to the address and port of the packet source. The connection call does not actually set-up an explicit connection. It rather allows the socket to send data without specifying the destination address and also ignores data received from any host-port pair other than the one the socket is “connected” to. The sender periodically checks to see if the socket is connected. If the system is in call mode then the socket will already be connected. The video data will only be sent if the socket is connected.

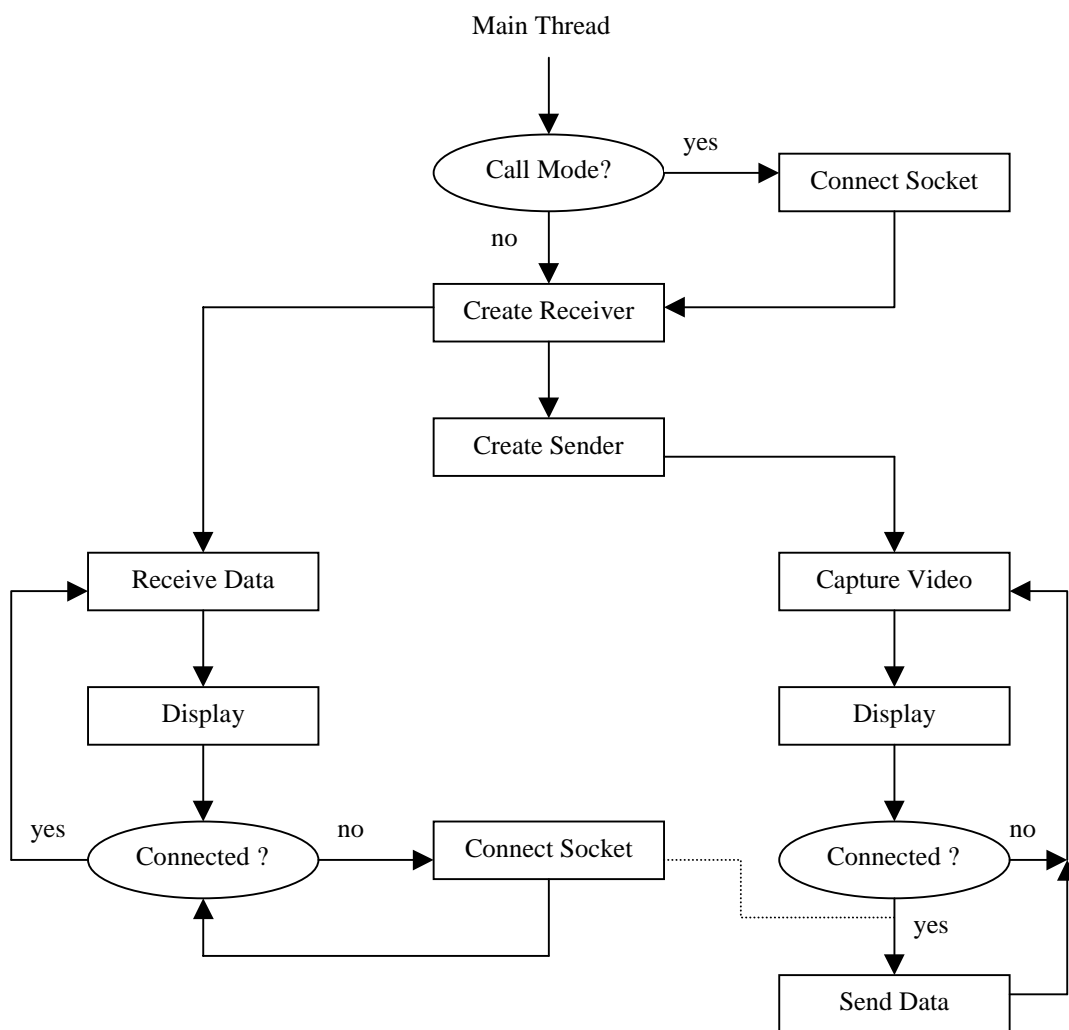
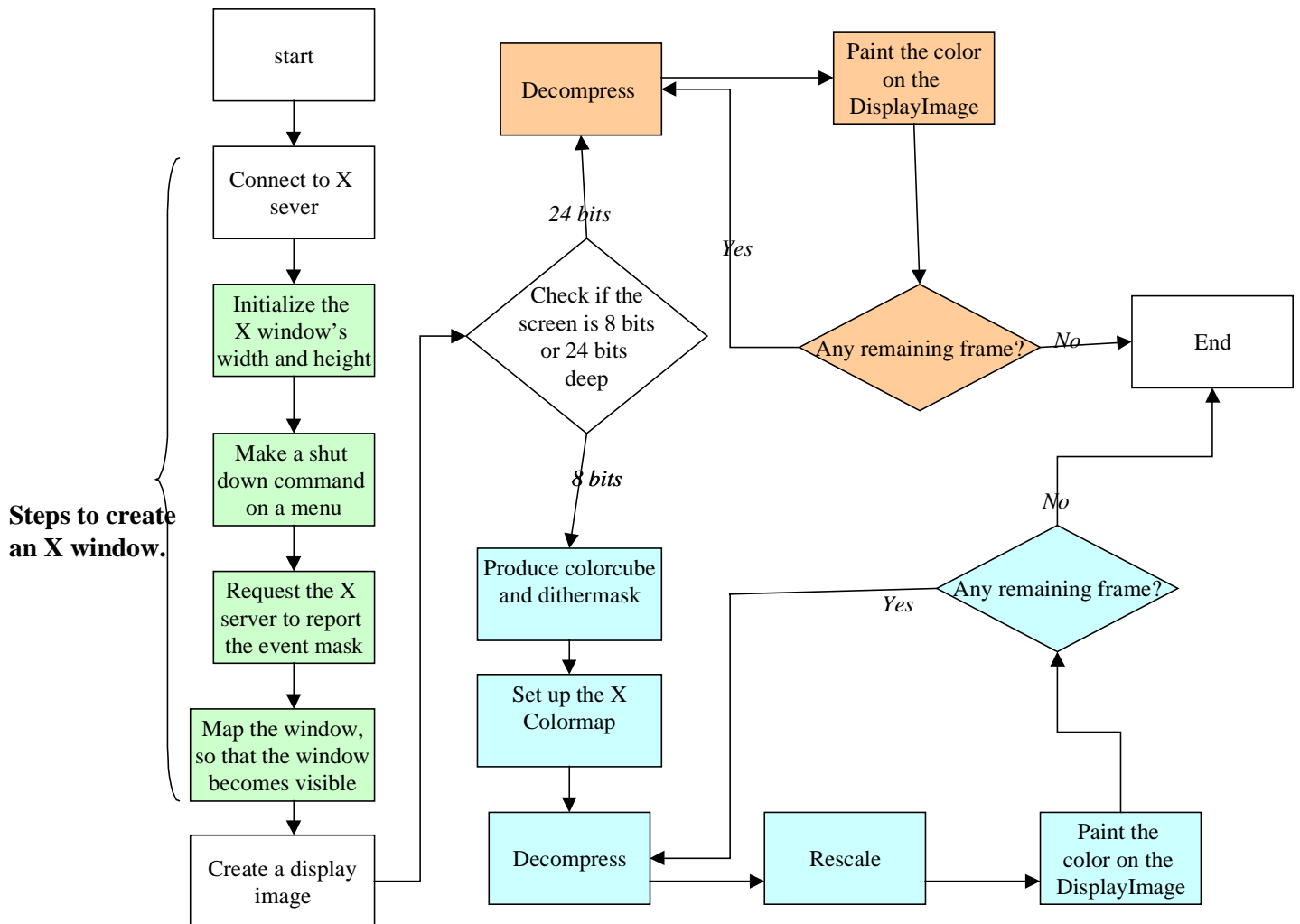


