

Real-Time Transmission of Video and Text complexity through RTP (Real-Time Transport Protocol)

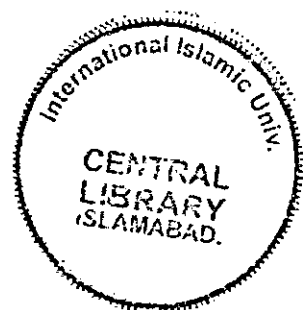


Developed by:

**Rehana Yasmin
324-CS/MS/F06**

Supervised by:

Dr. Muhammad Sher



Department of Computer Science
Faculty of Basic and Applied Sciences
International Islamic University Islamabad
(2008)

**WITH THE NAME OF ALLAH ALMIGHTY,
THE MOST BENEFICIENT,
THE MOST MERCIFUL**

Department of Computer Science
International Islamic University Islamabad

Date: _____


Final Approval

This is to certify that we have read the thesis submitted by **Rehana Yasmin** 324-CS/MS/F06. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by International Islamic University, Islamabad for the degree of MS in Computer Science.

Committee:

External Examiner

Mr. Shiraz Baig
CEO
Askari Information Systems
Islamabad



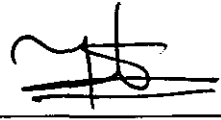
Internal Examiners

Mr. Matta-ur-Rehman
Assistant Professor
Department of Computer Sciences,
Faculty of Basic and Applied Sciences,
International Islamic University,
Islamabad



Supervisor

Dr. Muhammad Sher
Chairman
Department of Computer Sciences,
Faculty of Basic and Applied Sciences,
International Islamic University,
Islamabad



Dedication

Dedicated to My Beloved Parents

*After Allah Almighty, I am very grateful to my parents for this dissertation,
whose affection has always been a source of boast for me, and whose
prayers always work out to be a key to my success.*

Rehana Yasmin
324-CS/MS/F06

**A dissertation Submitted To
Department of Computer Science,
Faculty of Basic and Applied Sciences,
International Islamic University, Islamabad
As a Partial Fulfillment of the Requirement for the Award of the
Degree of MS in Computer Sciences.**

Declaration

I hereby, declare that this thesis “*Real-time Transmission of Video and Text Complexity through RTP (Real-Time Transport Protocol)*” neither as a whole nor as a part has been copied out from any source. It is further declared that I have done this research with the accompanied research thesis report entirely on the basis of my own personal efforts under the proficient guidance of my teachers and supervisor Dr Muhammad Sher. If any part of the system is proved to be copied out from any source or found to be reproduction of any project from any of the training institute or educational institutions, I shall stand by the consequences.

Rehana Yasmin

324-CS/MS/F06

Acknowledgment

With the name of *Allah Almighty*, WHO is the most Merciful and Beneficent. The most Gracious and Compassionate, WHOSE abundant blessings enabled me to pursue and complete this research project. After *Allah almighty* all praises are for His *Holy Prophet Muhammad (SAW)* who enabled us to recognize our Creator, *Allah almighty* and who is a role model for us in every aspect of our life.

I would consider it a proud to express my warmth gratitude and deep sense of obligation to my supervisor *Dr. M Sher*, Head of Department of Computer Science, International Islamic University Islamabad for his skilful guidance, knowledge sharing and kind behavior during this research project.

This would be also a proud for me to express my heartily gratitude for my most honorable teacher Mr. Shiraz Baig, for his dedication, encouraging attitude, untiring help and kind behavior throughout of this project work. His encouragement boasted me to initiate and complete this project.

In last but not least it was mainly due to my parents whose moral and financial support enabled me to complete this task particularly my mother who always prays for my success.

I am also thankful to all my faculty members and my friends.

Rehana Yasmin
324-CS/MS/F06

Project in Brief

Project Title	Real-time Transmission of Video and Text Complexity Through RTP (Real-Time Transport Protocol)
Objective	To develop an environment that allows real-time applications to send text data along with the video data in real-time through RTP
Undertaken by	Rehana Yasmin 324-CS/MS/F06
Supervised by	Dr. Muhammad Sher Head of Department of CS, International Islamic University Islamabad.
Starting Date	September 2007
Completion Date	October 2008
Tool Used	C/C++ language
Operating System	Linux Distribution - Fedora Core 5 Kernel 2.6.15
System Used	Intel Pentium IV

Abstract

This research is a study for an efficient transmission of video data along with the textual data that is linked with this video, on the network. The study includes finding an efficient way of transmitting Video and text data contents through RTP (Real-Time Transport Protocol) without impairing quality of both types of data. The study will be based on RTP (Real-Time Transport Protocol) and RTCP (RTP Control Protocol), two companion parts of the protocol RTP. The issues handled are how to send text data along with the video data in real time using RTP, which is basically a protocol used for the transmission of real time multimedia contents over network, and how to synchronize both types of data on receiving side.

Possibilities to be investigated are either to maintain a single RTP session for both types of data or to maintain separate sessions for them. This research will help to find an efficient approach to send text data in real-time together with the other multimedia types using RTP.

Table of Contents

Chapter	Page #
1. Introduction	1
1.1 Motivation behind the project	1
1.2 Real time Transport Protocol	1
1.2.1 Background	1
1.2.2 Real-Time Transport Protocol (RTP) Description	2
1.2.3 Protocol Structure (RTP)	3
1.2.4 RTP Control Protocol (RTCP) Description	6
1.2.5 Protocol Structure (RTCP)	6
1.3 RTP Video	7
1.3.1 Video Compression	7
1.3.2 Video Codec	8
1.3.3 RTP Video Encodings	8
1.4 Problem Identification	12
1.5 Proposed Solution	12
1.6 Outline of the Thesis	14
2. Literature Survey	16
2.1 Introduction	16
2.2 Transport Techniques	16
2.2.1 Transmission Control Protocol (TCP)	16
2.2.2 User Datagram Protocol (UDP)	16
2.2.3 Real Time Streaming Protocol (RTSP)	17
2.2.4 Real Time Transport Protocol (RTP)	17
2.2.5 Other Related Research Work	18
2.3 Limitations	18
2.4 Summary	19
3. Requirement Analysis	21
3.1 Introduction	21

5.2.2.3 Installation and Configuration of System.....	42
5.2.2.4 Image Capture through Camcorder.....	44
5.2.2.5 Video Encoding Used.....	48
5.2.3 Text Input.....	50
5.2.4 Packetization of Text and Video Data.....	50
5.2.4.1 Packet Format for Video Data Only.....	50
5.2.4.2 Packet Format for Video and Text Data.....	51
5.2.5 Implementation of Buffering	51
5.2.6 Implementation of Redundant Text Data.....	52
5.2.6.1 Packet Format for Redundant Text Data.....	52
5.2.7 Duplication of Received Packets.....	53
5.2.8 Port Numbers for RTP & RTCP.....	53
5.2.9 Modification of libjpeg.....	53
5.2.10 Synchronized Display.....	54
5.2.10.1 Display Program.....	55
5.2.11 Graphical User Interfaces.....	56
5.3 Implementation Tools.....	56
5.4 Summary.....	57
6. Testing and Performance Improvement.....	58
6.1 Overview	58
6.2 Testing.....	58
6.3 Performance Improvement.....	59
6.3.1 Single Packet to Reduce Bandwidth Wastage.....	59
6.3.2 Buffering of Text Data	59
6.3.3 Redundant Text Data.....	60
6.3.4 No Duplication	60
6.3.5 Read Image Data into Memory for Decompression.....	60
6.3.6 Text Compression.....	60
6.4 How to use this software over Internet.....	61
6.5 Summary.....	62

7. Conclusions and Future Enhancements.....63

 7.1 General Discussions 63

 7.2 Conclusions 63

 7.3 Future Enhancements 64

Appendix A: Header File (rtp.h).....65

Appendix B: Interfaces69

Appendix C: JPEG Library.....78

Appendix D: References.....83

CHAPTER 1

INTRODUCTION

1. Introduction

This research is an attempt to find a way to handle text contents in real-time. Real-time applications have their own constraint in terms of delay. Text data is also more sensitive to packet loss as compare to other multimedia type's contents. In order reception and display of text data contents are also very much important. Loss of a few text packets or even a single text packet can disturb the whole text conversation meanings. Therefore, real-time text contents require careful handling of data. RTP is the best choice to handle text contents in real-time.

1.1 Motivations behind the project

Demand for the real time multimedia applications has been increasing day by day. These applications transfer real time contents i.e. digital audio and video in real time, for example multimedia conferencing, capture and broadcast of live events by TV channels and news agencies etc. RTP is a protocol which is currently being used for the transmission of the real time multimedia contents in real time. The Transmission Control Protocol (TCP), designed to transfer text packets over the Internet, overcomes the problem of transmission of text in a reliable manner. However, TCP does not suite to carry real-time data. On the other hand, UDP alone does not provide mechanism to handle real time contents. So there is a need to modify RTP (Real time transport protocol) in order to accommodate real time text data as well. Attempts have been made to transmit textual data alone through RTP. This research is a study to find the possibility of sending text data along with video data through RTP in real time and synchronizing both types of data on receiving side.

1.2 Real-time Transport Protocol (RTP)

1.2.1 Background

Guarantee of services delivery is always a problem on the Internet, how best effort they are built. The Transmission Control Protocol (TCP), designed to transfer text packets over the Internet, overcomes this problem by retransmitting the packets that fail to reach their destination. But this causes an end-to-end packet delay.

Today there is a well-known demand of the real-time applications that transfer multimedia contents i.e. digital audio and video in real time, for example multimedia conferencing. These real-time applications have their own requirements, in terms of delay, error rate, jitter and throughput those are different than other applications. Timely delivery is most crucial for these applications.

In order to maintain the good quality of audio and video applications, these types of data must be played back at their sampling rate. For a better quality playback, data must arrive in time otherwise playing back process will be affected and human ears and eyes will pick the artifacts. A latency of 250 milliseconds can be tolerated by internet telephony applications but if it exceeds this limit, it will affect the quality of voice.

Network congestion is another serious issue for real-time applications. It has more serious effects on real-time traffic than non-real time traffic. The only effect of congested network on non-real time traffic is that the transmission takes longer time to complete, but the real-time data becomes outdated. It ultimately affects the quality of playback. The retransmission of lost packets would make the situation worse and jam the network.

Multimedia applications are mostly multicast applications for which the same data is sent to a group of receivers at the same time rather than sending separate copies. For example, in video conference, the video data is sent to a group of all participants of the conference simultaneously. So the protocols designed for multimedia applications must allow for multicast in order to reduce the traffic burden.

Internet is a packet-switching datagram network. The other transport technologies cannot guarantee that real-time data will reach the destination in time and without being messy and bouncy. So a need was felt that a new transport protocol must be put into effect to take care of the timing issues so that audio and video data can be played back continuously within the time limit and synchronized as well [10].

Hence, in order to meet these requirements of the real-time multimedia applications, Internet Engineering Task Force (IETF) designed the Real-time Transport Protocol for continuous media transmission in real-time.

1.2.2 Real-Time Transport Protocol (RTP) Description

The Real-time Transport Protocol (RTP) provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video or simulation data, over multicast or unicast network services. Those services include:

- Time stamping
- Sequence numbering
- Identification of payload type
- Monitoring QoS of data transmission

The RTP is primarily designed to meet the needs of multicast of real-time data, but it can also be used in unicast applications. Primary focus of RTP was multi-participant multimedia conferences and IP telephony but its implementation was not restricted. Applications can implement it according to their own need. It can be used for one-way transport such as video-on-demand as well as interactive services such as Internet telephony. Other than these, RTP is also being used in different types of applications like storage of continuous data, interactive distributed simulation, active badge, and measurement and control applications.

The RTP specifications recommend the use of two different generalized destination ports within one RTP session, one port for the reception of RTP payload packets and one for RTCP control packets. RTP is used in combination with other network or transport protocols as shown in the following figure.

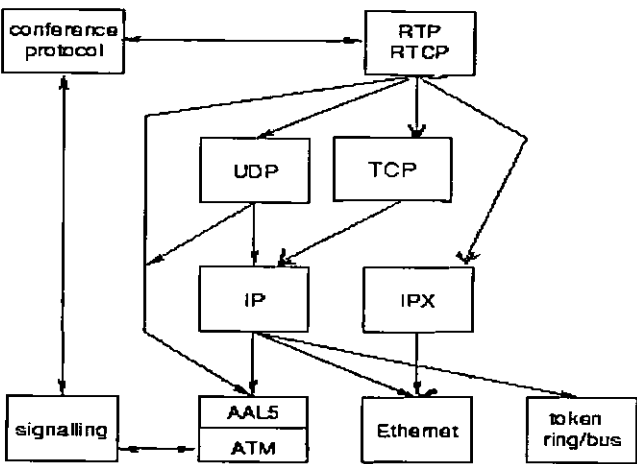


Figure: RTP over other network protocols

Typically, RTP run on top of UDP to make use of its multiplexing and checksum services. Both these protocols together provide transport protocol functionality. However, RTP may be used with other suitable underlying network or transport protocols. RTP supports data transfer to multiple destinations using multicast distribution if provided by the underlying network.

RTP Packet in a UDP/IP stack: The RTP header is created first and then the packet is moved down the stack to UDP and UDP header is attached and then IP header is attached. The following figure shows the RTP stack.

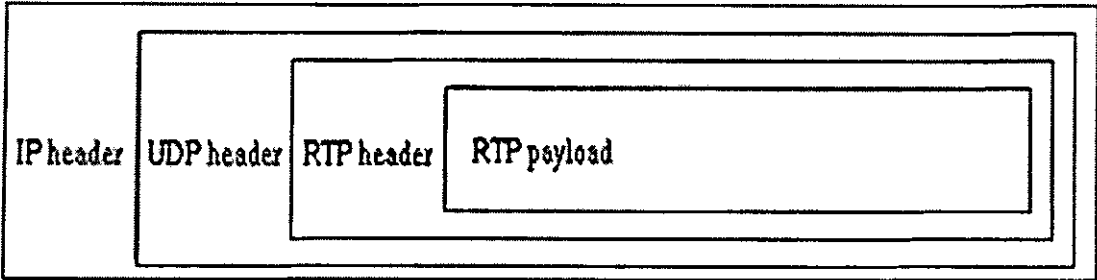


Figure: RTP/UDP/IP protocols stack

1.2.3 Protocol Structure (RTP)

RTP data packets consist of a header followed by payload data. The RTP header has the following format:

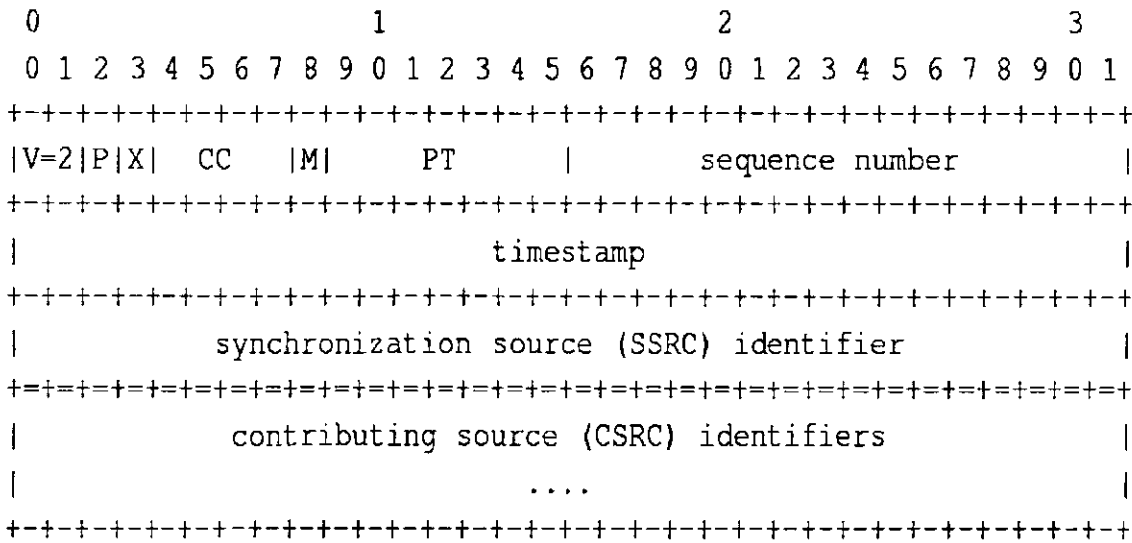


Figure: RTP header format [according to RFC 3550]

First 12 bytes of the header are fixed for every data packet, while the list of CSRC identifiers is present only when inserted by a mixer. The payload data can be either a video frame or several audio samples. The fields have the following meanings:

Version (V): 2 bits

This field identifies the version of RTP. The version defined by RFC 3550 specification is two (2). (The value 1 is used by the first draft version of RTP and the value 0 is used by the protocol initially implemented in the "vat" audio tool).

Padding (P): 1 bit

If the padding bit is set, the packet contains one or more additional padding octets at the end which are not part of the payload. The last octet of the padding contains a count of how many padding octets should be ignored, including itself.

Extension (X): 1 bit

If the extension bit is set, the fixed header is followed by the exactly one header extension, with a defined format.

CSRC count (CC): 4 bits

The CSRC count contains the number of CSRC identifiers that follow the fixed header.

Marker (M): 1 bit

The interpretation of the marker is defined by a profile. It is intended to allow significant events such as frame boundaries to be marked in the packet stream.

Payload type (PT): 7 bits

This field identifies the format of the RTP payload and determines its interpretation by the application. A profile may specify a default static mapping of payload type codes to payload formats. Additional payload type codes may be defined dynamically through non-RTP means.

Sequence number: 16 bits

The sequence number increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. The initial value of the sequence number is random (unpredictable) to make known-plaintext attacks on encryption more difficult.

Timestamp: 32 bits

The timestamp reflects the sampling instant of the first octet in the RTP data packet. The sampling instant must be derived from a clock that increments monotonically and linearly in time to allow synchronization and jitter calculations. The initial value of the timestamp is random, as for the sequence number. Several consecutive RTP packets will have equal timestamps if they are (logically) generated at once, e.g., if they belong to the same video frame. Consecutive RTP packets may contain timestamps that are not monotonic if the data is not transmitted in the order it was sampled. (The sequence numbers of the packets as transmitted will still be monotonic).

SSRC (Synchronization Source): 32 bits

This identifier is chosen randomly, with the intent that no two synchronization sources within the same RTP session will have the same SSRC identifier.

CSRC (Contributing source identifiers) list: 0 to 15 items, 32 bits each

The CSRC list identifies the contributing sources for the payload contained in this packet. The number of identifiers is given by the CC field. If there are more than 15 contributing sources, only 15 can be identified.

RTP itself does not provide any means to ensure timely delivery or provide other quality-of-service guarantees. For these functions, it relies on lower-layer services. It does not provide guarantee of delivery or in-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence. Receiver can restructure the sender's packet sequence using the sequence numbers included in the received RTP packets. The sequence number can also be used to determine the proper location of a packet in decoding without necessarily decoding the packets in sequence [2].

RTP consists of two protocols: RTP and RTCP.

RTP: responsible for real time transmission of data packets.

RTP control protocol (RTCP): provides control functionality. It helps in monitoring QoS and in conveying participants' information in an on-going RTP session.

1.2.4 RTP Control Protocol (RTCP) Description

The control functionality of RTCP is described below:

- 1- **QoS Monitoring:** RTCP provides traffic monitoring by gathering some control statistics and sending them as a feedback of quality of data distribution in the form of reports. These statistics helps to control congestion which causes delay. The feedback is maintained by two RTCP reports: *Sender Report (SR)*, and *Receiver Report (RR)*. Both reports contain performance statistics on number of packets lost, highest sequence number received, jitter, and other delay measurements to calculate the round-trip delay time and these statistics may be used to modify sender transmission rates in order to avoid congestion and for diagnostics purposes.
- 2- **Identification of source:** RTCP maintains a persistent transport-level identifier for an RTP source, called canonical name, CNAME. Receivers use CNAME to keep track of each participant in the RTP session and to synchronize related media streams (with the help of NTP).
- 3- **Calculation of transmission rate:** Because of bandwidth limitations and expected large number of participants, the rate at which packets are sent must be controlled. The rate can be calculated by getting the total number of participants in an RTP session using RTCP reports.
- 4- **Session control information:** An OPTIONAL function of RTCP is to convey minimal session control information, for example participant identification to be displayed in the user interface.

1.2.5 Protocol Structure (RTCP)

The RTCP packet carries following control information:

- SR: sender reports; sending and reception state
- RR: receiver reports; for reception statistics from multiple sources
- SDES: source description item, include CNAME
- BYE: indicates end of participation
- APP: application specific functions

1.3 RTP Video

Traditionally RTP is used for the transmission of continuous media, e.g. audio, video or audio plus video. The possibility of transmitting such media over networks facilitates new kinds of distributed applications, such as digital television broadcasting, teleconferencing, or multimedia information systems.

