VoIP Client for Multi-core Server Enhancing Quality of Real Time Service Delivery

Jemil Bereka

Degree project in Electronic and Computer Systems
KTH-The Royal Institute of Technology
Stockholm, Sweden
April 2012

TRITA-ICT-EX-2012:103
Supervisor: Dr. Iyad Al Khatib
Examiner: Professor Ahmed Hemani
**Abstract**

Voice over Internet Protocol (VoIP) is a fast growing service in communication technology. Due to the cost-effectiveness, many organizations have been deploying VoIP technology for their teleconferencing and video conferencing services. In recent decades, various types of client and server applications have been developed, and different application protocols have been standardized. However, most of VoIP applications were developed for single core architecture. The increasing demand of VoIP services results in increasing number of users who need a reliable and efficient communication.

In this paper, a VoIP client has been designed and implemented which is optimized for best real time service delivery and increases the number of concurrent users. The VoIP client has been implemented for multi-core processor based on Thread-level Parallelism (TPL), where each task is processed simultaneously. This paper has also focused on studying and improving the performance of RTP protocol by using multiple data at the same time based on the already designed iRTP protocol. The results show that, the presented VoIP client doubles performance when multiple data is used at the same time.
Acknowledgments

First I want to thank “Allah” my creator, who gave me the power to finish the thesis.

I would like to express gratitude to my supervisor, Dr. Iyad Al Khatib, for his endless guidance and support. I would also like to thank my examiner, Professor Ahmed Hemani.

Special thanks to my family especially my wife, Leyila, for her continued support and encouragement. Once again I would like to extend warm thanks to all friends especially Hawa and Shumet for their motivation and support.
# Table of Contents

Abstract ....................................................................................................................................... i
Acknowledgments ...................................................................................................................... ii
List of Figures ............................................................................................................................ v
List of Tables .............................................................................................................................. v
Terminologies ................................................................................................................................ vi

1 Introduction ........................................................................................................................ 1
   1.1 Motivation ................................................................................................................... 1
   1.2 Project Goals ................................................................................................................ 1
   1.3 Organization of the Thesis ........................................................................................... 1

2 Literature Review ............................................................................................................... 2
   2.1 Session Initiation Protocol (SIP) ................................................................................. 2
       2.1.1 SIP Messages ........................................................................................................ 2
           2.1.1.1 Request ......................................................................................................... 2
           2.1.1.2 Response .................................................................................................... 2
       2.1.2 SIP Architecture Components .......................................................................... 3
           2.1.2.1 SIP User Agent (UA) .................................................................................. 3
           2.1.2.2 OpenSIP Server (OpenSIPS) ...................................................................... 3
       2.1.3 SIP Call Examples ............................................................................................... 4
       2.2 Session Description Protocol (SDP) ........................................................................ 5
       2.3 Real-Time Transport Protocol (RTP) ....................................................................... 5
       2.4 Real-Time Transport Control Protocol (RTCP) ....................................................... 6
       2.5 Audio Codec .............................................................................................................. 7
       2.6 Modular Programming: ........................................................................................... 7
       2.7 Parallel Programming ............................................................................................... 7
       2.8 Multi-core Hardware ............................................................................................... 7

3 Implementation of VoIP Client .......................................................................................... 8
   3.1 iRTP Header ................................................................................................................ 8
   3.2 Sender .......................................................................................................................... 9
       3.2.1 Flow Chart ............................................................................................................. 9
           3.2.1.1 Main Module ............................................................................................... 10
           3.2.1.2 Sender Socket Module .............................................................................. 11
           3.2.1.3 Read Input File Module .......................................................................... 11
           3.2.1.4 Capture Voice Module ............................................................................. 11
           3.2.1.5 Audio Setting ............................................................................................ 11
           3.2.1.6 Build Packet Module ............................................................................... 12
           3.2.1.7 Send Packet Module ................................................................................... 12
List of Figures