Figure 7. Sender/Receiver Flow Diagram

Figure 8--Implementation of the Decode and User Display



2.2 Video Capture and Display Implementation

Our videoconferencing tool captures video using a Sun video camera attached to a SunVideo input card. The XIL library allows the creation of XilImages (a standard XIL structure to store an image) that are associated with capture device.

This involves

1. Creating a handle to the device (of type XilDevice)
2. Issuing system calls to initialize the device
3. Passing the handle to create an XilImage associated with the device

The created device image behaves just as a regular XilImage. By reading the device image's contents a captured image can be retrieved from the device's frame buffer. Once a frame is read, it is discarded from the frame buffer. Repeated reads from the image will yield a sequence of frames representing the captured video.

By default video is captured at the NTSC resolution of 640x480 but the MPEG-1 compressor in the current version of XIL only supports a resolution of 320x240. So the device image must be scaled to a temporary image before it is passed to the MPEG compressor.

2.3 Video Display

Images either captured from the video camera or received from the network must be displayed in a window. XIL allows the creation of XilImage's that are associated with a window.

This involves:

1. Creating a handle to the window
2. Issuing system calls to initialize the window
3. Passing the handle to create an XilImage associated with the window

The created window image behaves just as a regular XilImage. By copying an image to the window image's buffer the image will be automatically mapped to the window's display area. This is only possible if the types of both images match.

In the situation where the display only supports 8-bit pseudo colour a straight copy cannot be done since the captured images are in 24-bit true colour. Therefore an intermediate conversion must take place in which the 24-bit true colour captured image must be converted to an 8-bit image suitable for displaying directly to the window.

This involves:

1. Creating and installing a colourmap, which maps 24-bit colour, values onto 8-bit colour values.
2. Creating a dither mask
3. Colour conversion of the image using the colour map and dither mask.

Refer to figure 8.0

2.4 MPEG-1 Video

2.4.1 MPEG-1 Implementation

Today, most video conferencing software utilizes video compression to reduce digital bandwidth. Likewise, this project requires a video compression utility that has satisfactory video quality, compression ratio, and is readily available for implementation. MPEG-1 was chosen for the compression and decompression of video images in this project for several reasons. During research into different compression technologies, MPEG-1 was found to have good video quality and compression ratio. The operating system at NAL contained MPEG-1 libraries that would make development of video compression easier, since there was no need to purchase and install a third party CoDec. Also, MPEG-1 is widely used today in many multimedia applications. This would make our software easier to test and debug in the beginning, since sample video images can be easily found from the Internet.

The development for MPEG-1 was started by studying the XIL library system call functions. In addition, the videoconferencing software from Rasheed was examined to familiarize with the MPEG-1 implementation. Rasheed's software was analyzed thoroughly and various components of his design were reused into our project.

The implementation of MPEG-1 was divided into three major components to facilitate the organization and execution of the software. The first component requires specific video values to initialize the overall system and to meet the demands of the user. This would allow hardware initialization and precise video specifications to be met. The second component consists of using the initialization routine to acquire and transmit MPEG-1 compressed video images using the communication protocols. Finally, the last component involves acquiring the MPEG-1 video data from the communication protocol and decompressing it into the user's display.