Handling video transmission over packet switched network and designing an optimal video transmission application for such network is a very challenging job. For the efficient transmission of video data a good knowledge of the transport network is required. It also requires a good understanding of the whole video data and its manipulation. Video applications also need some form of video data compression to achieve reasonable size for storage and transmission. The digital video compression is one of the main issues in digital video coding. It enables efficient transmission of visual information.

1.3.1 Video Compression

To transverse network optimally, video data size should be smaller enough. As we know digital video files tend to have a larger size. For an efficient transmission file size is an important concern. So to reduce the transmission size of video, the only solution is compression.

How compression reduces digital video data size?

Compression is the process of skipping or eliminating those data from video data, which can not be perceived by human's eyes. Compression utilizes human's limitations to reduce video data size. For example, humans can see only about 1024 color shades, although there are billions of colors. As human eye can not distinguish the slight difference between the two consecutive shades, so this fact is utilized to reduce the file size by not keeping every color. In this way digital video can be compressed without affecting the perceived quality by human eye. Same can be applied to redundancy in consecutive images. There may be redundant data in consecutive frames. This redundant data can be eliminated.

There are different video formats which provide different compression ratios. But *compression should be till one limit. Too much can be a bad thing and can affect perceived quality of video.* More you compress, more data is skipped, and eliminating more data may result in changes which are noticeable by human eyes. Ultimately it will affect perceived quality. So compression should be as much as possible until data loss becomes noticeable. Otherwise color fidelity fades, artifacts and noise appear in the picture, the edges of objects become over-apparent, and eventually the result is unwatchable.

As compare to video data text files can be compressed to a high level. Spaces in text files can be utilized to achieve maximum compression. A text file can be made 80 to 90 percent smaller.

1.3.2 Video Codec

Video Codec is the technology through which compression is achieved. It is used to compress and decompress as well as to encode and decode video streams. The main goal of coding is to reduce bit-rate for storage and transmission of the video source while preserving the video quality as good as possible. The word codec may be a combination of any of the following: 'compressor-decompressor', 'coder-decoder', or 'compression/decompression algorithm'. Various types of codecs have been developed implementable either in software or in hardware, and sometimes utilizing both. Codecs allow video to be translated to and from its compressed state with good grace.

1.3.3 RTP Video Encodings

Payload format specification documents define how a particular payload is to be carried in RTP. Currently, payload format specification RFCs exist for H.261 video streams [RFC2032], for CellB video encoding [RFC2029], for JPEG-compressed video [RFC2035], and for MPEG video [RFC2038].

Standard payload formats for RTP are [12]:

- CelB
- JPEG
- H261
- H263
- MPV
- MP2T

All of the above video encoding standards use an RTP timestamp frequency of 90,000 Hz which results in exact integer timestamp increments for the typical 24 (HDTV), 25 (PAL), and 29.97 (NTSC) and 30 Hz (HDTV) frame rates and 50, 59.94 and 60 Hz field rates. Although 90,000 Hz is the recommended rate for the video encodings used within this profile, other frequency rates can also be used. However, a frame rate between 15 and 30 Hz is not suggested because it does not provide sufficient resolution for typical synchronization requirements when calculating the RTP timestamp corresponding to the NTP timestamp in an RTCP SR packet. The timestamp resolution must also be adequate for the estimation of jitter in the receiver reports.

RTP timestamp encodes the sampling instant of the video image enclosed in the RTP data packet for most of these video encodings. Packets from different video images are differentiated by their timestamps. If a video image has more than one packet the timestamp remains the same for all of those packets. It may also require that for most of these video encodings that for the last packet of a video image, the marker bit of the RTP header should be set to one otherwise it should be set to zero. It will tell the receiver that it was the last packet of the image and it is not necessary to wait for a packet with a different timestamp.

to detect that next a new frame should be displayed. Most of these video encodings also specify that the marker bit of the RTP header should be set to one in the last packet of a video frame and otherwise set to zero. Thus, it is not necessary to wait for a following packet with a different timestamp to detect that a new frame should be displayed.

CellB

The CELL-B encoding is designed by Sun Microsystems. It is a variable bit-rate coding scheme. Cell image compression algorithm provides high quality, low bit-rate image compression at low computational cost. Cell encoder produces a byte stream that consists of instructional codes and information about the compressed image.

According to RFC 2029 – “RTP Payload Format of Sun's CellB Video Encoding” currently, there are two versions of the Cell compression technology: CellA and CellB. CellA is primarily designed for the encoding of stored video which are intended for local display. While CellB which is derived from CellA, has been optimized for network-based video applications. CellB is computationally symmetric in both encode and decode. To achieve compression, it makes use of a fixed colormap and vector quantization techniques in the YUV color space.

JPEG

The JPEG encoding is specified in ISO Standards 10918-1 and 10918-2. JPEG is the image compression standard developed by the Joint Photographic Experts Group for continuous-tone, still images, both grayscale and color. It works best on natural images (scenes). This still image compression can also be applied to video. It compresses each frame of video as an independent still image and transmits them in a series resulting full motion video. Video coded in this way is often called Motion-JPEG.

The JPEG standard defines four modes of operation: the sequential DCT mode, the progressive DCT mode, the lossless mode, and the hierarchical mode (see details in chapter 5). Image is scanned in one or more passes depending on the mode of operation used. Each such pass is called a frame. This frame is further broken down into one or more scans. Within each pass there are one to four components, which represent the three components of a color (red, green, blue) for a colored image or a luminance signal and two chrominance signals for a grey scale image. These components are either encoded as separate scans or interleaved into a single scan.

In case of JPEG, information about compression parameters like quantization tables and Huffman coding tables is contained in a header which leads each frame or scan. These headers and optional parameters are identified with markers and encompass a marker segment.

H261

H.261 is a video coding standard proposed by ITU. It was designed for the data rates which are multiples of 64Kbit/s, and therefore is sometimes called $p \times 64\text{Kbit/s}$ (p is in the range 1-30). These data rates go well with ISDN lines, for which this video codec was originally designed. H.261 transports video data stream as payload data within the RTP protocol, with any of the underlying protocols on which RTP runs.

The coding algorithm of H.261 is a mix of inter-picture prediction, transform coding, and motion compensation. The data rates are able to be set to between 40 Kbits/s and 2 Mbits/s. In Intra coding, 8×8 pixels blocks are encoded with reference to each other. While in Inter coding frames are encoded with respect to another reference frame. Temporal redundancy is removed through inter-picture prediction while spatial redundancy is removed through transform coding. To remove any further redundancy, variable length coding is used.

H.261 supports two image resolutions, QCIF (Quarter Common Interchange format) which is 144×176 pixels and CIF (Common Interchange format) which is 288×352 .

The following fields of the RTP header are specified:

The *payload type* should specify H.261 payload format.

For H.261, the *RTP timestamp* is based on a 90,000Hz clock. This clock rate is a multiple of the natural H.261 frame rate. For each frame time, the clock is just incremented by the multiple to remove inaccuracy in timestamp calculation. Initial value of the RTP timestamp is chosen randomly to avoid known-plaintext attacks on encryption. The RTP timestamp is set for the first video image contained in the RTP packet. If a video image is contained in more than one RTP packets then the same RTP timestamp is set for all of those packets. RTP packets of different video images must have different timestamps. This helps to differentiate between the packets of different video images.

For H.261, the *marker bit* of RTP header is set to one for the packet containing last sampling instance of a video image to inform receiver that next is the packet for new image otherwise it must be zero.

H263

The H.263 encoding is specified in the 1996 version of ITU-T Recommendation H.263, "Video coding for low bit rate communication". Its packetization and RTP-specific properties are described in RFC 2190.

It was designed for low bit-rate communications; it never really worked well over POTS (plain old telephone service) lines [ITU H.263 Video Compression]. H.263 is based on the ITU-T Recommendation H.261 and has replaced H.261 for video conferencing in most applications. It also dominates Internet video

streaming today. H.263 employs similar techniques as compare to H.261, to reduce both temporal and spatial redundancy, but there are several major differences between the two algorithms that affect the design of packetization schemes significantly.

H263-1998

The H.263-1998 encoding is specified in the 1998 version of ITU-T Recommendation H.263, "Video coding for low bit rate communication". Its packetization and RTP-specific properties are described in RFC 2429.

Since H.263-1998 is a superset version of the above mentioned H.263, its payload format can also be used with the above mentioned version of H.263, and is recommended for this use by new implementations. This payload format does not replace RFC 2190, which is still used in the existing implementations while the implementations which use the new features of the H.263-1998 must use the payload format proposed in the RFC 2429.

MPV

MPV chooses the use of MPEG-1 and MPEG-2 video encoding elementary streams as specified in ISO Standards ISO/IEC 11172 and 13818-2, respectively. The RTP payload format is as specified in RFC 2250, Section 3.

To restrict the selection of the payload type of MPEG video, the MIME registration for MPV in RFC 3555 specifies a parameter that may be used with MIME or SDP. Applications which use this media type include audio and video streaming and conferencing tools.

MPV video format is mostly used for burning to digital media (read SVCD, KVCD, KDVD for further details).

MP2T

MP2T selects the use of MPEG-2 transport streams, for either audio or video. The RTP payload format is described in RFC 2250.

MP2T is encoding for compressed video and audio data multiplexed with signaling information in a serial bit stream. MP2T format was primarily developed for the transmission of compressed television programs via broadcast, cablecast, and satellite, and afterward adopted for DVD production and for some online delivery systems.

Idea behind this research is to study the possibility of using RTP to send video along with the text data (which is not a continuous media type) in real time scenario. Furthermore, it will review several possibilities which have been developed recently by the Internet community for the transmission of time-critical data to send video along with the text.

One motivation for this separation is to allow some participants to receive only one medium if they choose. Despite the separation, synchronized playback of a source's video and display of text can be achieved using timing information carried in the RTCP packets for both sessions.

1.4 Problem Identification

Most applications require transmission and reception of continuous media type (video) along with the non continuous media type (text) in real time and synchronization of both types of data on receiving end. For example, televideoconferencing tools require transmission of three types of data; audio, video and text. Transmission of simulation data may also require sending text data along with other media types. Text data in these cases may be the coordinates of the mouse or positions of the pointers. News agencies and TV are other candidates of the transmission of real-time video along with text. Video frames are captured at the site of the event and then transmitted to a number of locations. But besides the video, they also need to transmit text as a commentary for this video. This is then broadcast in case of TV and multicast in case of news agencies

RTP is the protocol designed for the transmission of real time contents but currently RTP provides support for the combined transmission of continuous media streams that are audio and video and not the text. Therefore, current possible strategy to send both types of real-time data types may be to transmit real time video data using real time transport protocol and text data through TCP, UDP or any other mean. Synchronization of both types of data received through two different means on receiving end is then a problem. This approach does not suit to real-time application for which time frame is most crucial.

So there is a need to find a mean to transmit and receive both non continuous and continuous media types which meet the requirements of real-time applications and get the synchronization of both data types on receiving end.

1.5 Proposed Solution

In this research, we are focusing at the problem of transmission of text data, along with the video data. This would be helpful for the development of different applications like a tele-videoconference tool, transmission of simulation data including the transmission of the coordinates of the mouse or pointer positions in real time and News agencies or TV transmission applications.

Real time multimedia applications transmit audio and video contents over networks in real time and this transmission is mostly related to multicast. Combining these two features, RTP is the protocol that helps to formalize this concept. RTP is capable of transmitting the video and audio data in real time for unicast as well as for multicast. The current RTP format does not cater for video plus text. Therefore, this research is an attempt to make some modification in the packet format of RTP and make it possible the combined transmission of video along with the text using RTP. This scenario creates certain additional problems to be investigated. These are listed hereunder:

- a. What should be the format of Textual data ([1] suggests T.140). Transformation of textual data is to be used or not.
- b. Placing of textual data in the RTP header or in RTP payload.
- c. Placing textual and video data in separate RTP packets or use the same RTP packet to send both types of data.
- d. Transmitting text and video frames in the same RTP session or in separate sessions, which will require separate timestamps and sequence numbers for both media types.
- e. Taking care that the video quality is not impaired.
- f. Synchronization of different media types at the receiving end.
- g. Analyzing the systems that are optimal and also do not impair the quality.

Proposed approach for the proposed solution will be to create client server architecture, where server will be the sending process while client will be the receiving process. Server will take text and video input. Then the packetization of data will be done. Both types of data will be placed in the RTP packets. These packets will be then transmitted through IP network. On receiving side the RTP packets will be received. Video and text data placed in these packets will be decoded and synchronized using the timestamping and sequence numbers contained in the RTP packets. Then both types of data will be displayed on the screen simultaneously and synchronized.

For the proposed solution there is further a need to analyze, test and evaluate various possible solutions. But the strategy of the solution is decided as follows:

- a. For format of text data, first of all we shall cater for Unicode type of data. In addition we shall look at T.140 format [1]. It is already coded in UFT-8 format.
- b. A new packet format will be generated. We shall compare the three options and choose the most optimal one, 1) Existing RTP packet [2], 2) Existing text packet [4] or 3) A new packet format which will be a combination of the two packets.
- c. In order to reduce video data size, chose most suitable video encoding for video data. As H.263 is mostly used for RTP applications, this will be preferred. H.263 is a protocol used for creating a Codec (Coder-decoder) that is used for transmission of video. In H.263, the pictures are divided into groups of blocks (GOB) which are numbered according to the vertical scan of the picture. It can use the formats CIF (Common Intermediate Format), QCIF (Quarter CIF source format), Sub-QCIF, 4CIF, 16CIF. The blocks can be a consisting of $k \times 16$ lines where k depends on the picture format ($k=1$ for QCIF, CIF and sub-QCIF; $k=2$ for 4CIF and $k=4$ for 16CIF). The other one is MB, A macro block (MB) containing four blocks of luminance and the spatially corresponding two blocks of chrominance.

- d. It will be investigated that whether the compression will also be applied to the accompanying text or not.
- e. RTP does not guarantee reliable delivery of service. Packets could arrive in an order which is different than the one originally used. In addition, the jitter conditions could emerge that would also impair the transmission. But RTP has certain mechanism to cater for these problems and these will be used. The flags for H.263 features could also provide additional information for video streams.
- f. Finally, we shall use RTP for packetization of video and textual data, dispatch it on the network and then receive it on the other side, render the video data on the screen and the textual data will be displayed along with the video data. Synchronization of text and video data will be maintained through timestamping feature of RTP. It will provide an optimal and efficient means of transmission of such data.
- g. It may also enhance existing packet format which can then form recommendation for a new RFC.
- h. It may also be implemented for SRTP which is a secure version of RTP. The key creation and key distribution mechanism will be devised for SRTP. Symmetric encryption will be used. Session key and Salt key will also be used. The encryption will be carried out at the sender end and decryption at the recipient end.

1.6 Outline of the Thesis

In this research study we are going to propose a strategy to send text data together with the video data in real time which will be used for many real-time applications which need to transmit the text data along with the video data and get the synchronization of both types of data in real-time. This proposed solution is supposed to improve the efficiency of such real-time applications by decreasing the processing time and headache required for the synchronization of both types of data on the receiving end. The thesis consists of the following chapters:

Chapter 1

Contains the introduction to the research topic and motivations behind this research. It also discusses RTP, Real-time Transport Protocol used for the transmission of real-time contents which is focus of this research. It further discusses some of the video payload types of RTP video transmission. Then an overview of the problem and the proposed solution is given in this chapter.

Chapter 2

Contains the literature review which describes the existing techniques used for the transmission of data over network. These techniques include TCP, UDP and RTSP. It further discusses some of their limitations.

Chapter 3

Contains requirement analysis which in turn includes the problem analysis, functional and non functional requirements. This chapter further discusses the main points which are to be focused during this research study. It provides an outline of the research to be followed.

Chapter 4

Contains detailed discussion about proposed solution and the proposed architecture, its components. It further explains system structure and design which includes client server specification, control modeling and modular decomposition of the proposed architecture.

Chapter 5

Describes implementation details of the proposed architecture. It includes overview of the implementation environment, implementation details of text and video input, Packetization of both types of data, transmission and reception and in last display of both types of data. It also gives list of implementation tools used.

Chapter 6

Gives an overview of the testing environment, performance improvements achieved during the implementation which includes synchronized display of both types of data, buffering of text data, redundant text data, reading image data into memory for display.

Chapter 7

Concludes this research and also specifies some areas where further research can be done for the improvements.

CHAPTER 2

LITERATURE SURVEY

2. Literature Survey

2.1 Introduction

A lot of work has been done on the transmission of text contents (non continuous media type) and multimedia contents (continuous media type). Different protocols have been developed which can be used for the transmission of text and multimedia contents in real time. However, there is still a need of an efficient approach to carry combined transmission of real time text and video (multimedia) contents. Only a few approaches which are widely used for the transmission of data over network are discussed here.

2.2 Transport Techniques

This section will discuss the different transport mechanisms developed over the years which are being used for the transmission of text and video contents over the network.

2.2.1 Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) [19] is a connection oriented transport protocol which runs over Internet protocol (IP) in order to provide end to end delivery. TCP best suits the requirements of non real time textual data and is mostly used for its transmission. TCP is a reliable protocol. In TCP, reliability is achieved through sequence numbers and acknowledgments (ACK). At the receiving end sequence numbers are used to arrange the received packets correctly in order and to eliminate duplicated received packets. Positive acknowledgment (ACK) of each successfully delivered packet is received from the receiver. If the ACK is not received within timeout interval for any of the transmitted packet, that packet is retransmitted until its successful delivery. Checksum provides a mechanism to handle damage. Checksum is recalculated at the receiving end and every damaged packet is discarded and then is retransmitted again.

TCP also provides a mechanism for flow control to handle congestion. This is achieved through a "window". A "window" is returned with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. This gives an indication to the receiver about the allowed number of packets that the sender may transmit next time. Whenever congestion occurs, this window size is reduced. TCP is a connection oriented protocol so does not support multicast applications.

2.2.2 User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) [18] is used when reliable transmission is not necessary rather robustness is more important. UDP is a connection less transport protocol. Like TCP, UDP also run over IP to provide end to end delivery. It does not provide any mechanism to ensure ordered reliable delivery of packets of data as is provided in TCP through sequence

numbering. A 16-bit checksum of UDP provides a mechanism to handle packet damages. Checksum for each packet is calculated for every packet to be transmitted and is sent along with the packet. On receiving end this checksum is recalculated and is verified to check the delivery of packet without damage. This checksum procedure is the same as is provided in TCP. Its robust delivery makes it suitable for the transmission of real time multimedia contents. Unlike TCP, UDP supports multicast applications.

2.2.3 Real Time Streaming Protocol (RTSP)

The Real Time Streaming Protocol (RTSP) [3] is an application level protocol. RTSP provides a control over the delivery of real-time data. It enables controlled, on-demand delivery of real-time data. RTSP establishes and controls either a single or several time-synchronized streams of continuous media types such as audio and video. The sources of audio and video data may be live captured data or stored files. RTSP merely provides a control over streams; it does not deliver the continuous streams itself. So RTSP acts as a "network remote control" for multimedia servers. Although interleaving of the continuous media streams with the control stream is possible.

In RTSP no files or contents are stored at the receiver [8]. RealNetworks' RealPlayer is an example of an RTSP application. It provides play, fast forward, pause, and other controls. RealNetworks developed the protocol in conjunction with Netscape and submitted it to the IETF for standardization.

2.2.4 Real Time Transport Protocol (RTP)

The Real Time Transport Protocol (RTP) [2] is an end to end real time transport protocol primarily designed for applications like audio, video conferencing. As other general purpose transport protocols are unable to meet the requirements of the real time applications so IETF (Internet Engineering Task Force, AVT WG) defined this new protocol. It has been accepted as a standard for the real time multimedia transmission. RTP provides end-to-end network transport services suitable for applications transmitting real-time multimedia data, such as audio, video or simulation data over multicast or unicast network services. Those services include Identification of payload type, Sequence numbering, Time stamping and Monitoring QoS of data transmission. It provides support for the functions but does not restrict implementations. Applications can implement according to need. RTP consists of two closely linked parts. The real-time transport protocol (RTP) to carry real time data while RTP control protocol (RTCP) to monitor the quality of service and to convey information about the participants in an on-going session

RTP runs over any of the transport protocol like TCP, UDP but typically, RTP run on top of UDP to make use of its multiplexing and checksum services. Both these protocols together provide transport protocol functionality. RTP provides timestamping, sequence numbering, and other mechanisms to take care of the timing issues and in order delivery which best suit to the requirements of real time multimedia contents transmission and synchronization on receiving end. Sequence number helps to detect the packet loss and to keep the real time data in order. Timestamping also helps in the synchronization of the different media streams. The payload type identifier specifies the payload format as well as the encoding/compression

schemes. From this payload type identifier, the receiving application knows how to interpret and play out the payload data. RTP is primarily designed and used for the transmission of real time multimedia contents (continuous media types).