Figure 1: SIP architecture ........................................................................................................ 3
Figure 2: SIP call example ........................................................................................................ 4
Figure 3: RTP encapsulation [5]. ............................................................................................. 5
Figure 4: RTP header [6]. ........................................................................................................ 5
Figure 5: iRTP header [15]. ..................................................................................................... 8
Figure 6: Sender flow chart ..................................................................................................... 10
Figure 7: Retransmission flow chart ....................................................................................... 13
Figure 8: Receiver flow chart .................................................................................................. 14
Figure 9: Send acknowledgment ............................................................................................ 16
Figure 10: Network topology .................................................................................................. 17
Figure 11: Performance of iRTP and RTP audio and data ...................................................... 21
Figure 12: Back to back delay of RTP and iRTP ..................................................................... 22
Figure 13: Performance of iRTP and RTP only audio ............................................................. 22

List of Tables

Table 1: System specification ................................................................................................. 17
Table 2: Isolated scenario audio and data ............................................................................... 19
Table 3: Isolated scenario only audio ...................................................................................... 19
Table 4: 50% background overloaded traffic audio and data ................................................. 20
Table 5: 90% background overloaded traffic audio and data ................................................. 20
Table 6: 50% background overloaded traffic only audio ....................................................... 20
Table 7: 90% background overloaded traffic only audio ....................................................... 20
Terminologies

VoIP        Voice over Internet Protocol
TPL         Thread-level Parallelism
RTP         Real-Time Transport Protocol
RTCP        Real-Time Transport Control Protocol
SIP         Session Initiation Protocol
SDP         Session Description Protocol
iRTP        Iyad Real-Time Transport Protocol
UDP         User Datagram Protocol
TCP         Transmission Control Protocol
UA          User Agent
UAC         User Agent Client
UAS         User Agent Server
OpenSIPS    Open Session Initiation Protocol Server
NAT         Network Address Translation
IP PBX      Internet Protocol Private Branch Exchange
ACK         Acknowledgment
SACK        Selective Acknowledgment
ITU         International Telecommunication Union
DSP         Digital Signal Processor
Chapter-1

1 Introduction
This chapter presents the motivation, project goals and organization of the thesis work.

1.1 Motivation
In recent decades, the demand for multimedia services such as IP telephony, video conferencing and video streaming increased enormously. Majority of real-time traffic is streamed by Real time transport protocol (RTP), which is a vital part of VoIP. Our target is to study and improve the performance of RTP. With the aim of doing that, we implement the enhancing version of RTP protocol for best real-time and efficient communication. RTP transport only single data on a single packet. In this thesis we have enhanced RTP so that it can carry multiple data such as audio, video and control information on a single packet.

1.2 Project Goals
The purpose of this project is to design and implement a VoIP client & server (receiver) that is optimized for best real time service delivery and increase the number of concurrent users. The major goals to be achieved in this project are:

• The VoIP client & server (receiver) will be implemented in a modular way, so that each module processed concurrently. The VoIP client is implemented for multi-core processor.
• The implementation of the VoIP client and server that utilize an enhanced real-time protocol.
• Implement the already designed iRTP (enhanced real-time protocol) that caries audio, video, data, and control information in a single packet.

1.3 Organization of the Thesis
This thesis is organized as follows:

Chapter 2 provides a detail literature on VoIP related protocols, audio codec, modular programming, parallel programming and multi-core hardware. Chapter 3 presents the design and implementation of the VoIP client and server (receiver) that we have implemented in this project. Chapter 4 presents the Network Topology and System Specification for the experiment. Chapter 5 discusses the result and analysis of the solution. Chapter 6 presents the conclusion and future work. Chapter 7 lists references being used.
2 Literature Review

In this section, we will explore literatures that are more relevant to the topic. We have reviewed the following topics: Session Initiation Protocol SIP Server, Real-time Transport Protocol, Real-Time Transport Control Protocol, Session Description Protocol, audio codec, modular programming approach, Parallel programming and multi-core hardware.

2.1 Session Initiation Protocol (SIP)

Session Initiation Protocol (SIP) is an application layer signaling control protocol that can establish, modify, and tear down multimedia sessions between two endpoints. It is used to initiate Internet telephony calls, instant messaging sessions and it can also used in video conferencing session [1].