2.4.2 MPEG-1 Initialization

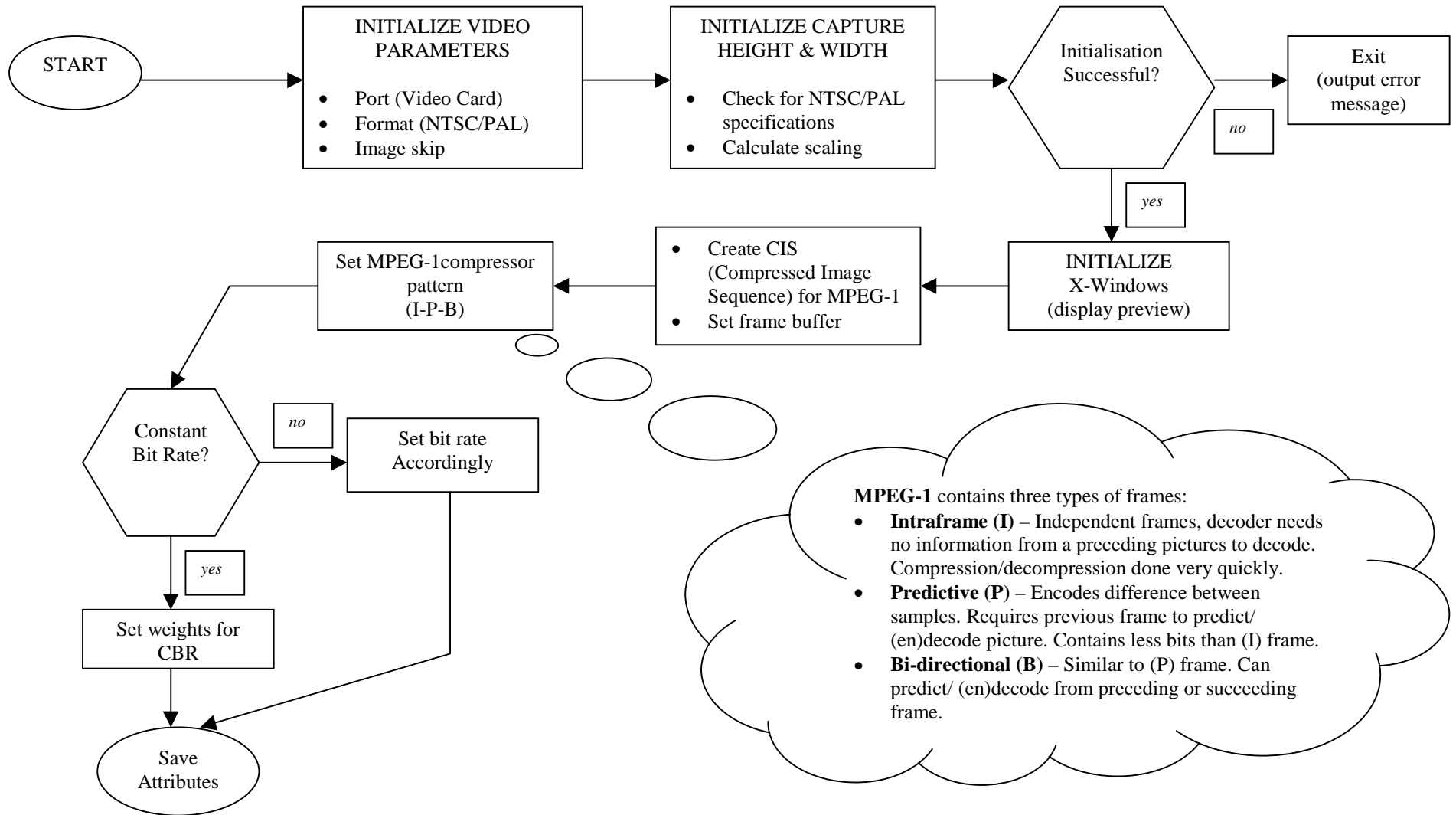
For the execution of the MPEG-1 system, there is a need for initializing the system and the user's requirements. This is accomplished by executing system calls offered by the XIL library. The routine consists of seven steps to successfully initialize the video parameters at the server (Fig. 9).

The first step consists of setting the hardware parameters such as video port, video format (NTSC/PAL), frame skip rate, and buffer size. The second step involves configuring the dimensions of the captured raw image data. A relatively small window capture size is created so that the video data will not occupy excessive memory. The raw image will be reduced to half the capture size in order for the image to be acceptable for viewing. As a result of this step, the raw image will be transformed to lower resolution. If these steps are not successful, the system will not proceed to the next step and will exit the initialization sequence. However, if the former sequences were successful, the system will proceed to initialize the X-window display, which was elaborated in chapter 2.3 (Video Display) of this paper.

The following creates the states and variables needed for an MPEG-1 sequence. It includes the creation of a compressor pattern specified by the user. The compressor pattern consists of three types of frames. Interlaced (I-picture), Previous (P-picture), and Bi-directional (B-picture). These frames are used for the motion estimation feature of MPEG-1. The I-pictures are based on the independence of each frame. Like raw digital images, I-pictures assume independence for each frame. In other words, each frame does not need any information from other frames in order to be displayed. However, P-pictures are based on the dependence of the previous frame, hence the name motion estimation. When compressing an image, two frames are needed and only the difference between the two frames will be saved. During decoding of a sequence, a "key" frame or I-picture is needed to start the decoding. Similarly, a B-frame involves the dependence of frames, not only using the preceding frames, but also the succeeding frames. Again, an I-picture is needed to start the decoding.

The next step, bit-rate initialization, the system will first require to recognize the type of frame. If the MPEG-1 sequence contains I-pictures only, the compression type will have a variable bit rate. However, if the sequence contains P or B-pictures, the compression type will have a constant bit rate. If P or B pictures are used, then specific weights for each type of frame are specified, and a constant bit rate will be produced. On the other hand, if I-pictures are only used, variable bit rate compression will be used and calculated dynamically by the XIL library.