Later on RTP payload for text was proposed for RTP to carry real-time text conversation contents in RTP packets. A mechanism based on RTP is specified in [4]. This mechanism gives text arrival in correct order, without duplication, with detection and indication of loss packets through sequence numbering. . In order to reduce the risk of loss text data, it also includes an optional possibility to repeat data for redundancy. Buffering of input text data, increases payload size in a packet during the conversation. Text conversation may be used alone or in connection with other conversational Medias, such as video and voice, to form multimedia conversation services. Uniform management of text and other media can be achieved in, for example, conferencing systems, firewalls, and network translation devices by using RTP for text transmission in a multimedia conversation application [4].

Later on an RTP payload for text conversation interleaved in an audio stream has been described in [5]. This payload format for real-time text transmission described in this RFC is anticipated for use between Public Switched Telephone Network (PSTN) gateways and is given a name of audio/t140c. The audio/t140c packets are generally transmitted as interleaved packets between voice packets or other kinds of audio packets. The objective is to create one common audio signal in the receiving equipment to be used for alternating between text and voice. Each medium in a session usually maintains a separate RTP stream. To achieve synchronization of the text with other media packets, it is recommended that the streams must be associated when the sessions are established and the streams must share the same reference clock. RTP timestamps of the voice, text, or other audio packets is utilized in order to reproduce the stream correctly when playing out the audio.

2.2.5 Other Related Research Work

In [9] an RTP based chat program has been developed which shows the possibility of sending data from non continuous media (i.e., text) using RTP in real-time applications and to synchronize it with other continuous media (i.e., audio or video) on receiving end. Final objective of this research is to develop a teleconference application integrating three tools: one for audio, one for video and another for text. The intension behind this ongoing research is to use its results in a security and telesurveillance system in which non continuous media streams such as text data from alarm sensors and access control devices will be transmitted together with video and audio streams.

2.3 Limitations

The mechanisms used for the transmission of data over network mentioned in section 2.2 have some limitations which do not suit for the combined transmission of real time text data (non continuous media type) and video data (continuous media type).

- For real-time applications, reliability is not as important as timely delivery. So reliable transmission mechanism provided by retransmission in TCP is not desirable.

For example, in case of network congestion, some packets may get lost and the application may result in lower but acceptable quality. If the protocol insists on a reliable transmission, the retransmitted packets could possibly increase the delay, jam the network and eventually starve the receiving application. Thus reliable transmission is badly chosen for delay-sensitive data such as real-time text and video data.

- The TCP congestion control mechanisms decreases the congestion window abruptly when packet losses are detected. Multimedia applications have natural rates that cannot be suddenly decreased without starving the receiver.
- Mostly real-time multimedia applications are of multicast type, while the connection-oriented TCP does not allow multicast and therefore is not suitable for such applications.
- Although UDP supports multicast, UDP is not a reliable protocol and does not provide any mean to arrange packets in order on receiving end.
- RTSP is used for real time multimedia applications but it alone does not transfer the real time contents on receiving end. it merely provides a control over media streams.
- These other transport protocols do not contain the necessary timestamp and encoding information needed by the receiving applications in order to synchronize different media types and play out of that media types in real-time.
- RTP provides a mechanism to transmit real-time multimedia contents over network and synchronize them on receiving end. But currently it focus is continuous media types. Although a payload for text alone and a payload for text interleaved in audio stream have been specified in [4] and [5] but no payload for combined text and video has been discussed.

2.4 Summary

Although many different mechanisms have been proposed and designed for the transmission of continuous and non continuous media contents but no one of them alone meets the requirements of text along with the video contents over network in real-time. Some are

designed, and are suitable, for the text data that is non continuous media type and some are suitable for the transmission of multimedia contents. The need exists for the enhancements and improvements in these approaches. Even the smallest enhancements in the above mentioned approaches and or combinations of the above approaches will make a better solution for the combined real-time text and video transmission.

CHAPTER 3

REQUIREMENT ANALYSIS

3. Requirement Analysis

3.1 Introduction

This focus of research is the transmission of real-time video and text contents over the network. It provides means for capturing, transmitting and displaying video stream as well as textual data over the network employing Real Time Protocol.

The system is to be implemented as a client server model where server acts as a sender process, responsible for capturing video data through camcorder and text data through keyboard, sending both types of data over the network in RTP packets. On the other hand client is responsible for receiving both types of data and displaying video data along with the text data received in RTP packets. RTP/RTCP take cares of real time issues. RTP packet contains video data as well as text data while RTCP packets contain control statistics like packet loss, delay and inter arrival jitter etc.

3.2 Problem Analysis

3.2.1 Functional Requirements Definitions & Specifications

1. The operating system will be Linux.

- The system is to be implemented in GCC under Linux environment.
- Standard Linux Sockets will be used for Data Transmission.
- raw1394 module is facilitated in the Linux kernel as a module.

2. The Image is to be captured through FireWire card.

- The input source to the FireWire card is Sony Handycam (camcorder).
- The capture card is to be installed at server.
- Server computer should provide means to configure FireWire card with the help of raw1394 module and ohci1394 drivers.
- FireWire card proper installation and working is to be verified through *gscanbus* application.

3. LAN network based on TCP/IP consisting of at least two workstations.

- The two Linux computers server-client will be connected together through a network.
- The network card will be configured in Linux
- Data can be sent between the two computers through network cable.

4. Server is to facilitate the capture of video stream from the video source and text data through keyboard.

- Individual frames are to be captured using camcorder while text is to be entered through keyboard.
- Server will also display the captured frame which is to be transmitted next, through XWindows program.
- Both types of data are to be placed in RTP packets, having proper header and payload.
- The header of RTP packet will contain information as sequence number and timestamp; while payload has the video and text data.
- If a frame occurs to be in more than one RTP packets it will have same timestamp.
- Text will be transmitted after every 10 packets to facilitate buffering.
- Initially server will be in waiting state.
- Only when it receives a request of connection from client it goes into transmission state and sends RTP packets containing video and text data to the client.
- Sender report of RTCP packet and SDES of sender is to be generated by the server.
- Receiver report of RTCP packet is to be received from the client computer through open sockets.
- Similarly bye packet indication is also received from the client computer after which the transmission is stopped, client is exited and server again goes in waiting state.

- Keyboard input of 'e' makes client to exit by force after which the server again goes into waiting state.
- Keyboard input of 'q' makes both server and client to finish transmission and exit.

5. Captured video stream and text data will be transmitted to the client computer in RTP packets through open sockets.

- UDP sockets will be opened and binded between the two computers.
- Both server and client will be able to communicate on two ports at a time, one for RTP that is 5004 and one for RTCP data that is 5005.
- RTP packet is composed of RTP header and RTP payload i.e. video and text data.
- RTCP packet provides control information and statistics on quality of services.
- FDset will be introduced through which the input can be taken from all three, RTP port, RTCP port and keyboard.

6. The client will receive the video and text data through open sockets.

- The UDP sockets are to be opened and binded at client end.
- Over there text and video data is to be received through RTP port i.e. 5004.
- RTCP control information and statistics on Quality of services is to be received through RTCP port i.e.5005.
- FDset is to be introduced at client end where the data can be received on the three possible inputs that are RTP port, RTCP port and keyboard.

7. The client will depacketize RTP packets and display the text and video data on screen.

- The RTP packets on reaching client are depacketized and RTP header is to be removed from them.
- The video data is to be written to a memory buffer area.
- Video data is then read from the memory location, decompressed and

displayed on the screen.

- Text data is also to be displayed in the same window as used for video data.
- Client produces and sends RTCP information as that of SDES information of client, receiver report and bye packet to the server.
- Keyboard input of 'q' at client computer causes client to send a bye packet to server after which the transmission is stopped and client is exited.

8. The codec will be used for image while it is transmitted.

- Transmission of video on network requires some sort of codec through which video stream can be encoded and transmitted on network. At receiver end that video stream is to be decoded and put to some use. The codec compresses and decompresses the data by which network latency time does not cast any bad effects on data transmission in real time.

9. System will provide a user interface through which it will supply the statistics about packet loss, delay, time stamp and inter arrival jitter.

- The interface will be designed through curses library and will be presented at server computer only.
- RTCP reports are to be produced and displayed on server computer.
- SDES information of both server and client are put on show on the server screen.
- Sender report generated by server and receiver report received from the client will also be presented here.

3.2.2 Non-Functional Requirements

This project involves a number of integration issues that makes a no. of requirements that are not directly the part of my project.

Image capture: In order to send a video stream through RTP on the network I must have data source capturing image in real time. Though it could have also been done with a stored video file but then the real time element is eliminated. Therefore it was required that I be able to facilitate the capture of image in real-time.

Camera/Video card: The only way chosen to capture image in real-time is through a digital video camera and a video capture card. The camcorder to be used is a Sony camcorder. It also required getting a video capture card whose drivers are available for Linux platform. Handycam will able to capture the live image at the same time.

Image display: Another difficult part is display of video data. It requires finding a suitable way of display of video data for Linux platform. Two possibilities are either SVGA library of Linux or writing own program using X Windows.

Final date of submission: Restricts to complete this project within deadline. Therefore less time constraint also limits to do the project.

Efficiency: System should be efficient enough to display the video stream on client as soon as it is captured on server. In order to match the quick pace of display module, some delays need to be implemented.

3.2.3 Hardware and Software Requirements

1. Hardwre

FireWire card

Handy Cam

i.Link cable

Network card

PIV – server

PIV – client

2. Software

Fedora core 5 (OS) Kernel 2.6.15

Libraries

FireWire card drivers

3.2.4 Phases of the project

The whole project comprises of three major modules:

Phase 1: Image and text data input

Capture the image through camcorder, compress it into jpeg. Take text input from keyboard.

Phase 2: Networking

Packetize the video and text data and transmit it through RTP to the client, and also receive RTCP reports.

Phase 3: Displaying video and text data

Receive the RTP packet, depacketize, de-compress video data and then display both type of data on screen.

3.2.5 Scope of the project

- The system to be implemented is Unicast and does not support multicast i.e. Server-Client Model where only single server will send data to only a single client.
- Implementation of client and server is Half-duplex i.e. only in one direction – Server can only send data and client can only receive data.
- Implementation of video is Without Sound, i.e., transmission of video stream only and no audio issues are handled in it.
- System is implemented only on LAN. However it can be used on the Internet also.
- Since speed and RAM of my computer is insufficient to meet the requirements of a multimedia server, so it may or may not disturb video display results.
- The highest resolution of video display is 352x288.
- Since data is received from only one source, that is server, therefore for the implementation of rtp I am restricting only to one ssrc
- There is no mixers and no contributed sources i.e. csrc.
- The architecture used will be Intel PC not macintosh and Alpha etc.
- Operating system used is Linux.
- IPv4 is used and not IPv6.

3.3 Focus of Research

Focus of the research in the above mentioned requirements analysis would be the following points.

3.3.1 Transport Protocol

Which transport protocol to be used in order to provide transport functionality needed by RTP? Two major alternatives are TCP and UDP.

3.3.2 Packetization of Text and Video data in RTP packets

It will require finding a suitable solution to Packetize and send both types of data that is continuous and non continuous data in RTP packets. Whether it will be efficient to maintain a single session for both types of data or maintain two different sessions for the one for each. Single session requires maintaining same timestamps and sequence numbers for both types of data streams while separate sessions means maintaining separate timestamps and sequence numbers. Maintaining a single session will further have two possibilities. One is to send both types of data in separate RTP packets and synchronize them on receiving end. Second is to use the same RTP packet to send text as well video data. So the focus of research at this step would be to find an efficient approach for the packetization of both types of data.

3.3.3 Size of Text and Video Data in One Packet

To find the suitable size of transmission of both types of data in order to avoid bandwidth wastage and also in order to meet the maximum transmission size (MTU) requirements of the underlying transport protocol. It will require to adjust the size of the image as well as to set the size of the text input taken through keyboard. The input size of both types of data will be so adjusted to enable them to send in RTP packets. Size of the image is adjusted so that whether to send it in a single RTP packet or to send in more than one RTP packets. In that case the timestamp would be the same for all such packets containing parts of the same image. Compression algorithms can also be applied to reduce transmission size. So finding an appropriate compression is also a part of this research.

3.3.4 Transmission and Reception of RTP Packets over Network

Focus would be to find an efficient approach for the transmission and reception of both types of data.

3.3.5 Synchronization and Display of both types of Data

This is the most important part of this research study. Synchronization of both types (continuous and non continuous) of data in real-time on receiving end is the major point of concern in this study. RTP provides timestamping and sequence numbering which help to synchronize different media types on receiving end. These both will be used to synchronize text and video data. So focus of research would be to find an efficient way to synchronize both data types of data and display them in real-time.

3.4 Summary

Focus of this study is to find an efficient way of transmission of text along with the video data in real-time. This will require studying all alternatives carefully and chose those which are the most appropriate. For example how to packetize text and video data in RTP packets, what should be the size one data in one RTP packet, how data will be synchronized and displayed on receiving end. These are the requirements of this research which are to be investigated.

CHAPTER 4

SYSTEM DESIGN

TH-5069

4. System Design

4.1 Introduction

The basic focus of this study has been the combined transmission of real-time text and video contents over the network and their synchronization on receiving end. Its various aspects have been studied, keeping in mind the applications like tele-videoconferencing, transmission of simulation data and TV and News agencies softwares and tried to find out the most efficient approach.

4.2 Proposed Solution

In this proposed solution, an effort has been made to decrease the over head of sending real-time text and video contents separately using two different protocols and on receiving end synchronize both types of data received through two different transport mechanisms. Video data is captured using camcorder, while the text input is taken from keyboard, entered by user.

4.2.1 Components of the proposed solution

- **Transport protocol to be Used**

The alternatives for transport protocols are TCP and UDP. The transport protocol selected is UDP for the following reasons.

- Retransmission mechanism of TCP causes and end to end delay which does not suit for the real-time applications.
- Congestion control mechanism of TCP suddenly decreases the window size which causes starvation.
- TCP does not provide support for multicast while UDP provides
- TCP header is larger than a UDP header (40 bytes for TCP compared to 8 bytes of UDP).

- **Packetization of data**

Packetizaion of data includes how to send both types of data in RTP packets. Among the three possibilities mentioned in section 3.3, the chosen one is to maintain a single session for both types of data streams and maintained same timestamps and sequence numbers for both data streams. A single RTP packet is used to send both types of data. It means pack both types of data in a single RTP packet rather than using separate packets, one for text and one for video. The reason behind this selection is that as text input is taken through keyboard, the rate

of character entry is usually at a level of a few characters per second or less. Even with a maximum typing speed, rate of character entered is a few characters per second or less than it. In worst it may be a single character or not even a single character. Which means text data available for transmission is only a few bytes or may be a single byte. On the other hand, minimum size of RTP header is 12 bytes, while that of UDP is 8 bytes. So sending separate packets for few bytes of text data requires sending minimum 20 bytes of header data with each packet. This is not an efficient approach rather is wastage of bandwidth. So it was selected to send text data in the same RTP packet used for video data. It required modifying the current RTP packet format. Currently RTP packet contains video data only, using it for text also requires adding new fields in it. So the ultimate result was a modified packet format.

- **Size of both types of data to be transmitted in one packet**

IPv4 (Internet Protocol version 4) uses 32 bit addresses. IPv4 provides packet delivery service for TCP, UDP, ICMP and IGMP. The maximum transmission size of an IPv4 datagram for UDP is 65535 bytes, including IPv4 header. The image size has been adjusted so that to meet the maximum transmission size of UDP datagram.

- **Transmission and reception of RTP packets over network**

RTP stack used is RTP/UDP/IP that is RTP data is encapsulated in RTP packet and RTP header is attached, then on transport layer UDP header is attached and then IP header is attached.

- **Synchronization and display of both types of data**

Synchronization of both types of data is achieved through time stamping and sequence numbering.

4.3 Proposed Architecture

A client-server architectural model is followed in implementation i.e. a distributed system model showing how data and processing is distributed across a range of processors. As the focus of the implementation is only a single server (sending process) and a single client (receiving process), so the whole processes are divided just between the two. A server, who will capture the video data and text data, packetize it into RTP packets, transmits it on network. A client, who receives RTP packets, obtains both types of data, synchronize it using time stamps and sequence numbers and display it. Whole system is implemented using C/C++.

Following figure shows the block diagram of purposed architecture.

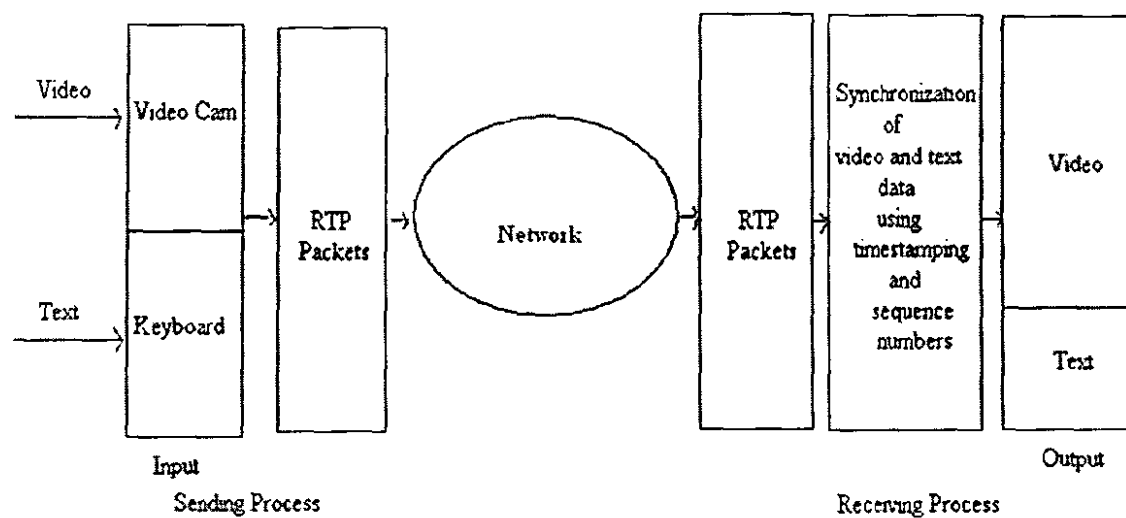
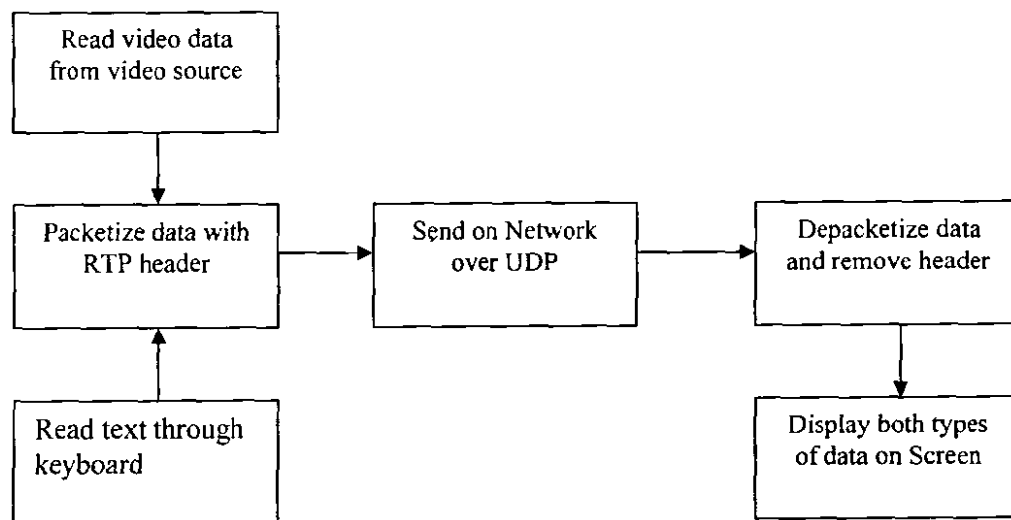


Figure: Proposed Architecture

4.3.1 System Structure

System is structured into a no. of principal subsystems as follows:



4.3.2 SERVER-CLIENT Specifications

- Reference: **SERVER**
- Events: Start Transmission
- End Transmission
- Server Exit
- Services: Capture video from video input source
- Text input from keyboard
- Produce and display Sender Report (SR)*
- Send RTP packets
- Receive RTCP packets from client, including SDES of client, Receiver report, Bye packet if any
- Display RTCP info on screen