According to [1], SIP supports five methods of establishing and terminating multimedia communications. These are:

- User location: find out the location of the target endpoint.
- User capabilities: decide the type of media or audio codec to be used in the communication through Session Description Protocol (SDP).
- User availability: find out the availability of the target client.
- Session setup: "ringing", setup a session between two clients.
- Session management: manage session, such as transfer, termination and modifying session parameters.

2.1.1 SIP Messages

There are two different types of SIP messages: requests and responses.

2.1.1.1 Request

According to RFC 3261, there are six basic SIP request:

- REGISTER: Used by a user agent to indicate its current location.
- INVITE: Used to set up a session between two user agents. Usually send by user agent clients.
- ACK: Confirms an INVITE message.
- CANCEL: Terminates awaiting request.
- BYE: Terminate the session by either the caller or callee.
- OPTIONS: is used to requests the capabilities of a caller or server.

2.1.1.2 Response

According to RFC 3261, response codes are grouped in the following class:

- 1xx: informational responses, such as 100, which means trying.
- 2xx: indicates success of request, such as 200, which means OK.
- 3xx: redirection of the call. The call has been redirected to different destination.
- **4xx**: this class indicates a request failure.
- **5xx**: this class indicates a server error.
- **6xx**: global failures responses. The request cannot be accomplish by any server.

### 2.1.2 SIP Architecture Components

From architecture standpoint, there are two basic components of SIP: SIP user agent (UA) and SIP network server. As depicted in Figure 1, the SIP server is an intermediate device.

![SIP architecture](image)

**Figure 1: SIP architecture.**

#### 2.1.2.1 SIP User Agent (UA)

A SIP user agent (UA) is logical entity that initiates and answers calls between two endpoints. UA is divided into two categories: User Agent Client (UAC) is SIP client that sends INVITE message to initiates a call. User Agent Server (UAS) that respond to SIP requests and answers calls. UA can be a phone, software or any device.

#### 2.1.2.2 OpenSIP Server (OpenSIPS)

As shown in Figure 1, SIP Server is an intermediary entity that facilitates exchanging of information between two user agents. The server also registers the current location of user agents (clients).

OpenSIPS (Open SIP Server) is a well known open sources implementation of SIP server. It can be used in several situations such as: SIP load balancing, instant messaging, user registration, presence server and NAT traversal. OpenSIPS is highly scalable, extremely fast in
forwarding requests and can handle tens of thousands of calls per second on a single server [3].

2.1.3 SIP Call Examples
Figure 2 shows, a SIP call between two user agents through a SIP server.

A SIP call session between Alice and Bob SIP phones is established as follows:
- The calling phone (Alice) sends “INVITE” to Bob through the proxy server. Both are already register in the proxy server.
- Bob sends to an information response “100 Trying”
- When Bob’s phone starts ringing a response “180 Ringing” is sent back to Alice through the proxy server. Alice can hear the ringing tone.
- When Bob picks up his phone, sends a “200 Ok” responses back to Alice.
- Alice response ACK acknowledgement.
- Both Alice and Bob can hear each other.
- Any one of the user hang up, the phone sends BYE.
- Response with “200 Ok” when receive the “BYE” request.
2.2 Session Description Protocol (SDP)
As the name suggests, Session Description Protocol (SDP) is an application layer protocol that describes the parameters of multimedia sessions, as it is defined in RFC 2327. SDP is encapsulating within a SIP message body.
SDP includes the following information [7]:
- Session name.
- Type of stream used (audio, video, etc).
- Type of codec.
- Port number.
- Contact information.

2.3 Real-Time Transport Protocol (RTP)
The Real-time Transport Protocol (RTP) is an application layer protocol that defines a standardized packet format for delivering multimedia sessions over the data network. RTP was developed by the Audio/Video Transport working group of the Internet Engineering Task Force (IETF) standards organization [4]. RTP is used to provide end-to-end delivery services for data that has real-time properties, such as telephony, video teleconference applications, video streaming and web-based push-to-talk features.

<table>
<thead>
<tr>
<th>IP</th>
<th>UDP</th>
<th>RTP</th>
<th>Voice payload</th>
</tr>
</thead>
</table>

Figure 3: RTP encapsulation [5].