Figure 9 - Server MPEG1 Initialisation

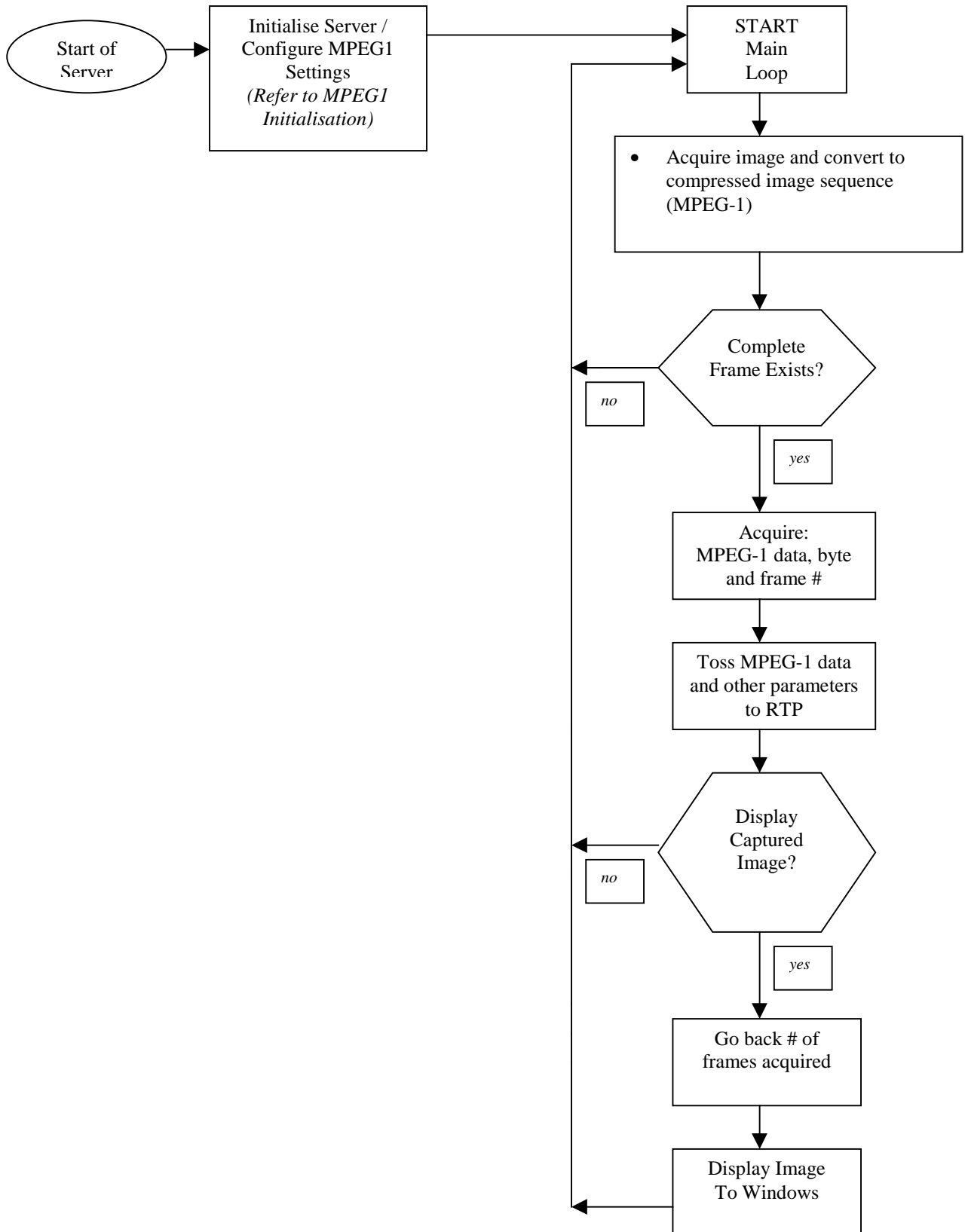


2.4.3 MPEG-1 Transmission

During the start of the server (Fig. 10), MPEG-1 must be first initialized as mentioned in the previous section. Once this has successfully been accomplished, the transmission routine will enter into a loop. Inside the loop, the first step is to acquire the raw image data from the video card buffer. Each raw frame will be converted into an MPEG-1 frame and stored in memory. If this procedure is successful, the MPEG-1 frame will be retrieved from memory and forwarded to the communication socket where it will be packetized by the RTP protocol.