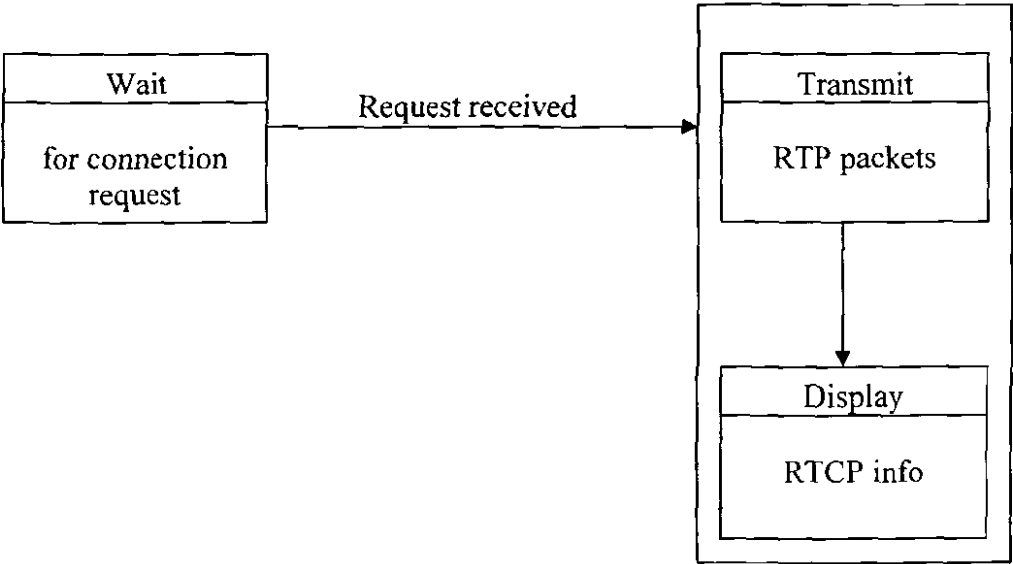


Figure: Server – Start Transmission

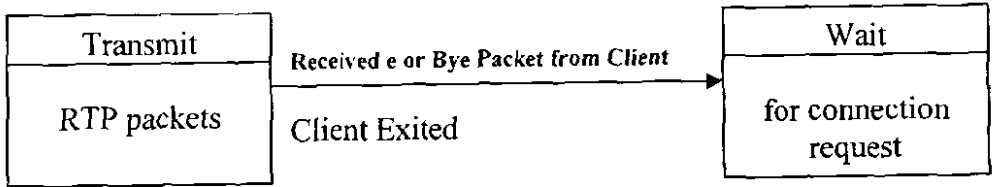


Figure: Server – End Transmission

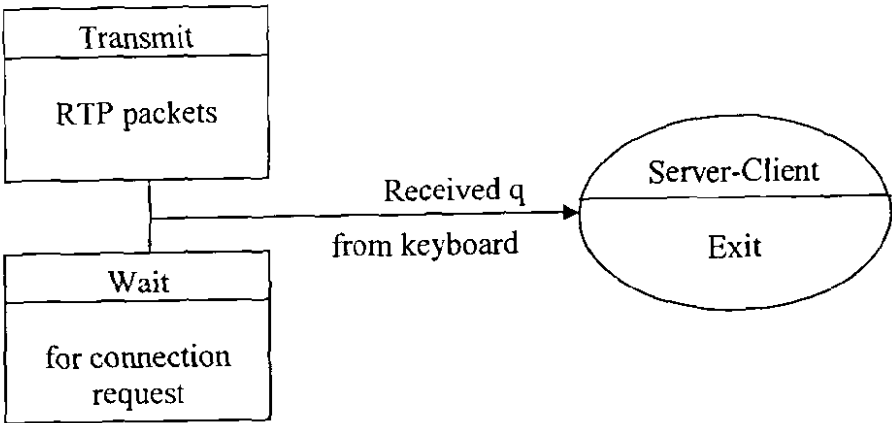


Figure: Server-Client Exit

- Reference: CLIENT
- Events: Send Request of Connection to server
End Transmission
- Services: Receive RTP packets from server
Display video and text on screen
Produce Receiver report and SDES of client
Send RTCP packets to server

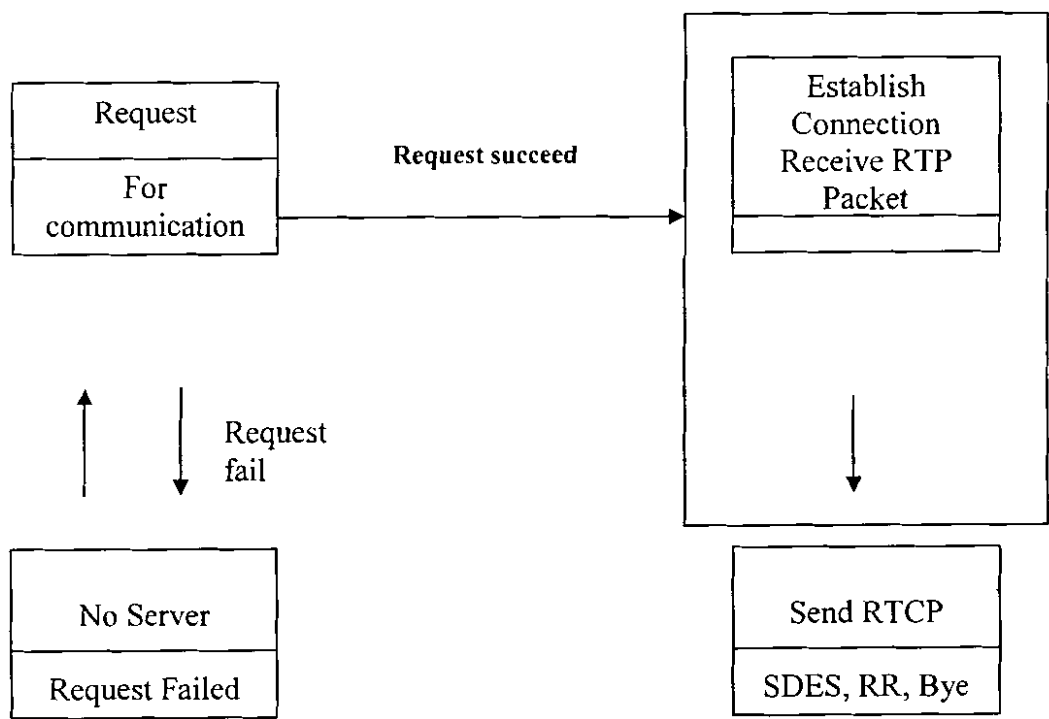


Figure: Client – Start Transmission

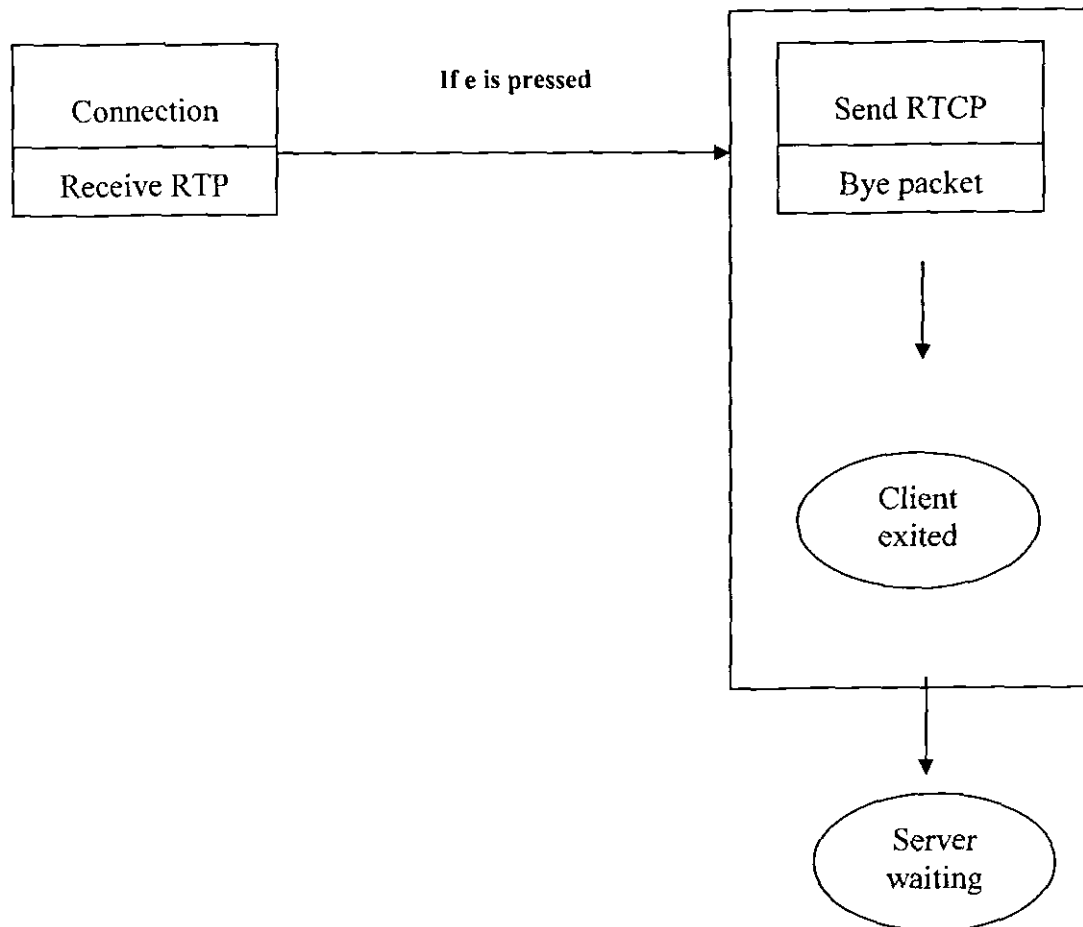


Figure: Client end transmission – server again waiting

4.3.3 Control Modeling

There is a *centralized control* implemented in this project. That is the server has the central control of sending the data to client. During transmission server can force the client to exit or can quit itself making the client exit as well at the same time. Server is responsible to capture video stream and text data and transmit it over the network. Furthermore the server also displays the RTCP information on its own end.

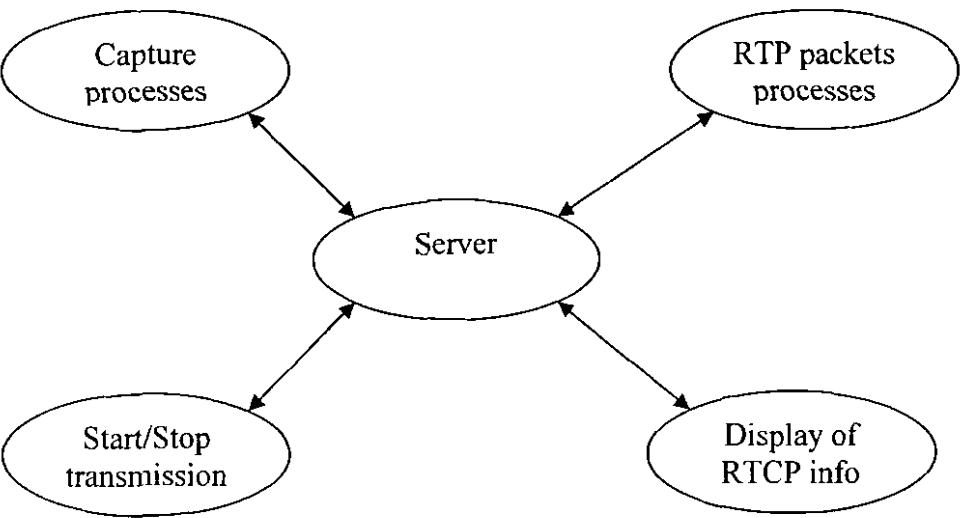


Figure: A centralized control model for a real time video plus text server

Event-based Control: Transmission of video and text data is event based i.e. initially server is in waiting state, when the client initiates a request, the transmission takes place. After every 10 sec the RTCP data is sent from the client to the server and reports are updated at server end. Client can send bye packet to the server computer to end transmission, it can do so by pressing b and hitting enter key. The client can exit and stop receiving network data by pressing q and hitting enter. The client is also responsible for displaying video and text data on receiving it from the server computer.

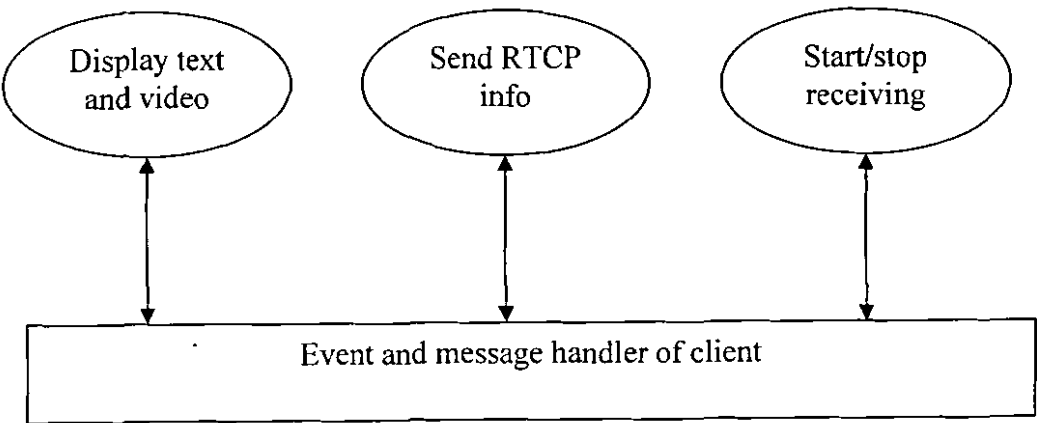


Figure – A control model based on selective event handling

4.3.4 Modular Decomposition

Each subsystem is decomposed into a number of modules that are as follows:

Server:

1. Establish/open sockets on two ports 5005 for RTCP and 5004 for RTP
2. Wait for connection from client and on connection receive SDES packet
3. Open video device and capture frame
4. Compress frame and put it into RTP packet
5. After 10 packets of video take text input entered through keyboard other wise move to step 8
6. Put text data in RTP packet along with the video data and set flag for text
7. Store same text data in a buffer to send it again in exactly next packet to implement redundancy
8. Attach RTP header
9. Allot sequence number and time stamp and send data to client
10. After every 10 sec display stats on screen: SR,RR,SDES
11. On receiving bye msg from client, display bye packet along with the source of Bye packet and reason to go.

Client:

1. Establish sockets
2. Send SDES packet to server.
3. Receive each packet
4. Depacketize rtp packet and remove rtp header from frame.
5. Decompress video frame
6. Take text from packet if it is present in it by checking text flag otherwise move to step 9
7. Check flag value: if 0 then original text otherwise repeated text
8. Check if repeated text is already received then discard this text otherwise read it for display
9. Display video frame and text data (if it is available) on screen of client

10. Concatenate the new received text (if text is received) with the previously received text for display
11. Send RTCP reports after every 10s
12. When finished send bye packet to server.

4.4 Summary

This chapter describes the components of proposed solution of sending real-time text and video data using RTP. It also discusses the proposed architecture which is client server architecture and its design in detail which include its systems structure, client and server specifications, control modeling and modular decomposition of the whole structure. Using this proposed solution for combined transmission of real-time text and video data, the processing time and overhead of synchronization of both types of data on receiving end is reduced, that enforced a great cost for such real-time applications. The RTP packet format is modified to accommodate both types of data in RTP packets which provides efficient processing of such real-time applications.

CHAPTER 5

IMPLEMENTATION

5. Implementation

5.1 Introduction

The proposed architecture, to test and verify the proposed solution and the improvements made by this proposed method, has been developed using C and C++. The application has been developed using object oriented implementation.

5.2 Implementation Details

5.2.1 Overview of the Implementation Environment

The environment which is created for the implementation is a client server model. Video input is taken through camcorder which is attached at the server computer. A separate application is used to grab video frames through camcorder. Compression is applied to video data. Text input is taken through key board on server side. Then text and video data is put into RTP packets and is transmitted to client. But before this packing and transmission, both types of data, that are to be transmitted right now, are displayed on server side as well. This display of data on server side is merely to verify the synchronization achieved on the client side. Client side receives both types of data packed in RTP packets. Read both types of data. Decode video data and display both types of data in a synchronized manner. Following are the major components and steps of the implementation procedure.

5.2.2 Image capture

First phase of the implementation is taking video input that is image capture. Input source used to take video input was camcorder. Image capture phase further has some steps.

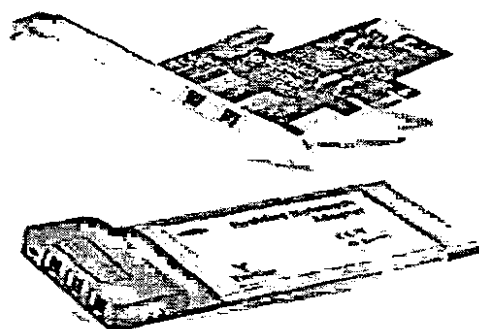
5.2.2.1 Attaching Camcorder with PC

Following are the possible two ways to attach camcorder with PC in order to take video input.

IEEE 1394 (FireWire Interface)

The IEEE 1394 interface is a serial bus interface standard for high-speed communications and isochronous real-time data transfer, frequently used in a personal computer. The interface is also known by the brand names of FireWire (Apple Inc.), i.Link (Sony), and Lynx (Texas Instruments). It is used in digital camcorders because it is capable of quickly transferring full-motion video. Most consumer video equipment uses 4-pin FireWire

ports and connectors, but some also use a 6-pin FireWire configuration. FireWire card is a plug and play and is automatically detected when inserted in PCI slot. The following figure shows 2 connectors and 3 connectors FireWire Cards.



FireWire-adapter-cards

With a 2 connectors, one can attach two devices with the computer at a time and with a 3 connectors, 3 devices can be attached at a time. The core component of a FireWire is an expansion card and a cable. FireWire card is plug_and_play and is plugged into a PCI slot. It is automatically detected when inserted into PCI slot.

USB Interface

USB (Universal Serial Bus) is a plug and play interface between a computer and peripherals which allows plugging in a device without adding an adapter card or even restarting computer. This is the easiest way to capture video to a computer. USB 2.0 is very fast, maximum up to 480 Mbps.

Why to prefer FireWire Interface?

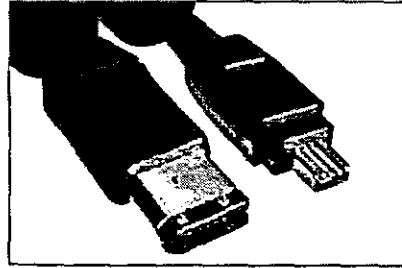
Because of technical differences, FireWire is a better way for transferring uncompressed (raw) video from digital camcorders to PCs, even though USB 2.0 has a higher maximum speed (400Mbps vs. 480Mbps). FireWire interface is the best solution for streaming video for many reasons. One of the reasons is it enables faster transfer of high resolution video data as compare to USB interface. Another reason is software solutions are available for streaming through camcorder using FireWire card for Linux platform like Kino, Dvgrab etc. while no such solution is available for USB interface. So FireWire interface is used for streaming through camcorder.

FireWire connecting cables

Two types of connecting cables available in market with which to connect different FireWire devices with PC are as follows :

4pin-to-4pin plugs – This is used between camcorder and computer, where the FireWire (i.Link) socket is of smaller type (picture left).

4pin-to-6pin plugs – This is used between camcorder and computer, where the sockets are of the 6-pin type (picture right).



5.2.2.2 Drivers for FireWire Card

Drivers for FireWire card to interact with `ieee1394` subsystem

- `raw1394`
- `ieee1394`
- `ohci1394`

Drivers' hierarchy

The GNU/Linux IEEE-1394 Subsystem is divided into three layers. The core of the entire 1394 subsystem is the module `ieee1394`. It manages all high- and low-level drivers in the subsystem, provides basic services like handling of the 1394 protocol, collecting information about bus and nodes. Below the `ieee1394` module are the low-level (hardware) driver modules, which handle converting packets and bus events to and from hardware accesses on specific 1394 chipsets.

Above the core (`ieee1394` module) are the high level driver modules, which use the services provided by the core to implement protocols for certain devices and act as drivers to these. One such driver is `raw1394`, which is designed to accept commands from user space to do any transaction wanted (as far as possible from current core design). Through `raw1394`, user applications can access 1394 nodes on the bus and it is not necessary to write kernel code just for that. To access the raw 1394 bus from user application, `libraw1394` is used. User application is linked with `libraw1394`, which handles the communication with the `raw1394` high-level driver.

The following figure shows the driver hierarchy.

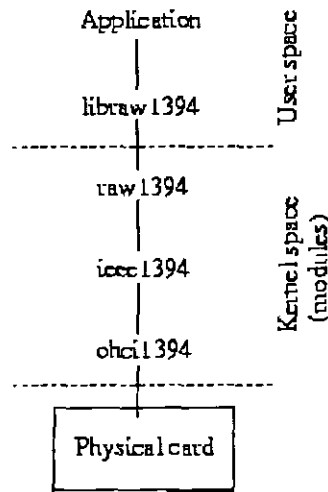


Figure: The driver hierarchy in a simple setup.