RTP does not have a delivery mechanism; it must be used with Transport Protocol, TCP or UDP. Real-time multimedia applications are less sensitive to packet loss, but very sensitive to delays, for such type of application UDP is a better choice than more complex and slow TCP. Figure 3 shows, RTP encapsulated in UDP.

<table>
<thead>
<tr>
<th>0</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>P</td>
</tr>
<tr>
<td>X</td>
<td>CC</td>
</tr>
<tr>
<td>M</td>
<td>PT</td>
</tr>
<tr>
<td>sequence number</td>
<td></td>
</tr>
<tr>
<td>time stamp</td>
<td></td>
</tr>
<tr>
<td>SSRC</td>
<td></td>
</tr>
<tr>
<td>CSRC</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: RTP header [6].
The RTP header has a minimum size of 12 bytes. According to [6], RTP header has the following format:

Version (V): 2 bits.
Indicate the version of the protocol. The version of RTP used nowadays is version two (2).

**Padding (P):** 1 bit.
If the padding bit is on, the packet includes one or more additional padding bytes at the end of the RTP which are not part of the payload.

**Extension (X):** 1 bit.
If the extension bit is on, the fixed header is followed by one extension.

**CSRC count (CC):** 0 to 15 items, (32 bits).
Defines the list of contributing sources that. The number is given in the CC field. Maximum 15 contributing sources are identified.

**Marker (M):** 1 bit.
If it is set, it means that the current payload has some important information to the application layer.

**Payload Type (PT):** 7 bits
Indicates the current encoding format used by the RTP. By using this field the sender informs to the receiver if encoding is changed in the middle of the call.

**Sequence number: 16 bits**
A random number increment by one for each packet sent. The receiver restore arrived packet according to their sequence number and detect packet lost.

**Timestamp: 32 bits long**
The timestamp indicates the sampling instant of the RTP. This sampling instance is used to calculate the jitter and to play back the received packets in the correct timing order.

### 2.4 Real-Time Transport Control Protocol (RTCP)
The Real-Time Transport Control Protocol (RTCP) is a sister protocol of the RTP. Typically RTP is sent on an even numbered port but RTCP is sent over the next higher odd number port of RTP [8]. Its functionality is defined in RFC 3550. RTCP monitors and reports quality of service between sender and receiver. RTCP messages are sent periodically from all participants in the session. The total bandwidth used by RTCP should not be more that 5% of the total session bandwidth [6].

**Types of RTCP packets [8]**
- **SR:** Sender report, statistics from contributors that are active senders.
- **RR:** Receiver report, statistics from contributors that are not active senders.
- **SDES:** Source description items, information about the source data.
2.5 Audio Codec
Encoding is a process of converting voice signals into digital data. This digitized data can be stored in the computer and transmitted over a data network. When the digitized data reaches its destination it has to be decoded back to analog state, so that the recipient hears and understands it correctly. This encoding/decoding is achieved by codec programs.

There are different types of codec described in RFC 3551. G711 is commonly used codec; it uses pulse code modulation (PCM). G711 has sampling rate of 8,000 samples per second and 8 bits is used to represent each sample, resulting in a 64 kbps bit rate [10].

2.6 Modular Programming:
Modular programming is an important software development technique for large-scale programs. The idea is dividing a large scale program into different manageable modules. Each module contains all needed source code. Modules are independent of each other, they accomplish a specific task but they can exchange data through parameter argument, global variables and by returning a local variable [11].

2.7 Parallel Programming
Parallel programming: Is a modern way of computing. In this way large and time consuming programs are divided into different independent programs. Each piece of program is solved simultaneously and independently without waiting each other in different core [12].

2.8 Multi-core Hardware
"A multi-core processor is an integrated circuit in which multiple processors have been embedded on a single computing component" [13]. These multiple cores can run multiple processes (programs) at the same time. The presented VoIP client is implemented based on Thread-level Parallelism (TPL). This will achieve parallel execution of programs on different processors, which is mentioned in section 2.7.
Chapter-3

3 Implementation of VoIP Client
This chapter presents the modular implementation of VoIP client and enhanced RTP for multi-core processor, which is named iRTP for iyad Real Time Protocol [15]. The implementation is based on Thread-level Parallelism (TPL), where each module executes with the main thread [14]. Section 3.1 presents the iRTP as a protocol and its packet header that describes different parameters needed to create the iRTP packet. Section 3.2 presents the sender that sends the voice and the data to the recipient. Section 3.3 presents the receiver. The iRTP idea, design and protocol invention (header shown in Figure 5) is created by Dr. Iyad Al Khatib and presented firstly in technical report referred to in [15].