In addition, the user will have the option to display the captured image on his/her monitor. If this option is exercised, the system will have to go back the number of frames that were transmitted and retrieve those frames from memory. Once this has been accomplished, the transmitted/-displayed frames will be flushed from the computer system. The above sequences will repeat until the user window is closed or until an error has occurred.

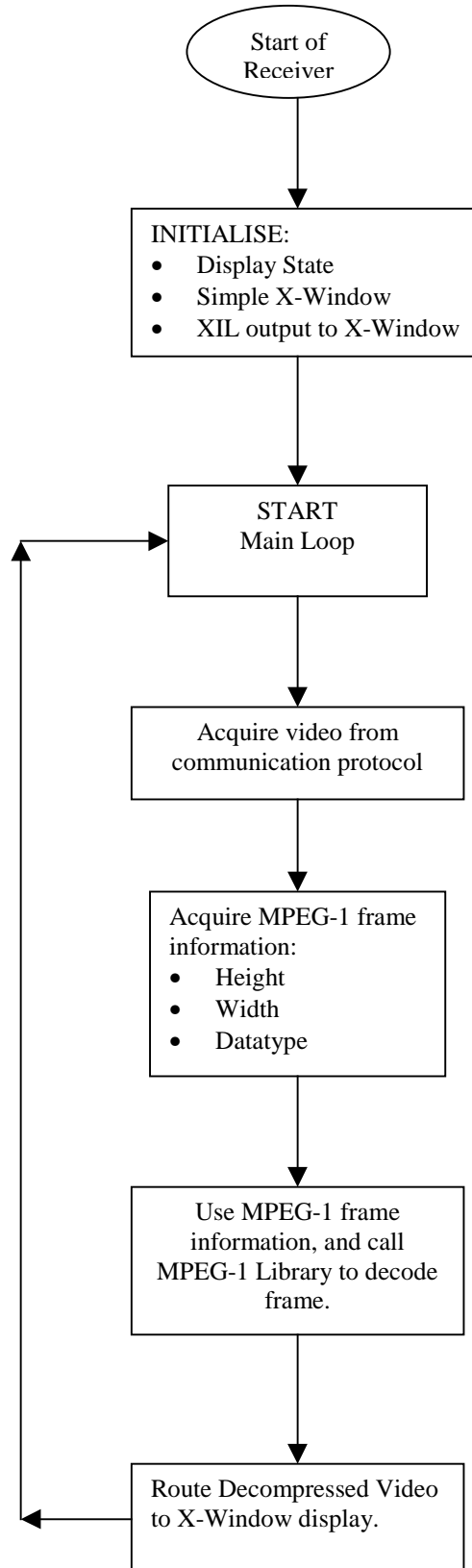
Figure 10 - SERVER Transmission



2.4.4 MPEG-1 Reception

During the start of the client (Fig. 11), the X-window must be first initialized (refer to chapter 2.3). Once the X-window this has been successfully initialized, the receiver system will enter into a loop. Inside the loop, the first step is to acquire the depacketized MPEG-1 frame from the receiver queue. This part of the data will contain valuable information for decompressing purposes. Information such as height, width, and frame type will be acquired and along with the actual MPEG-1 video data, the frame can be successfully decompressed. The decompressed image will be routed to the X-windows display, where the user can view the received image. The above sequences will repeat until the user window is closed or until an error has occurred.

Figure 11 - CLIENT Receiver



Chapter 3: CONCLUSION

3.1 Concluding Remarks

So far, we have successfully implemented a bi-directional video conferencing tool that uses UDP/IP protocols for communication. We have previously installed the RTP protocol, but is currently not in our system. During testing with the RTP protocol, our system was functioning erratically. We have noticed that implementing the RTP protocol causes the system to become unstable and to crash periodically. We attempted to debug this problem but we were not successful due to time constraints. Instead, we have commented the source code that implements the RTP protocol. Nevertheless, testing of the video conferencing tool proved that the receiver can playback the video frames smoothly.