There are three low-level drivers:

aic5800	Adaptec AIC-5800 PCI-IEEE1394 chip driver
pcilynx	Texas Instruments PCILynx driver
ohci1394	1394 Open Host Controller Interface driver

An OHCI compliant FireWire card is used for the implementation, so ohci1394 module is used to interface the 1394 card. It is also possible to have all three low-level driver modules loaded and active at the same time. All the low-level drivers can control more than one card.

5.2.2.3 Installation and Configuration of System

Following are the steps followed to install and configure FireWire card to capture video on fedora core 5. Detailed steps to install FireWire for different kernel versions of Linux are given on the web site of IEEE 1394 for Linux [6].

Installation

1. Install the adapter card into computer's PCI slot and make sure everything is still working before continuing.
2. Download libraw1394.
You can save this where you choose—your home directory, /tmp, /usr/src, etc.
3. Compile libraw1394:

```

cd /where/you/downloaded/libraw1394
tar xvfz libraw1394-1.2.1.tar.gz
cd libraw1394-1.2.1
./configure

```

- ```

make
make install
4. Create the raw device (/dev/raw1394):
 make dev
5. Reboot:
 shutdown -r now
6. After the system reboots, logon and load the modules:
 modprobe ohci1394
 modprobe raw1394
 modprobe ieee1394

```

Module dependencies should ensure that the `ieee1394` subsystem module installs automatically. If you receive errors about *unresolved symbols*, then try the following:

```

insmod ieee1394
insmod ohci1394
insmod raw1394

```

### Testing

1. View messages using `dmesg`:  
`dmesg | pager` or `dmesg | ieee1394`

Look for lines beginning with "`ieee1394`" or "`ohci1394_0`." Do not be alarmed by the number of messages or messages with the words "error," "timeout," or "inconsistent" in them because often these are not indicators that the initialization has failed! Identify one or more messages that read "`ieee1394: Host added: Node [...]`" These messages indicate the detection of each host adapter and its acceptance by the `ieee1394` subsystem.

2. Run `libraw1394-1.2.1/src/testlibraw`. It performs simple tests on all detected nodes.
3. Download `gscanbus`. [To check the proper installation of FireWire]
4. Unpack `gscanbus`:  
`tar xvfz gscanbus-0.7.1.tgz`
5. Compile `gscanbus`:  
`cd gscanbus-0.7.1`  
`./configure`  
`make`
6. Run `./gscanbus`. If you are using X windows, then run `gscanbus` from a terminal window to view errors and warnings. If you are experiencing errors on any of the test steps, then run `gscanbus` with the `-v2` switch to get more verbose error reporting. `gscanbus` opens a window displaying each node as an icon. You can click an icon to get more information about the node.
7. Without rebooting, plug in and/or turn on any device connected to your host adapter.
8. Repeat steps 1, 2, and 6.

This was the process to install and configure the FireWire card. It was a hit and trial process for me. But ultimately I succeeded to install FireWire card.

#### 5.2.2.4 Image Capture through Camcorder

After the installation and configuration of FireWire card, next step was video capturing from camcorder to PC. For which video capturing software was required. Kino and dvgrab are the two softwares, which are both used for image grabbing for Linux distributions. These both are developed by the same team and are available under GPL to further use and modify. Kino is similar to dvgrab but with enhanced functionalities of video editing. As only it was needed to capture images and store them on disk, so dvgrab was the best solution. Its binary code was downloaded from internet and modified according to the need of implementation. It also required installing required libraries needed to work with dvgrab. Finding a proper combination of these libraries was a tough job. Libraries used for this application are:

➤ libraw1394

Summary: Streaming library for IEEE1394

➤ libavc1394

Summary: FireWire AV/C interface

➤ libiec61883

Summary: Streaming library for IEEE1394

➤ libdv

Summary: provides codecs

#### libraw1394 functions used to access raw1394:

##### 1). raw1394\_new\_handle

**Name:** raw1394\_new\_handle -- create new handle

**Synopsis:** raw1394handle\_t raw1394\_new\_handle (void);

**Arguments:**

*void* -- no arguments

**Description:**

Creates and returns a new handle which can (after being set up) control one port. It is not allowed to use the same handle in multiple threads or forked processes. However, it is allowed to create and use multiple handles. Use one handle per thread which needs it in the multithreaded case.

**Returns:**

It returns the created handle or NULL when initialization fails. In the latter case `errno` either contains some OS specific error code or 0 if the error is that libraw1394 and raw1394 don't support each other's protocol versions.

**2). raw1394\_destroy\_handle**

**Name:** raw1394\_destroy\_handle -- deallocate handle

**Synopsis:** void raw1394\_destroy\_handle (raw1394handle\_t handle);

**Arguments:**

*Handle* -- handle to deallocate

**Description**

It closes connection with raw1394 on this handle and deallocates everything associated with it. It is safe to pass NULL as handle; nothing is done in this case.

**3). raw1394\_get\_port\_info**

**Name:** raw1394\_get\_port\_info -- get information about available ports

**Synopsis:** int raw1394\_get\_port\_info (raw1394handle\_t handle, struct raw1394\_portinfo \* pinfo, int maxports);

**Arguments:**

*handle* -- libraw1394 handle

*pinfo* -- pointer to an array of struct raw1394\_portinfo

*maxports* -- number of elements in *pinfo*

**Description**

Before we can set which port to use, we have to use this function to find out which ports exist. If the program is interactive, the users are presented with this list to let them decide which port to use if there is more than one. A non-interactive program (and probably interactive ones, too) should provide a command line option to choose the port. If *maxports* is 0, *pinf* can be NULL, too.

### Returns

It returns the number of ports and writes information about them into *pinf*, but not into more than *maxports* elements.

#### 4). raw1394\_set\_port

**Name:** raw1394\_set\_port -- choose port for handle

**Synopsis:** int raw1394\_set\_port (raw1394handle\_t handle, int port);

### Arguments:

*handle* -- libraw1394 handle

*port* -- port to connect to (corresponds to index of struct raw1394\_portinfo)

### Description

This function connects the handle to the port given (as queried with raw1394\_get\_port\_info). If successful, raw1394\_get\_port\_info and raw1394\_set\_port are not allowed to be called afterwards on this handle. To make up for this, all the other functions (those handling asynchronous and isochronous transmissions) can now be called.

### Returns

It returns 0 for success or -1 for failure with errno set appropriately. A possible failure mode is with errno = ESTALE, in this case the configuration has changed since the call to raw1394\_get\_port\_info and it has to be called again to update your view of the available ports.

#### 5). raw1394\_set\_bus\_reset\_handler

**Name:** raw1394\_set\_bus\_reset\_handler -- set bus reset handler

### Synopsis :

```
bus_reset_handler_t raw1394_set_bus_reset_handler (raw1394handle_t handle,
bus_reset_handler_t new_h);
```

### Arguments

*handle* -- libraw1394 handle

*new\_h* -- Pointer to new handler

### Description

It sets the handler to be called on every bus reset to *new\_h*. The default handler just calls `raw1394_update_generation`.

### Returns

The old handler

## 6). raw1394\_get\_fd

**Name:** `raw1394_get_fd` -- get the communication file descriptor

**Synopsis:** `int raw1394_get_fd (raw1394handle_t handle);`

### Arguments

*handle* -- libraw1394 handle

### Description

This can be used for select/poll calls if you wait on other fds or can be integrated into another event loop (e.g. from a GUI application framework). It can also be used to set/remove the `O_NONBLOCK` flag using `fcntl` to modify the blocking behavior in `raw1394_loop_iterate`. It must not be used for anything else.

### Returns

The fd used for communication with the raw1394 kernel module.

`dvgrab` was used to grab video frames from camcorder and save on computer disk on server side. `dvgrab` receives audio and video data from a digital camcorder via an IEEE 1394 (widely known as FireWire) link and stores them into some defined format file by default into an AVI file. It supports saving the data as raw frames, AVI type 1, AVI type 2, Quicktime DV, or a series of JPEG stills (which is utilized to implement M-JPEG in my case).



### 5.2.2.5 Video Encoding Used

Frames obtained through camcorder were in raw format which produces a large amount of data. To transverse network optimally, video data size should be smaller enough. As we know digital video files tend to have a larger size. For an efficient transmission file size is an important concern. So to reduce the transmission size of video, the only solution is compression.

Specifications of this research did not include writing any codec or compression mechanism for video data. Because writing a compression algorithm is in itself a complete project. It was proposed to use H.263 codecs for video data, which is CCITT standard for videoconferencing applications based on RTP. But as focus of this research was transmission of real time data for applications such as news agency applications, live TV broad cast, and transmission of simulation data along with the coordinate point data etc. rather than videoconferencing applications. So it was further studied which video format is most suitable for the implementation of this proposed solution. After studying different video formats supported by RTP vide types it was got to know that rather than H.26x video codecs M-JPEG is more suitable for the purpose of focused applications for the following reasons.

- "Temporal compression" techniques (MPEG4, H.263 and H.264) provide advantages over "frame-by-frame compression" techniques (MJPEG) when the background is fixed and there is not a lot of motion in the scene. In these applications, they can really compress images (small file sizes) with very little loss of image quality like video conferencing applications. Temporal compression also makes it easier to synchronize audio with video. However, temporal compression offers little or no advantage over frame-by-frame compression when there is a lot of motion in the scene or when the background is changing [13]. Focus of my research is applications like TV and News agency transmission applications where background is not fixed.
- The video compression algorithms have conventionally developed for constant bit-rate channels. However in case of heterogeneous packet networks like the Internet, packet losses are not rare and loss patterns may be bursty. In case of MPEG and H.261/ H.263, the packet loss causes a significant degrade in quality due to their method of removing temporal redundancy. Both MPEG and H.261 rely on intra frames to eventually resynchronize, but at low bit rates the resynchronization intervals can be too broad and decoded bit stream may virtually never be error free. This can not be tolerated in case of live broadcast of media contents. So the solution is to reduce the resynchronization interval. This scheme is used in M-JPEG where each frame is coded independently. However, this approach results in low compression because of redundant information [14].

- In Motion JPEG, each video frame is compressed separately using the JPEG still image compression standard. Frame differencing or motion estimation is not used to compress the images. This makes frame accurate editing without any loss of image quality during the editing possible (Suitable for TV and News agency who need to edit their video clips before broadcast or multicast) [15].
- Another reason was that the source code of JPEG library was available and its implementations examples were also available.

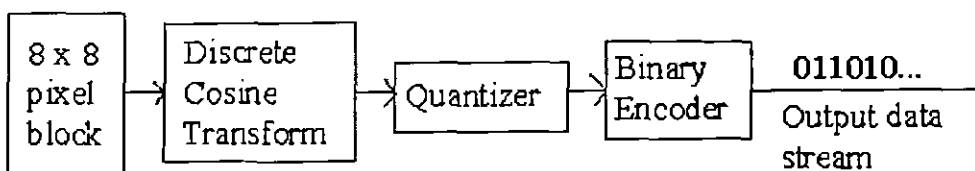
Therefore, M-JPEG video format was used for the implementation of video rather than H.26x formats. M-JPEG stands for Motion JPEG. M-JPEG is a video format that uses ISO JPEG picture compression in each frame of the video. JPEG standard handles compression of continuous tone image for color and grayscale images. In M-JPEG, frames of the video don't interact with each other in any way like they do in other video compression techniques for example MPEG-1, MPEG-2, etc... It makes the video editing easier because each of the frames has all of the information they need stored in them.

#### How JPEG works?

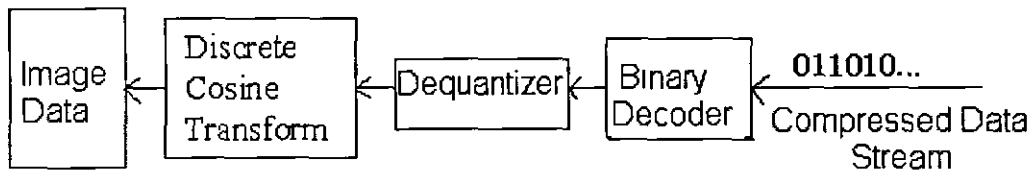
For Compression, JPEG divides the whole image into 8 by 8 pixel blocks, and then calculates the discrete cosine transform (DCT) of each block. A quantizer then calculates the DCT coefficients according to the quantization matrix. This step produces the "lossy" nature of JPEG, but allows for large compression ratios. Then it uses a variable length encoding on these coefficients, and writes the compressed data stream to an output JPEG file. For decompression, repeats the process in reverse order. JPEG recovers the quantized DCT coefficients from the compressed data stream, takes the inverse transforms and displays the image [16].

Following figures describe the JPEG Compression/Decompression process.

### **BLOCK DIAGRAM OF JPEG COMPRESSION**



**BLOCK DIAGARAM OF JPEG DECOMPRESSION**



libjpeg library is used to convert raw frames into jpeg format. On client side, there is no need to store video on disk as it is merely wastage of storage space, so this library is modified in order to achieve this goal.

**5.2.3 Text Input**

Text input is taken from keyboard entered by user. The rate of character entry is usually at a level of a few characters per second or less. Beacause volume of text input is very small so no further ecnoding or copression is applied to textual data. It merely increases the processing burden on server and client.

**5.2.4 Packetization of Text and Video Data**

After text and video input next step was Packetization and transmission of both types of data. The proposed solution devised to send text data in same packet as that of video data. It required changing the current RTP packet format (payload type). Currently RTP packet contains video data only, using it for text also required adding new fields in it. RTP packet format used for the video data only is a given below.

**5.2.4.1 Packet Format for Video Data Only**

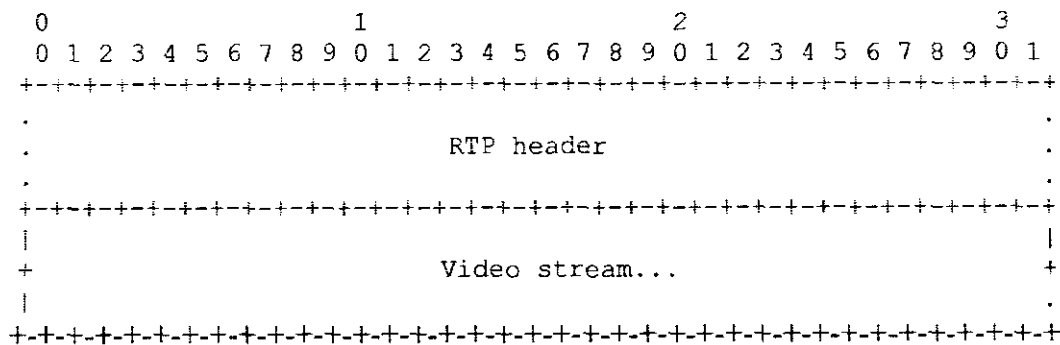


Figure: RTP packet

Where maximum value for PSIZE is 60000 and is defined in file unp.h. It means maximum this size of image only can be transmitted in one packet.

#### 5.2.4.2 Packet Format for Video and Text Data

In order to add text in same RTP packet, two fields are added: one for text data and second for the size of the text data to be transmitted. New packet format (payload type) then became.

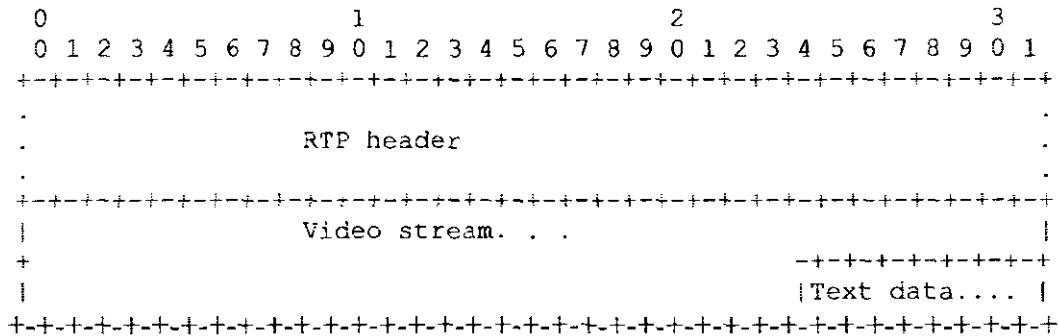


Figure: RTP packet

Where maximum value for TSIZE is 500 and it is also defined in unp.h file. It means maximum 500 bytes can be sent in one RTP packet.

### 5.2.5 Implementation of Buffering

Initially text input was taken for each RTP packet but it was found that even with a maximum typing speed, only a few characters could be typed for every RTP packet and sometimes not even a single character was available. So in order to avoid transmission of a single or only a few characters in each packet, buffering is provided. Small blocks of text data are prepared by the user and transmitted after some delay. It resulted to carry more text data with each packet. Buffering time is selected so that text users on receiving end will perceive a real-time text flow. So text data is sent after every 10th packet of video data. On receiving end, client expects and checks for text data after receiving ten packets of video data.

Even with this arrangement, only a very small text data could be sent but it reduced the case of sending only a single character or empty text file and also reduced the overhead of client to check every RTP packet for text data.

### 5.2.6 Implementation of Redundant Text Data

Textual data is more sensitive to packet loss as compare to video data. Loss of even a single packet containing text may change the meaning of whole conversation. Therefore a mechanism based on RTP is used. It specifies a possibility to repeat text data for redundancy to lower the risk of loss. Since RTP packet overhead is usually much larger than the text contents, the usage of bandwidth with the transmission of redundant data is minimal.

### 5.2.6.1 Packet Format for Redundant Text Data

In order to cope with the loss of packets containing text data, it was decided to send repeated text data. It required to add one more field in RTP packet: A flag field. So the RTP packet format for redundant data is:

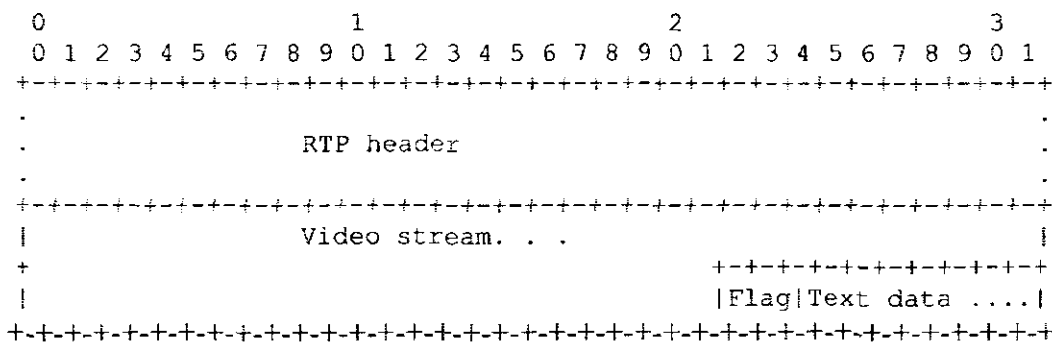


Figure: RTP packet

The mechanism used to implement redundancy idea is as follow: Initially this flag is set to 0 when original text is sent. The same text is buffered on server side. In exactly next RTP packet this same text is sent again but flag is set to 1 this time. On receiving end, client checks for text after receiving ten packets of video data. Then it checks for value of flag. If it is set to 0, which means original text data is received, it uses sequence number and timestamp to display this text data on screen in order along with the video data.

But if the value of flag is set to 1, which means repeated text data is received in this packet, client first compares the sequence number of this newly received packet with the sequence number of last received packet containing text. If it is exactly the one plus the last sequence number received then it means that this text has already been received. So client will not read this text data again and will discard this repeated text data.

If the difference between the sequence number of this newly received packet and the last received sequence number is larger than one then it indicates that newly received packet

contains text data which has already not been received. So client will read this text data and will display it on screen.

More than one levels of redundancy can be implemented. It means redundant data of many previously transmitted packets can be sent in each packet. But only one level of redundancy have been provided in this application.

### 5.2.7 Duplication of Received Packets

In order to avoid duplication of received text data, sequence number of each newly received RTP packet is compared with the last received sequence number to check whether this packet data has already been received or not. If it has already been received, then discard this packet otherwise read data from this packet.

### 5.2.8 Port Numbers for RTP & RTCP

Ports for RTP and RTCP are not fixed and can be allocated in application. For allocation of ports to RTP and RTCP, there are two requirements:

- 1) They should be numerically greater than 1024 and
- 2) They should be consecutive in sequence.

A constant is defined for RTP Port at server in my application, as

```
#define SERV_PORT 9877
```

Same constant is used at the client end. The RTCP port has been defined as SERV\_PORT +1, both at client and server. So port number for RTP communication is 9877 and for RTCP it is 9878. Both of these are defined in unp.h header file.

### 5.2.9 Modification of libjpeg

On client side storing video data on disk and then reading from disk to decompress that video before display was merely wastage of the disk space. Secondly it causes delay for real-time application. It involves two disk accesses. So it was decided to read video data (image) from RTP packet into memory buffers rather than storing it on the disk and then directly display this data from memory buffer using through display application.