Our project meets the bi-directional requirement. The conferencing tool allows each end system (e.g. Jupiter and Saturn) to act as both a sender and a receiver at the same time. As a result, each system was able to capture, send, receive, and display images concurrently.

Our successful implementation of the Video Conferencing has a number of contributions. First, we have solved several Solaris system problems. Before we work on this project, several other previous NAL projects that use XIL cannot be compiled using the XIL 1.3 library in the newly upgraded Solaris 2.6 System at NAL. Also, no one was able to use the attached Sun Video Device to capture images in the newly upgraded system. During our progress of implementing the project, we were able to fix all these problems with the help of the UNIX administrator, Wing-Chung Hung. As a result, users in NAL are able to use the XIL 1.3 library and also use the Sun Video Device to capture images.

In conclusion, our current video conferencing tool satisfies several key communication features and is a valuable research tool in the field video conferencing. Indeed, this current working program serves as an immediate point for next year's design project to extend the functionality of our system.

3.2 Suggestions for Future Work

The currently implemented videoconferencing tool only supports one-to-one communications. One possible extension would be to add support for many-to-many communications, which is available in most other videoconferencing software. This would allow a person using the videoconferencing tool to talk to more than one person at a time. To do this efficiently, IP multicasting must be taken advantage of. IP multicasting makes more efficient use of the network bandwidth by sending only a single copy of data into the network, which can be picked up by multiple receivers. IP multicasting is supported in the RTP library used for the project.

Another extension would be to add support for ATM transport. ATM transport could allow for more tightly controlled Quality of Service guarantees in data delivery.

Our system has minimal user interface consisting of windows displaying video. A well-designed user interface could be built around the videoconferencing system making it much more user friendly. The user interface could be used to report session statistics and allow the user to dynamically change system parameters.

The video-processing library we used for our project at this point does not support MPEG-2 compression/decompression. If however MPEG-2 were to be supported in future versions of the library, MPEG-2 could replace the current MPEG-1 compression to provide higher compression rates and better video quality.

References

- B.P. Lathi, 1998. Modern Digital & Analog Communication Systems – 3rd edition, Oxford University Press, New York, sec. 8.6.
- Hoffman, D., G. Fernando, V. Goyal, M. Civanlar. January 1998. RTP Payload Format for MPEG1/MPEG2 Video. RFC 2250. <http://info.internet.isi.edu/in-notes/rfc/files/rfc2250.txt>.
- Katzela, I. 1998. Multimedia Networking. University of Toronto.
- Liu, C. August 1997. Multimedia Over IP: RSVP, RTP, RTCP, RTSP.
http://www.cis.ohio-state.edu/~jain/cis788-97/ip_multimedia/index.htm
- Leon-Garcia, A., Widjaja I. 1998. Communication Networks: Fundamental Concepts and Key Architectures. University of Toronto.
- Motion Pictures Expert Organization, 1998. MPEG-1 FAQ.
<http://www.mpeg.org/faq>
- Robinet, J. 1998. An Implementation of a Gateway for Heirarchically Encoded Video Across ATM and IP Networks. Masters Thesis. University of Toronto. Department of Electrical and Computer Engineering.
- Schulzrinne, H. January 1996. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 1890. <http://info.internet.isi.edu/in-notes/rfc/files/rfc1890.txt>.
- Schulzrinne, H. October 1997. About RTP and the Audio-Visual Transport Working Group.
<http://www.cs.columbia.edu/~hgs/rtp>.
- Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V. January 1996. RTP: A Transport Protocol for Real-Time Applications. RFC 1889. <http://info.internet.isi.edu/in-notes/rfc/files/rfc1889.txt>.
- Schulzrinne, H., Lennox, J., Rubenstein, D., Rosenberg, J. June 1998. RTP Library API Specification Lucent Technologies.
- Stevens, W. 1998. UNIX Network Programming. 2nd ed. New Jersey: Prentice-Hall.
- Sun Microsystems Inc. 1994. Sun Video User's Guide.
- Sun Microsystems Inc. 1997. SunVideo Plus Answerbook.
http://docs.sun.com/ab2/@Ab1CollToc?abcardcat=%2Fsafedir%2Fspace4%2Fpkgs%2Foldversioncol%2Fab1%2FSUNWsvpab%2Fab_cardcatalog;subject=workstation
- Sun Microsystems Inc. 1997. Solaris XIL 1.3 Answerbook.
http://docs.sun.com/ab2/@Ab1CollToc?abcardcat=/safedir/space3/pkgs/collections/ab1/answerbooks/english/solaris_2.6/SUNWAXi/ab_cardcatalog