As it is mentioned in the description of libjpeg that its source manager reads data to decompress from file rather than from memory and that memory read can be implemented through application.

See Appendix C for JPEG Library details.

One of the standard function calls of Jpeg Library is:

```
jpeg_stdio_dest (j_compress_ptr cinfo, FILE * outfile)
```

In this call, cinfo is a Jpeg object. The outfile is a standard pointer of the type FILE \* (for disk files as used in C language). Similarly, when compressing the file, we have a similar call, as

```
jpeg_stdio_src (j_decompress_ptr cinfo, FILE * infile)
```

It means that whenever the image is compressed, it will have to write on the disk. Similarly, whenever, de-compressing the image, it will have to read from the disk. This was not a very satisfactory arrangement. So there was a need to find a way to read data from memory. After some search, I got a solution that by using the files Jmemdst.c and Jmemsrc.c in place of Jdatadest.c and Jdatasrc.c this will be possible for the source manager and destination manager to read from and write into memory respectively.

These files changed the above functions to:

```
jpeg_memory_src (j_decompress_ptr cinfo, const JOCTET * buffer, size_t bufsize)
```

```
jpeg_memory_dest (j_decompress_ptr cinfo, const JOCTET * buffer, size_t bufsize)
```

A buffer was defined of the desired size and a variable indicating the buffer size of type size\_t. These arguments were passed to these functions. So, that when compression was carried out, the compressed data was written to the buffer defined by the application. Similarly, when the decompression was taking place, the data was being read by from application's buffer. This helped me to reduce the time required to first store data on disk and then read from disk to display on client side.

### 5.2.10 Synchronized Display

For video display on Linux platform two alternatives at hand are: Either to use SVGA library and utilize the VGA functions provided by this library or use X Window System and write any program using X Windows to display both types of data. The image data received on client side was Jpeg compressed and therefore, it needed to be decompressed before it could be displayed on monitor screen. First it was decided to use SVGA library but due to some difficulties with this library discarded this option of SVGAlib and went for X Windows.

There are different reasons which compelled to use X Window system for display rather than SVGAlib.

- Using SVGAlib to display video takes control of whole monitor screen. As received text data was also to be displayed along with the received video data on client side, so giving whole control of screen to video does not solve the problem. Therefore, it was

preferred to use X Windows and created a program, which displayed video data as well as text data in the same windows. GTK+ toolkit is used to create this interface.

- Latest versions of Linux distributions do not contain SVGAlib, or it is not installed in proper directories [17]. Fedora core 5 is used to implement this thesis. It does not have SVGAlib.
- It is not possible to switch back and forth quickly between two consoles using SVGAlib graphics, it may result in screen corruption, forcing reboot.
- X Windows system is cross-platform which runs on variety of UNIXes while only Linux uses SVGAlib.

#### 5.2.10.1 Display Program

GTK+ toolkit is used to create widgets. GTK+: GTK+ is the GIMP toolkit that form basis of GNOME Desktop environments. It is free software [20]. My display program contains one main window. Text and video data is displayed in the same window. Same X window application is used on both client and server. Upper portion of the window displays video data while the lower portion contains an entry widget which displays text data. On client side, each RTP packet received from server contains video data as well as the text data. Received video data is Jpeg compressed. This data is directly decompressed into a buffer (memory) rather than storing on disk. Then this decompressed video data is displayed on screen on the upper portion of the window. While the associated text data is displayed in the entry widget in the lower portion of the same window.

On client side `settext()` function is used to set received text in entry widget.

Same program is used on server side to display a frame that is to be transmitted and to take text input from user. Same interface is used for this program. The only difference is that the entry widget, used to display text on client side, is used to take text input from user on server side. On server side the display program is taking video input from stored video file on disk rather than from memory as it is doing on client side. Display program on server side reads a frame stored on disk, decompresses it and displays it in upper portion of the window before transmission of that frame.

On server side `gettext()` function is used to get text from entry widget entered by user.

On client side, to give a look of flowing text on display, the newly received text is concatenated with the already received text each time and then it is displayed. It exactly gives the impression of flowing text as is displayed on TV channels on the lower portion of any video clip display. Text look like moving from right to left, entering on right side and exiting on left side.

See Appendix B for interfaces used to take input from user on server side and display it on client side.



### 5.2.11 Graphical User Interfaces

In implementation of the solution graphical user interface is only provided on server side to display server status, client details, sender reports data and receiver report data. To create this interface, curses library is used. Curses is a terminal control library for Unix-like systems. It enables the construction of text user interface (TUI) applications. It is a library of functions that manage an application's display on character-cell terminals (e.g., VT100). Curses library is a part of most Linux distributions.

When you write a program that uses this library, you have to include the directive

```
#include <ncurses.h>
```

When compile the program with g++, add -lncurses to the command. See Appendix B for Interfaces developed using ncurses library functions.

## 5.3 Implementation Tools

### Hardware

- Firewire card
- i.Link cable to connect camcorder with firewire card
- Sony HCR-DC32E Handy camcorder

### Softwares

#### Drivers

Drivers for firewire card

- raw1394
- ieee1394
- ohci1394

#### Libraries

- libraw1394
- libavc1394
- libiec61883
- libdv
- libjpeg
- libcurses
- libX11

# **CHAPTER 6**

## **TESTING AND PERFORMANCE IMPROVEMENT**

## 6. Testing and Performance Improvement

### 6.1 Overview

This research is an attempt to send textual data (non-continuous data) in real-time using RTP which is primarily designed for the transmission of real-time multimedia contents (continuous media types). RTP provides the necessary measures for the handling of real-time contents. Its time stamping and sequence number options make it easy to re-arrange real-time data on receiving side in its original order and process it in real-time. Using RTP for the transmission of real-time text together with other media types provides a way to achieve uniform handling of text and other media types. This fact can be utilized in applications like conferencing systems, firewalls, and network translation devices etc.

Other than conferencing systems, RTP can also be used for the transmission of real-time contents for other applications, for example, to transmit simulation data. Some applications may also require sending textual data along with the other multimedia types. For example, data from sensors, pointer coordinates etc., can be transmitted along with other multimedia types and need to be synchronized with other media types at receiver. TV and News agencies are other users with the need of textual data transmission along with the other media types like video in real-time.

Keeping in view the requirements of these applications this research is an attempt to see the possibility of sending textual data along with the video data using RTP and get synchronization of both types of data on receiving end. Video data is captured using camcorder, while the text input is taken from keyboard entered by user.

### 6.2 Testing

Client server architecture has been implemented and run on LAN connected by two computers. 1000 packets are transmitted and jitter is calculated using the jitter calculation formula given in [2].

Average inter arrival jitter for video only packets was 20149 microseconds measured in timestamp unit.

Average inter arrival jitter for video plus text packets was 30871 microseconds measured in timestamp unit.

Achieved synchronized display of text and video data on receiving end which is verified by displaying the same data before transmission on server as well.

## 6.3 Performance Improvement

### 6.3.1 Single Packet to Reduce Bandwidth Wastage

Text input was the characters entered by the user through keyboard. Even with a maximum typing speed, rate of character entered was a few characters per second or less than it. In worst it may be a single character or not even a single character. Which means text data available for transmission is only a few bytes or may be a single byte. On the other hand, minimum size of RTP header is 12 bytes, while that of UDP is 8 bytes. So sending separate packets for few bytes of text data requires sending minimum 20 bytes of header data with each packet. This was merely wastage of bandwidth. Also extra processing was required in this case for synchronizing both types of data received through two different packets on receiving end.

So this approach to send text data in the same RTP packet used for video data reduced the bandwidth wastage and synchronization overhead on receiving end.

### 6.3.2 Buffering of Text Data

Initially text input was taken for each RTP packets to transfer text along with the video data in same packet. But it was found that even with a maximum typing speed, only a few characters could be typed for every RTP packet and sometimes not even a single character was available. It means video frames were being transmitted in every RTP packet but no text data was available for every packet or very few characters were available to transmit. But on receiving end, client was checking every packet for the text data. So there was a need to provide some means of buffering in order to increase text volume and circumvent client from checking each packet for text data.

Therefore in order to avoid transmission of a single or only a few characters in each packet, buffering is provided. Small blocks of text data are prepared by the user and transmitted after some delay. The result was to carry more text data with RTP packet. Buffering time is selected so that on receiving end an illusion of a real-time text flow is achieved. This is achieved by sending text data after 10 packets of video data. On receiving end, client expects and checks for text data after receiving ten packets of video data.

This mechanism reduced the chance of sending only a single character or empty text file. This improved the size of transmitted text data and it enabled to send more text data than in case of no buffering. It also reduced the overhead of client to check every RTP packet for text data. Now client checks for text data after processing of ten packets of video data. If the buffering time was further increased, it might reduce the real-time effect of text data on client side. So sending textual data after 10th packet best suited the situation.

### 6.3.3 Redundant Text Data

Text data is also more sensitive for packet loss as compare to video data. Loss of a single or even a few video packets may not affect the perceived quality of video data as much. But loss of even a single packet of text data may change the meaning of whole text conversation. Reception of duplicated text data also causes the same effects. In order to overcome this problem, a mechanism is provided to avoid the effects of loss of text data and reception of repeated text data.

Repeated data is transmitted in consecutive RTP packets to lower the risk of loss. Since RTP packet overhead is usually much larger than the text contents, the usage of bandwidth with the transmission of redundant data is minimal. If a packet containing original text is lost, then text can be read from the next coming RTP packet.

Although more than one level of redundancy can be implemented, but only one level of redundancy is provided here. Those applications for which loss of packet is much more serious issue, they can either send repeated text data in each RTP packet or they can send text data in separate RTP packets containing original text along with the repeated text of many previously transmitted packets.

### 6.3.4 No Duplication

In order to avoid duplication of received text data, a mechanism based on sequence number is provided. In this way duplicate reception of same packet is avoided.

### 6.3.5 Read Image Data into Memory for Decompression

On client side there is no need to store the video data. Display of the synchronized video and text data is the primary focus of this study. Storing video data on client side is merely wastage of the disk space. Secondly it is a real time application, so first storing data on disk and then reading it from disk causes delay. It involves two disk accesses. So jpeg library is modified to read video data from memory for decompression before display. It decreased number of disk access which is ideal for real-time applications.

### 6.3.6 Text Compression

Text data is transmitted without applying compression. As text data comprises only few bytes as compare to video data so there is no need to apply compression on it. Compression is applied on video data which has size in megabytes. Applying compression on text also, merely increases the processing burden of client and server. Similarly no encoding is applied to text data. Text input is simply taken from keyboard and is transmitted through network.

## 6.4 How to use this software over Internet

This application can also run over Internet. For this purpose keep image size smaller, 176x144 otherwise the packet size becomes larger to easily travel over the Internet. To run over the Internet, make sure that a serial connection and a modem are installed on both client and server computers and that ppp is also installed. If you have fixed IP address for the server then there is no issue. Use it to connect the client with the server. But mostly IP addresses are dynamic. In that case use the following steps.

### On Server Computer:

a). Dial your ISP through your PPP connection. Use wvdial, if you are using fedora or RedHat for dialing.

b). When you have connected with ISP, assuming you had already included your username and password in the wvdial configuration file. Now you issue the same command:

```
ifconfig
```

You will see an output for each of your interface, which would most likely be eth0, lo and ppp0. We are interested in ppp0. In this interface, you will see an IP address in the form x1.x2.x3.x4. Note it down on piece of a paper. If you do not find ppp0 listed, then you are not connected with Internet and you will have to try it again.

c). Create a file say addrfile, which contains this address. Send this file to client computer via normal email. Now the client knows the IP address of the server. Run your software for capturing the images and start sending them on the network.

### On Client Side:

a). The client also uses Linux and connects to its ISP. It should also use the "ifconfig" command to check that a connection to Internet is established or not. When gets the IP address from the server side, through normal email, client should ping the server computer as follows:

```
ping x1.x2.x3.x4.
```

If the response is 64 bytes being received, then both are on Internet. The client can also communicate its IP address to the other party. Now you can also use FTP and telnet etc on these computers, since you now know the IP addresses of each other.

b). Next run your client software on the client computer and you should start seeing the video.

That's all there is to it.

## 6.5 Summary

In this way attempt has been made to give a mechanism to meet the needs of real-time applications. Sending text and video data in a single packet reduces overhead of synchronization for display of both types of data on receiving end. It has been tried to resolve both problems of slow typing speed, by introducing buffering and packet loss, by transmitting redundant data. This mechanism of transmission gives text arrival in correct order, without duplication, and with detection and indication of loss. Real-time display on client side is improved by reading video data from memory for decompressing image rather than from disk.

# **CHAPTER 7**

## **CONCLUSIONS AND FUTURE ENHANCEMENTS**



## 14. Conclusions and Future Enhancements

### 14.1 General Discussions

This thesis is an extension of the work already done by two students of the same university. Their work deals only with the video capture on server, packetization of video data only, transmission and reception and rendering on client side over Linux platform. This work includes video capture along with the textual data on server, packetization of both types of data, transmission, reception on client side, synchronized display of both types of data on client side. Due to the requirements of sending both types of data, the implementation details of video also differ from existing work as follows:

### 14.2 Conclusions

- In previous work, video capture is achieved with analog video camera and then analog to digital conversion is used to convert data in digital form. So capture card and its driver used are also different, one suitable to work with analog camera. In this case, digital camcorder is used to grab video data, which works using FireWire card, and uses Linux IEEE1394 subsystem. Its drivers are different than an analog video camera.
- In previous work, application used to grab frames is vgrab which uses Linux API, video for Linux and works for analog camera. Because capturing images through analog camera requires different settings for image capture like selection of color palette etc as compare to those required while using digital camcorder. In this work dvgrab is used which accesses digital camera through Linux IEEE1394 module. It requires a number of different libraries to capture video from camera. Accessing digital camera in Linux is comparatively a tough job because it cannot be simply opened on /dev/ like other devices. It requires accessing handle of ieee1394 bus. Details of Linux IEEE1394 subsystem are given in chapter Implementation Details.
- For display of video SVGA library is used in previous work which displays video over the whole monitor. In this work, it is required to display text as well along with the video data; secondly SVGA is no more part of each latest version of Linux distributions, so X Window System is used to display text data along with the video data.
- Basic working of client server is the same like Sockets have been used to establish a connection between the client and server. In previous work threads have been used to separately manage video capture. In this work threads have not been used for video capture rather image grabbing is managed separate in order to cope with the difference of processing speed of computer and the digital camcorder.

- For text, input is taken from keyboard and is packetized in the same packet as video. In order to facilitate the slow typing speed as compare to the fast video frames transfer speed, buffering is implemented, which is specified in RFC 4103 "RTP Payload for Text Conversation". Video data is sent in each RTP packet while the text input is taken and sent after sending ten packets of video.
- As textual data is more sensitive to packet loss as compare to video data. Losing one or few video frames do not affect the result so much but losing even a single text packet can change the meaning of whole text. So in order to make textual data more reliable, the concept of redundancy of textual data is applied here which is also specified in RFC 4103 "RTP Payload for Text Conversation". Only one level of redundancy is implemented. That is redundant data is sent only for one packet which is the exactly previous packet carrying text data. Details of these implementations are given in chapter Text.

### 14.3 Future Enhancements

This research is an attempt to send textual data using RTP which is basically developed and used for multimedia real-time applications. Now a day attempts have also been made to use RTP for the transmission of non-continuous real-time media type like text. Some of details are given in Chapter Literature Survey. Initially it have been tried to keep the things as simple as possible. In future this research can be enhanced to get further results. Some possible enhancements could be:

- Currently text input has been taken through key board. The rate of character entry is usually at a level of a few characters per second or less. Only a few new characters are expected to be transmitted with each new packet. Stored text file, handwriting recognition, voice recognition or any other input method can be used to increase the volume of textual data.
- Currently it has been selected to send data in the same packet as used for video because keyboard input results in availability of only few characters to send. So sending it in separate packet causes bandwidth wastage. In future, text can be taken from any other source (capable of providing large text data input) and can be transmitted in separate RTP packets.
- Currently image size is kept smaller to meet the maximum packet size restriction to travel through the network. Complete frame is sent in a single packet. To send high resolution video, in future image size can be made larger and a single image can be transmitted in more than one packet for which details are given in RTP specifications.
- Same work can be implemented for SRTP, which is a secure version of RTP. It will require implementing encryption/decryption of transmitted data. It will also require a mechanism to generate and distribute encryption keys.

## **APPENDIX A**

### **HEADER FILE (RTP.H)**

## Appendix A – Header File (rtp.h)

This is a header file for rtp and needs to be used with server as well as client. This is given in RFC 3550 and provides examples of C code for aspects of RTP sender and receiver algorithms.

```

/*
 * rtp.h -- RTP header file (RFC 3550)
 */
#include <sys/types.h>

/*
 * The type definitions below are valid for 32-bit
architectures and
 * may have to be adjusted for 16- or 64-bit architectures.
 */
typedef unsigned char u_int8;
typedef unsigned short u_int16;
typedef unsigned int u_int32;
typedef short int16;

/*
 * Current protocol version.
 */
#define RTP_VERSION 2

#define RTP_SEQ_MOD (1<<16)
#define RTP_MAX_SDES 255 /* maximum text length for SDES
*/

typedef enum {
 RTCP_SR = 200,
 RTCP_RR = 201,
 RTCP_SDES = 202,
 RTCP_BYE = 203,
 RTCP_APP = 204
} rtcp_type_t;

typedef enum {
 RTCP_SDES_END = 0,
 RTCP_SDES_CNAME = 1,
 RTCP_SDES_NAME = 2,

```

---

```

 RTCP_SDES_EMAIL = 3,
 RTCP_SDES_PHONE = 4,
 RTCP_SDES_LOC = 5,
 RTCP_SDES_TOOL = 6,
 RTCP_SDES_NOTE = 7,
 RTCP_SDES_PRIV = 8
} rtcp_sdes_type_t;

/*
 * RTP data header
 */
typedef struct {
 unsigned int version:2; /* protocol version */
 unsigned int p:1; /* padding flag */
 unsigned int x:1; /* header extension flag */
 unsigned int cc:4; /* CSRC count */
 unsigned int m:1; /* marker bit */
 unsigned int pt:7; /* payload type */
 unsigned int seq:16; /* sequence number */
 u_int32 ts; /* timestamp */
 u_int32 ssrc; /* synchronization source */
 u_int32 csrc[1]; /* optional CSRC list */
} rtp_hdr_t;

/*
 * RTCP common header word
 */
typedef struct {
 unsigned int version:2; /* protocol version */
 unsigned int p:1; /* padding flag */
 unsigned int count:5; /* varies by packet type */
 unsigned int pt:8; /* RTCP packet type */
 u_int16 length; /* pkt len in words, w/o this
word */
} rtcp_common_t;

/*
 * Big-endian mask for version, padding bit and packet type
pair
 */
#define RTCP_VALID_MASK (0xc000 | 0x2000 | 0xfe)
#define RTCP_VALID_VALUE ((RTP_VERSION << 14) | RTCP_SR)

/*
 * Reception report block
 */

```

---

```

typedef struct {
 u_int32 ssrc; /* data source being reported
*/
 unsigned int fraction:8; /* fraction lost since last
SR/RR */
 int lost:24; /* cumul. no. pkts lost
(signed!) */
 u_int32 last_seq; /* extended last seq. no.
received */
 u_int32 jitter; /* interarrival jitter */
 u_int32 lsr; /* last SR packet from this
source */
 u_int32 dlsr; /* delay since last SR packet
*/
} rtcp_rr_t;

/*
 * SDES item
 */
typedef struct {
 u_int8 type; /* type of item
(rtcp_sdes_type_t) */
 u_int8 length; /* length of item (in octets)
*/
 char data[1]; /* text, not null-terminated */
} rtcp_sdes_item_t;

/*
 * One RTCP packet
 */
typedef struct {
 rtcp_common_t common; /* common header */
 union {
 /* sender report (SR) */
 struct {
 u_int32 ssrc; /* sender generating this
report */
 u_int32 ntp_sec; /* NTP timestamp */
 u_int32 ntp_frac;
 u_int32 rtp_ts; /* RTP timestamp */
 u_int32 psent; /* packets sent */
 u_int32 osent; /* octets sent */
 rtcp_rr_t rr[1]; /* variable-length list */
 } sr;

 /* reception report (RR) */

```

---

```

 struct {
 u_int32 ssrc; /* receiver generating this
report */
 rtcp_rr_t rr[1]; /* variable-length list */
 } rr;

 /* source description (SDES) */
 struct rtcp_sdes {
 u_int32 src; /* first SSRC/CSRC */
 rtcp_sdes_item_t item[1]; /* list of SDES items*/
 } sdes;

 /* BYE */
 struct {
 u_int32 src[1]; /* list of sources */
 /* can't express trailing text for reason */
 } bye;
 } r;
} rtcp_t;

typedef struct rtcp_sdes rtcp_sdes_t;

/*
 * Per-source state information
 */
typedef struct {
 u_int16 max_seq; /* highest seq. number seen */
 u_int32 cycles; /* shifted count of seq. number
cycles */
 u_int32 base_seq; /* base seq number */
 u_int32 bad_seq; /* last 'bad' seq number + 1 */
 u_int32 probation; /* sequ. packets till source is
valid */
 u_int32 received; /* packets received */
 u_int32 expected_prior; /* packet expected at last
interval */
 u_int32 received_prior; /* packet received at last
interval */
 u_int32 transit; /* relative trans time for prev
pkt */
 u_int32 jitter; /* estimated jitter */
} source;

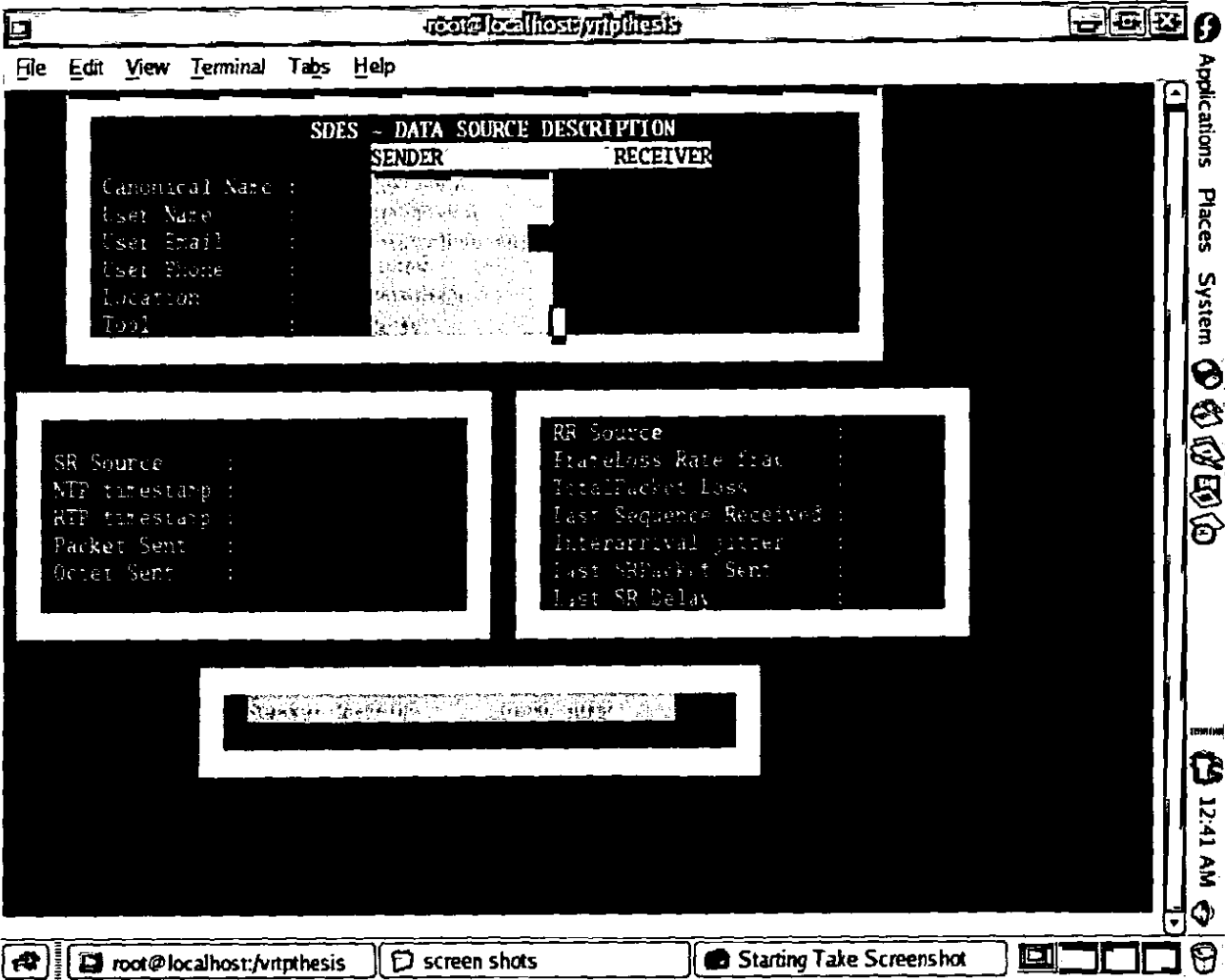
```

# **APPENDIX B**

# **INTERFACES**

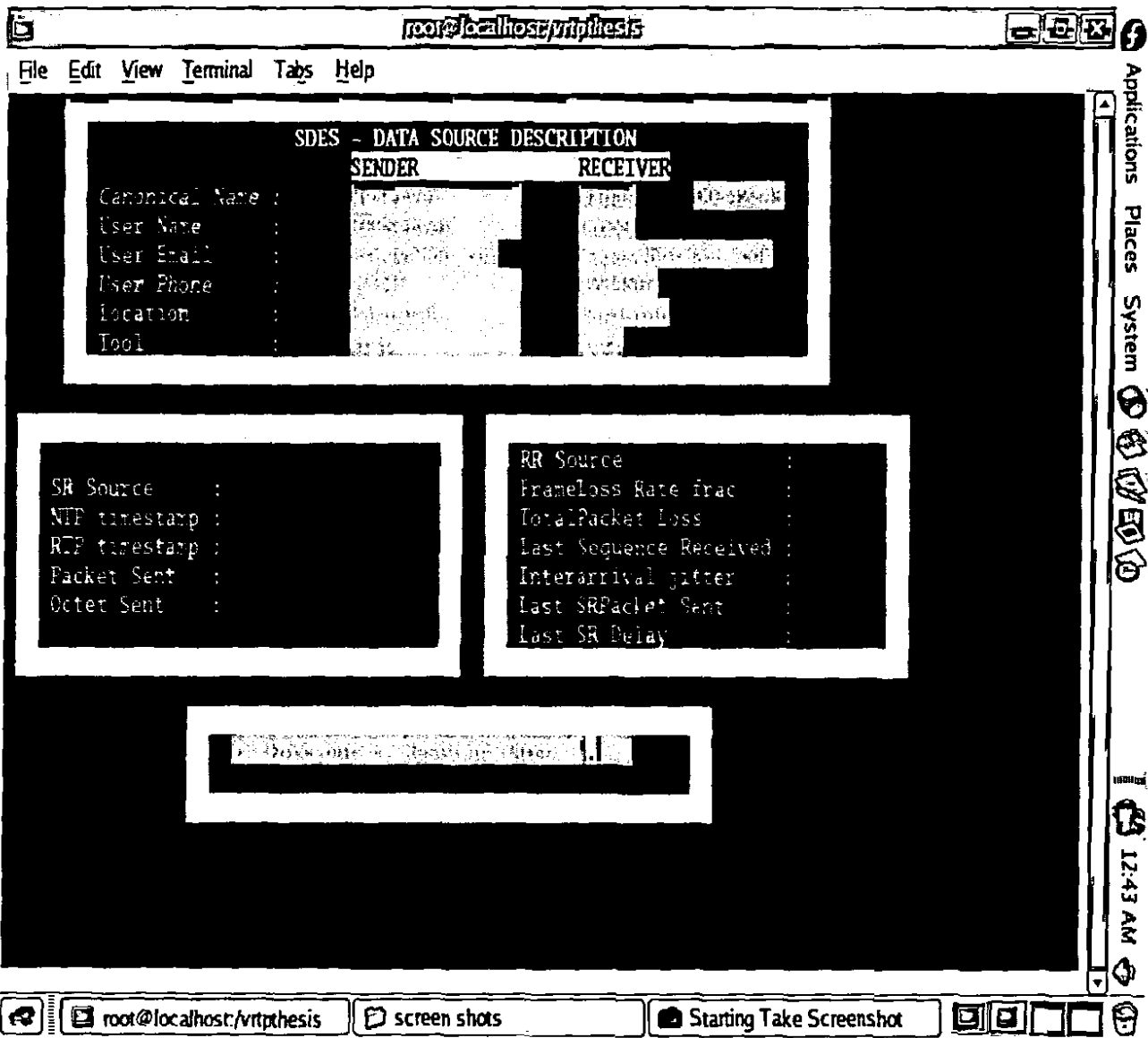


Appendix B – Interfaces



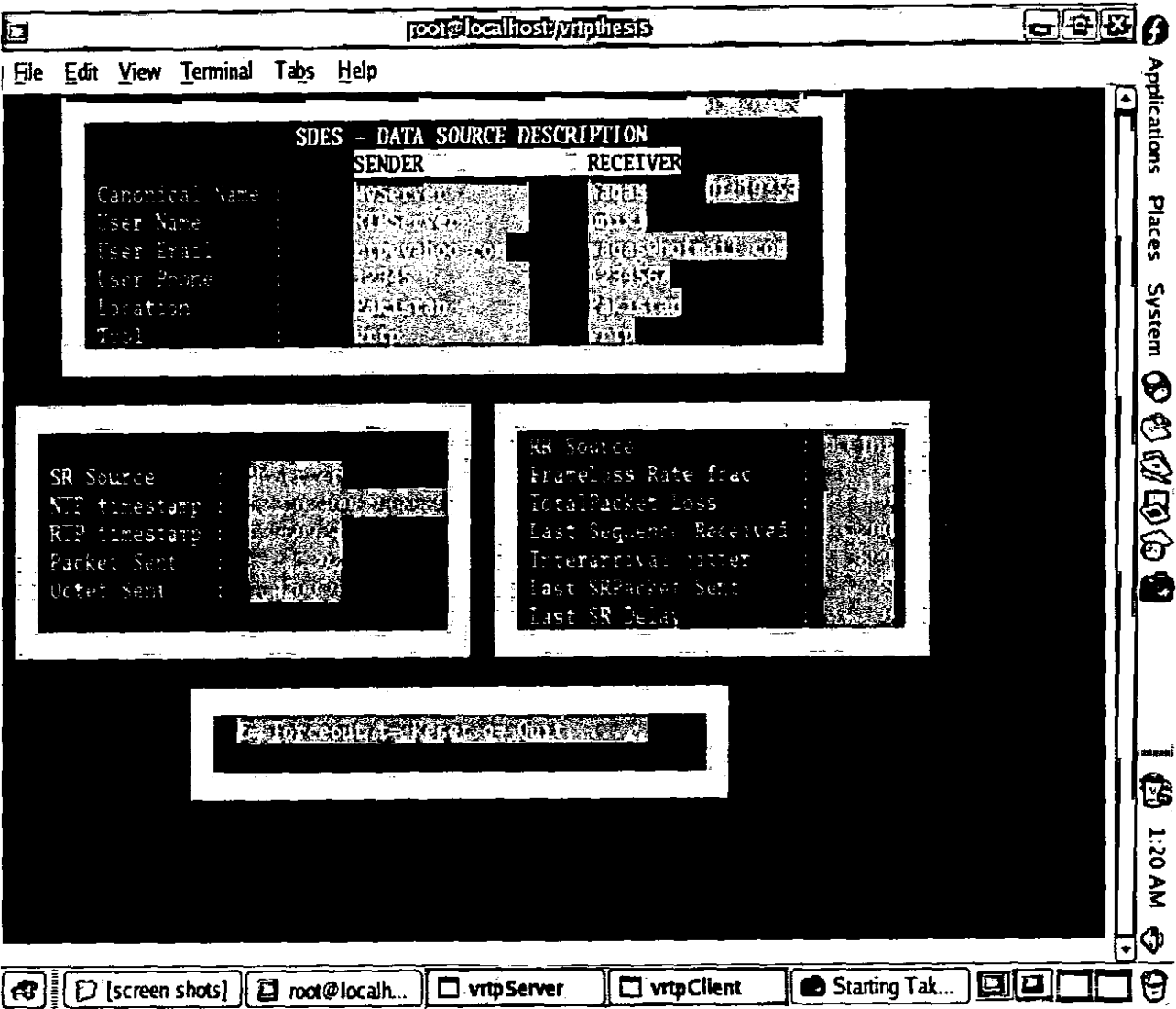
Server waiting for the connection request from client.

Data Source Description currently showing the details of server only because no client is connected.



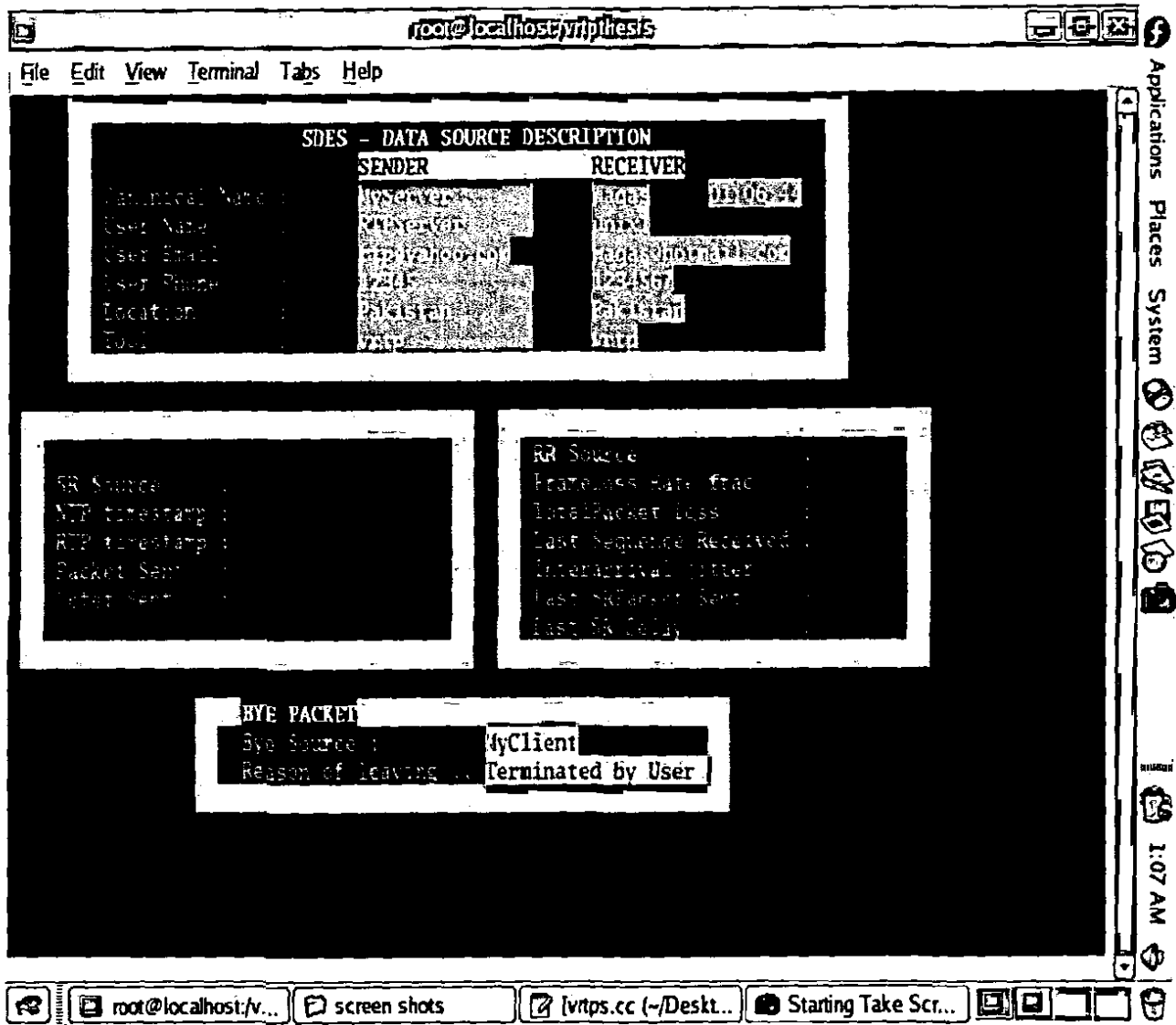
Connection established

Data Source Description showing the details of both the server and the client.



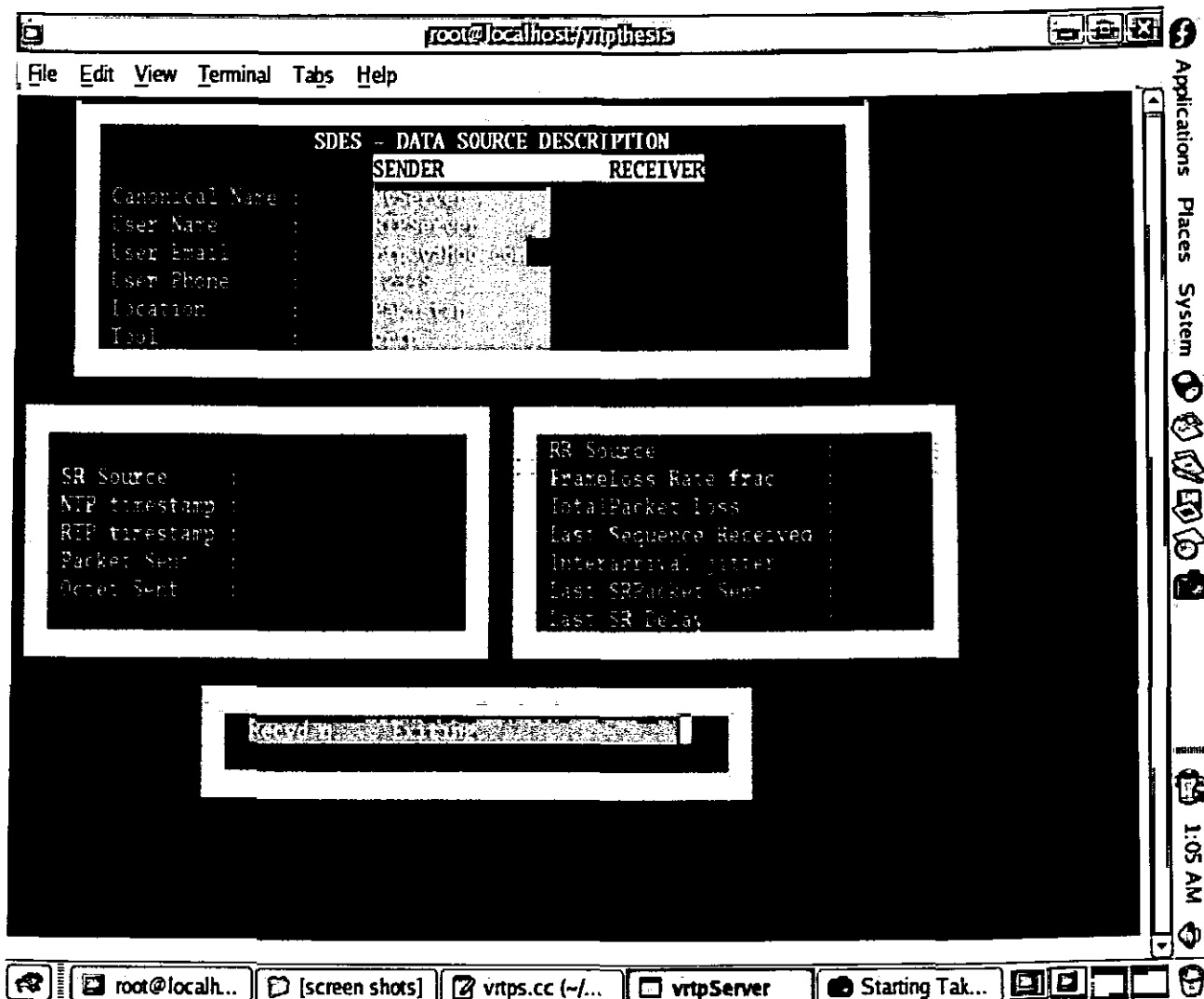
Transmission started

Sender and Receiver Reports both are displayed.