Appendix A: Code Module Descriptions

librtp.a

Lucent Technologies' static library for RTP.

librtpunix.a

Lucent Technologies' static library for RTP.

mpeg1.h

Defines the `RtvcMpeg1Weights` structure for setting the `MPEG_WEIGHTS` parameter in the RTVC device.

receiver.c

Main code for receiver process responsible for receiving MPEG-1 frames, decompressing them, and displaying them in a window.

receiver.h

Header file containing exported function prototypes from `receiver.c` module.

rtp_api.h

RTP API types, structures, and functions that a user of RTP might require. (Included with Lucent Technologies' RTP libraries).

rtp_highlevel.h

RTP API types, structures, and necessary functions exclusive to the high-level (network-dependent) parts of the interface. (Included with Lucent Technologies' RTP libraries).

rtp_util.c

Wrapper functions for RTP API calls. These functions encapsulate frequently used RTP API call sequences.

rtp_util.h

Header file containing exported function prototypes from `rtp_util.c` module.

sender.c

Main code for sender process responsible for capturing images from video camera, compressing images to MPEG-1, and sending them to the network.

sender.h

Header file containing exported function prototypes from `sender.c` module.

timeval.c

Contains functions for converting timeval structures (used in system calls to retrieve time).

udp.c

Wrapper functions for common socket API calls used with datagram sockets.

udp.h

Header file containing exported function prototypes from udp.c module.

vctool.c

Primary module containing the main function. This is responsible for opening XIL, opening the display, parsing command line parameters, initializing and running the sender and receiver.

xil_util.c

Utility functions for encapsulating sequences of XIL API calls such as compressing CIS's, creating colour maps, and creating windows.

xil_util.h

Header file containing exported function prototypes from xil_util.c module.

Appendix B: Manual Page

NAME

vctool – bi-directional Internet videoconferencing tool

SYNOPSIS

```
vctool [-p pattern] [-is image_skip] [-cbr cbr_rate] [-d device] [-dn device_name]
      [-dp device_port] [-l local_port] [-c host:port]
```

DESCRIPTION

vctool is a real-time videoconferencing tool that allows bidirectional transmission of captured video. The software is used with SunVideo card

The videoconferencing tool can be started up in one of two modes: listen mode or call mode.

The person expecting to receive a call must start vctool up in listen mode. To start up in listen mode a listening port must be specified. The -l flag is used to instruct the vctool to start up in listen mode and the number following the flag specifies the local port to listen on.

The person making the call must start vctool up in connect mode. To start up in connect mode the remote host and port to which you would like to connect to must be specified. The -c flag is used to instruct the vctool to start up in connect mode. The flag is followed by a string representing a host-port pair separated by a colon. If there is no receiver listening on the specified port then the program will exit.

OPTIONS

-p pattern

Use alternate pattern for MPEG-1 compression. Pattern can be a sequence of I, P, B representing I-frames, P-frames, and B-frames, respectively. (default = "T")

-is image_skip

Sets the number of frames to skip in capturing video. (default = 0)

-cbr cbr_rate

Sets the data rate for constant bit rate MPEG-1 encoding (default = 115,000)

-d device

Sets the device class to use. (default = "SUNrtvc")

-dn device_name

Sets the name of device to use for capturing. (default = "/dev/rtvc0")

-dp device_port

Sets the capture device port. (default = 1)

EXAMPLES

1. The following command:

```
example% vctool -l 5555
```

Starts up the video conferencing tool in listen mode listening on local port 5555 for a call.

2. The following command:

```
example% vctool -c jupiter:5555
```

Starts up the videoconferencing tool in call mode where it will try to connect to the computer named Jupiter who is already listening on port 5555.

Appendix C: Source Code Listing