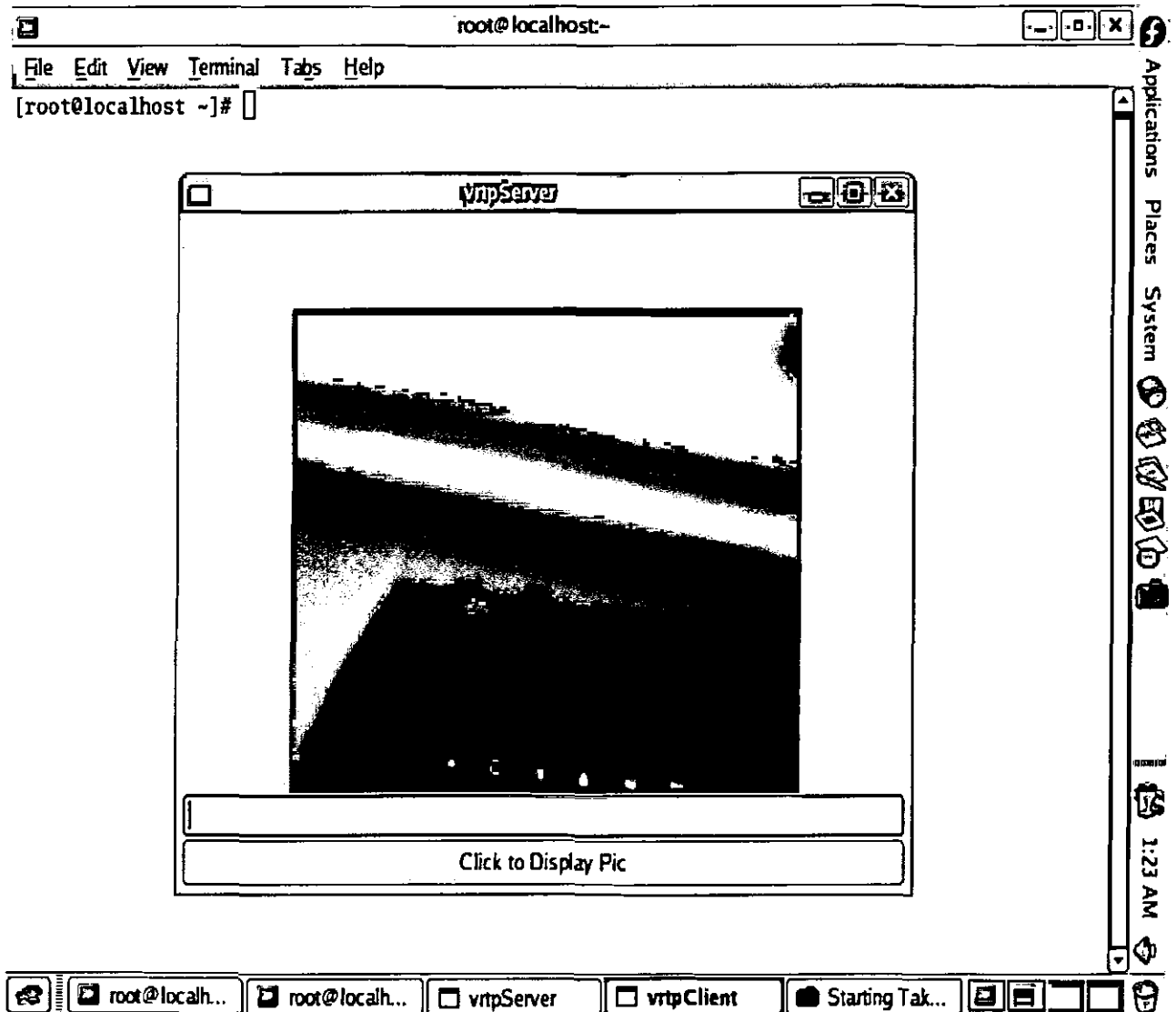


Received Bye packet from client.

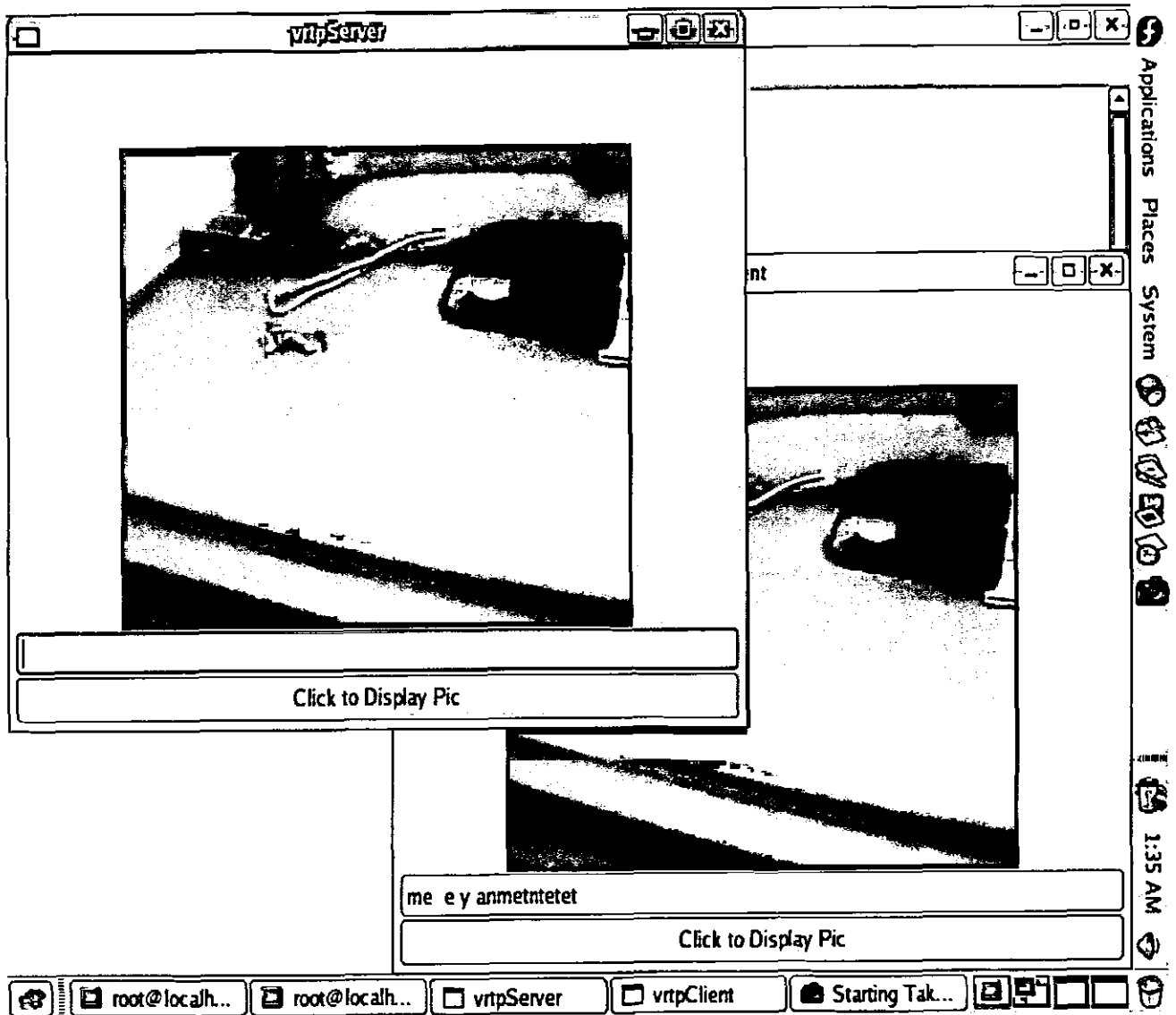
Connection between client and server is disconnected and server again goes to waiting state.



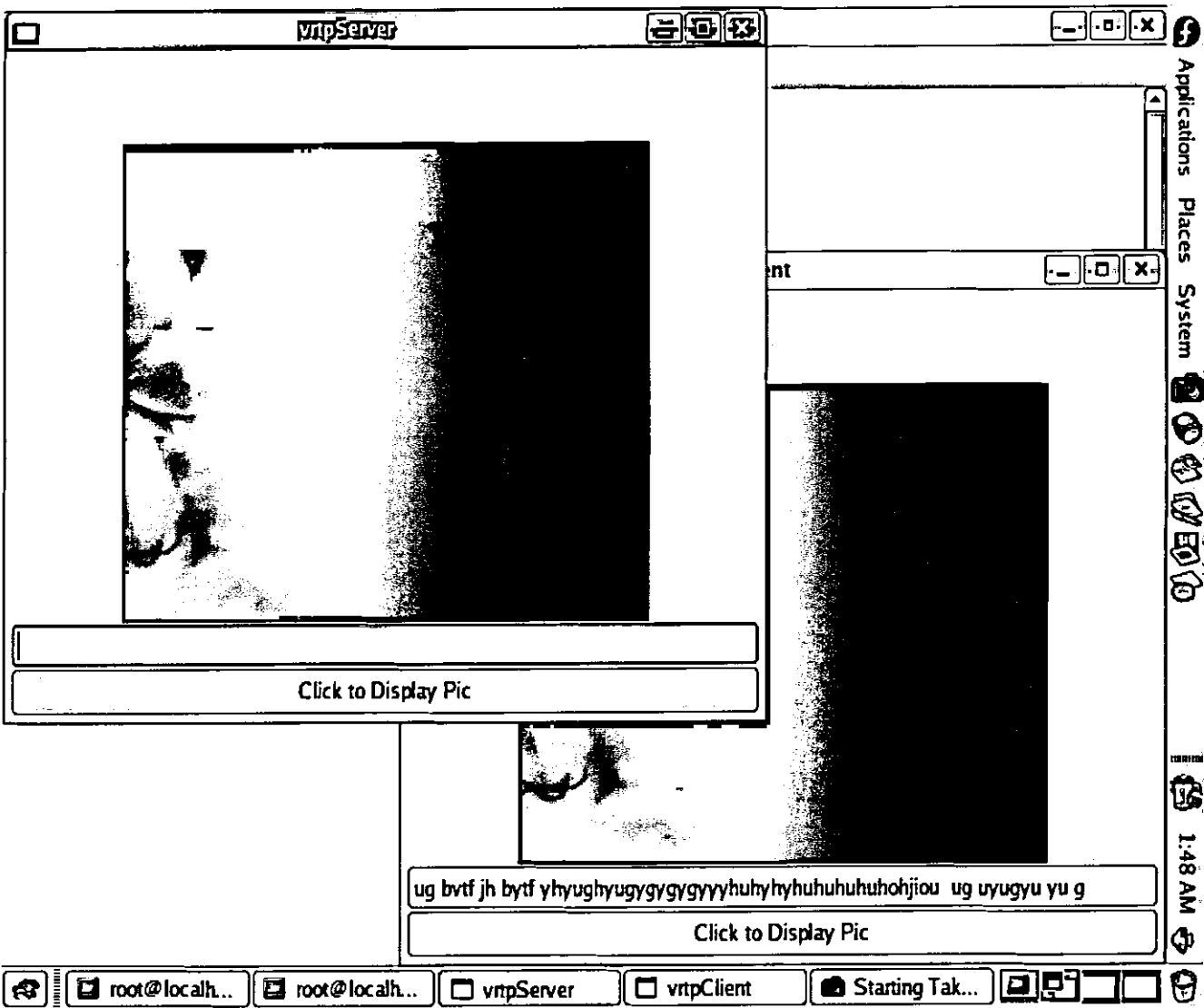
Server exiting after receiving q from keyboard.



Server window to display video plus text data.



Text entered in server window displayed in client window.  
Video transmitted by server is also displayed by in client window



Next received text is concatenated with the previously received text on client side.



# **APPENDIX C**

## **JPEG LIBRARY**

## JPEG Library (libjpeg)

### Library Overview

#### Typical compression/decompression steps

The rough outline of a JPEG compression operation is:

- Allocate and initialize a JPEG compression object

```
struct jpeg_compress_struct cinfo;
struct jpeg_error_mgr jerr;
...
cinfo.err = jpeg_std_error(&jerr);
jpeg_create_compress(&cinfo);
```

A JPEG compression object is a "struct jpeg\_compress\_struct". A structure representing a JPEG error handler is a "struct jpeg\_error\_mgr". `jpeg_create_compress` allocates a small amount of memory.

- Specify the destination for the compressed data (eg, a file)

Typical code for this step looks like:

```
FILE * outfile;
...
if ((outfile = fopen(filename, "wb")) == NULL) {
 fprintf(stderr, "can't open %s\n", filename);
 exit(1);
}
jpeg_stdio_dest(&cinfo, outfile);
```

Where the last line invokes the standard destination module. You may not change the destination between calling `jpeg_start_compress()` and `jpeg_finish_compress()`.

- Set parameters for compression, including image size & color space etc

```
image_width Width of image, in pixels
image_height Height of image, in pixels
input_components Number of color channels(samples per
pixel)
in_color_space Color space of source image
```

Typical code for a 24-bit RGB source image is

---

```

 cinfo.image_width = Width; /* image width and
height, in pixels */
 cinfo.image_height = Height;
 cinfo.input_components = 3; /* # of color components
per pixel */
 cinfo.in_color_space = JCS_RGB; /* colorspace of input
image */

 jpeg_set_defaults(&cinfo);
 /* Make optional parameter settings here */

```

- `jpeg_start_compress(...);`

Typical code:

```
jpeg_start_compress(&cinfo, TRUE);
```

Once called `jpeg_start_compress()`, cannot alter any JPEG parameters or other fields of the JPEG object until you have completed the compression cycle.

- while (scan lines remain to be written)

```
jpeg_write_scanlines(...);
```

write all the required image data by calling `jpeg_write_scanlines()` one or more times which returns the number of scanlines actually written. Image data should be written in top-to-bottom scanline order.

- `jpeg_finish_compress(...);`

After all the image data has been written, call `jpeg_finish_compress()` to complete the compression cycle. This step is *ESSENTIAL* to ensure that the last bufferload of data is written to the data destination. `jpeg_finish_compress()` also releases working memory associated with the JPEG object.

Typical code:

```
jpeg_finish_compress(&cinfo);
```

- Release the JPEG compression object

*When done with a JPEG compression object, destroy it by calling `jpeg_destroy_compress()`. This will free all memory. Typical code:*

```
jpeg_destroy_compress(&cinfo);
```

A JPEG compression object holds parameters. Creation and destruction of this object is separate from the start and finishing of the compression of an image, so same object can be used to compress/decompress a series of images. So the same parameters setting can be applied to a sequence of images. This fact is utilized for M-JPEG compression and decompression.

The image data to be compressed is supplied to the function `jpeg_write_scanlines()` from in-memory buffers. If file-to-file compression is required, then reading image data from the source file is the application's responsibility. The compressed data is written by the library by calling a "data destination manager", which typically writes the data into a file; but the application can provide its own destination manager to do something else.

Similarly, the rough outline of a JPEG decompression operation is:

- Allocate and initialize a JPEG decompression object

*This is just like initialization for compression, as discussed above, except that the object is a "struct `jpeg_decompress_struct`" and you call `jpeg_create_decompress()`. Error handling is exactly the same.*

*Typical code:*

```
struct jpeg_decompress_struct cinfo;
struct jpeg_error_mgr jerr;
...
cinfo.err = jpeg_std_error(&jerr);
jpeg_create_decompress(&cinfo);
```

- Specify the source of the compressed data (eg, a file)

*As previously mentioned, the JPEG library reads compressed data from a "data source" module. The library includes one data source module which knows how to read from a stdio stream. You can use your own source module if you want to do something else, as discussed later.*

*If you use the standard source module, you must open the source stdio stream beforehand.*

*Typical code for this step looks like:*

```
FILE * infile;
...
if ((infile = fopen(filename, "rb")) == NULL) {
 fprintf(stderr, "can't open %s\n", filename);
 exit(1);
}
jpeg_stdio_src(&cinfo, infile);
```

*where the last line invokes the standard source module.*

- Call `jpeg_read_header()` to obtain image info

*Typical code for this step is just*

```
jpeg_read_header(&cinfo, TRUE);
```

*This will read the source datastream header markers, up to the beginning of the compressed data proper. On return, the image dimensions and other info have been stored in the JPEG object.*

- Set parameters for decompression

*Default values are set by each call to `jpeg_read_header()`. However, you may alter these defaults before beginning the decompression in this step.*

- `jpeg_start_decompress(...);`

*This will initialize internal state, allocate working memory, and prepare for returning data.*

*Typical code is just*

```
jpeg_start_decompress(&cinfo);
```

- while (scan lines remain to be read)

```
jpeg_read_scanlines(...);
```

*Read the decompressed image data by calling `jpeg_read_scanlines()` one or more times. At each call, you pass in the maximum number of scanlines to be read (ie, the*

---

*height of your working buffer). Image data is returned in top-to-bottom scanline order.*

- `jpeg_finish_decompress(...);`

*After all the image data has been read, call `jpeg_finish_decompress()` to complete the decompression cycle. This causes working memory associated with the JPEG object to be released.*

*Typical code:*

```
jpeg_finish_decompress(&cinfo);
```

- Release the JPEG decompression object

*When you are done with a JPEG decompression object, destroy it by calling `jpeg_destroy_decompress()` or `jpeg_destroy()`. The previous discussion of destroying compression objects applies here too.*

*Typical code:*

```
jpeg_destroy_decompress(&cinfo);
```

This is similar to the compression outline except reading the data stream header step. This is helpful because information about the image's size, colorspace, etc is available when the application selects decompression parameters. For example, the application can choose an output scaling ratio that will fit the image into the available screen size.

The decompression library calls a data source manager to obtain compressed, which typically reads the data from a file; but other behaviors can be obtained through applications. Decompressed data is delivered into in-memory buffers passed to `jpeg_read_scanlines()`.

To abort an incomplete compression or decompression operation `jpeg_abort()` is called or `jpeg_destroy()` can also be called which also destroys the JPEG object.

JPEG compression and decompression objects are two separate struct types who share some common fields, and certain routines such as `jpeg_destroy()` works for both types of object. The JPEG library has no static variables: all state is in the compression or decompression object. Therefore it is possible to process multiple compression and decompression operations concurrently, using multiple JPEG objects.

**APPENDIX D**

**REFERENCES**

## References

- [1]. ITU-T Recommendation T.140 (1998) – “Text conversation protocol for multimedia application, with amendment 1”, (2000).
- [2]. Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "*RTP: A Transport Protocol for Real-Time Applications*", RFC 3550, July 2003.
- [3]. H. Schulzrinne, A. Rao and R. Lanphier, "*Real Time Streaming Protocol (RTSP)*", RFC2326, April 1998.
- [4]. G. Hellstrom, Omnitor AB, P. Jones, "*RTP Payload for Text Conversation*", RFC 4103, June 2005.
- [5]. G. Hellstrom, Omnitor AB, P. Jones, "*RTP Payload for Text Conversation Interleaved in an Audio Stream*", RFC 4351, January 2006.
- [6]. "*IEEE 1394 for Linux*", [http://www.linux1394.org/start\\_req.php](http://www.linux1394.org/start_req.php)
- [7]. Randa El-Marakby, David Hutchison, "*Evaluation of the Real-time Transport Protocol (RTP) for Continuous Media Communications*", url = [citeseer.ist.psu.edu/88756.html](http://citeseer.ist.psu.edu/88756.html)
- [8]. Tom Sheldon's Encyclopedia of Networking and Telecommunications "*Multimedia*",  
[Available at: <http://www.linktionary.com/m/multimedia.html>]
- [9]. Fernando Boronat Seguí, Juan Carlos Guerri Cebollada "*NON CONTINUOUS MEDIA STREAMS TRANSMISSION USING RTP. A MULTICAST RTP-BASED TOOL*" [Available at: [http://personales.gan.upv.es/~fboronat/RTP\\_Text.html](http://personales.gan.upv.es/~fboronat/RTP_Text.html)]
- [10]. Chunlei Liu, "*Multimedia Over IP: RSVP, RTP, RTCP, RTSP*"
- [11]. Prof. Jean-Yves Le Boudec, Prof. Andrzej Duda, Prof. Patrick Thiran, [www.cs.wustl.edu/~jain/cis788-97/ftp/ip\\_multimedia.pdf](http://www.cs.wustl.edu/~jain/cis788-97/ftp/ip_multimedia.pdf)
- [12]. "*RTP Video ( Real-time Transport Protocol Video )*"  
[http://www.protocolbase.net/protocols/protocol\\_RTP%20Video.php](http://www.protocolbase.net/protocols/protocol_RTP%20Video.php)



- 
- [13]. "Why we like MJPEG compression"  
<http://www.securityinfowatch.com/article/article.jsp?siteSection=430&id=14335>
- [14]. Ismo Anttila, Markku Paakkunainen, Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory, "Transferring real-time video on the Internet", <http://www.tml.tkk.fi/Opinnot/Tik-110.551/1997/iwsem.html>
- [15]. Gregory K. Wallace, Multimedia Engineering Digital Equipment Corporation Maynard, Massachusetts, "The JPEG Still Picture Compression Standard", Submitted in December 1991 for publication in IEEE Transactions on Consumer Electronics.
- [16]. "Welcome to the JPEG Tutorial!" <http://cobweb.ecn.purdue.edu/~ace/jpeg-tut/jpegtut1.html>
- [17]. "Linux", <http://easymamecab.mameworld.net/html/svgalib.htm>
- [18]. J. Postel, "User Datagram Protocol", RFC 786, 28 August 1980
- [19]. Prepared by Defense Advanced Research Projects Agency, "Transmission Control Protocol", RFC 793, September 1981
- [20]. M. Shiraz Baig, "Beginning Internet Communication Programming through X Windows on Linux using GTK+", Educational Publishers, New Urdu Bazar, Rawalpindi