3.1 iRTP Header
The structure of iRTP header is shown in Figure 5, and it is based on RTP. iRTP is designed to carry voice, data and control information in a single packet. An IP packet encapsulates a UDP packet, the UDP packet encapsulates an iRTP, and the iRTP packet encapsulates RTP header, data, audio and control information.

```
<table>
<thead>
<tr>
<th>Ethernet Header</th>
<th>IP Header</th>
<th>UDP Header</th>
<th>iRTP packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 bytes</td>
<td>20 Bytes</td>
<td>8 Bytes</td>
<td>≤ 1472 Bytes</td>
</tr>
</tbody>
</table>
```

```
iRTP Header
17 – 75 bytes 2 bytes 4 Bytes AD ... 2 bytes 4 Bytes AD
```

```
RTP header  iRTP Version  type  Future use bit  iRTP packet size
12 – 72 bytes 3 bits 3 bits 2 bits 2 bytes
```

*Figure 5: iRTP header [15]*
The iRTP protocol is implemented on top of UDP, which does not support guaranteed delivery of packets. To overcome the unreliable nature of UDP, we implemented reliable functionality on the application layer. The reliable implementation is described in the ‘Send Acknowledgment’ and ‘Retransmission’ modules.

The iRTP header includes the following parameters:

- RTP header: contains information about the voice.
- iRTP version: the version number.
- Type: types of data used in the session.
- Size: the total size of iRTP packet.
- Future use: for future use and should be set to zero.

### 3.2 Sender

The sender is an entry implemented in the user agent client. The main task of the sender is to establish a session, send voice and data to the recipient. To achieve this functionality, different kinds of tasks have been implemented. These are reading file, encoding a voice, building a packet, sending the packet and retransmit packets if the packet lost in the network.

#### 3.2.1 Flow Chart

Figure 6 presents the flow chart showing the flow of the sender. It also shows how different modules interact with each other to achieve the objective of the sender. Modules depicted in the flow chart are described in the next subsections.
3.2.1.1 Main Module
This section presents the main module that starts the execution of the programs. The main module is responsible for controlling the flow of the program. All other sub modules are called from the main module. Then, the data generated by all modules are returned to the main module.

The main module has three command line arguments. When the program starts execution, the user is required to input IP address, port number of the receiver and the data type.

The following tasks are accomplished in this module:

- Accept and parse the command line arguments.
- Open/close the input file.
- Open/close the DSP sound device.
- Start execution of all threads (parallel program that runs concurrently with the main program).
3.2.1.2 Sender Socket Module
This module has two tasks: first it creates a sender datagram socket, which is used to send packets and receive acknowledgments from recipients. The second task is that it initializes the destination socket address (sockaddr), which is used by the sender to send the packets to a remote socket.

To initialize the destination socket address, the module needs four parameters. The sender program accepts the destination port number and IP address from the command line argument when the program runs. The parameters are explained in the “Receiver Socket” section.

A successful execution of this module returns the socket descriptor (a non negative integer) to the ‘main’ module. The returned socket descriptor will be stored as a global variable, because it will be used by ‘Send Packet’ and ‘Retransmission’ modules. If failed to create the socket, returns a negative integer and display a message.

3.2.1.3 Read Input File Module
This module reads data from an input file and stores it in a temporary buffer. The successful execution of this module will return the buffer that holds the data and its size.

3.2.1.4 Capture Voice Module
This module captures voice (an analog signal) from microphone input, converts the voice signal to digital data and stores it in a buffer.

We have used the G.711 ITU standard codec [16], also known as pulse code modulation (PCM). According to RFC3551, G.711 codec default audio length is 20 milliseconds. This means each packet holds 20 milliseconds long audio in its payload. Packets are sent every 20 milliseconds.

To capture the audio, we used Digital Signal Processor (DSP) sound device, which is found in '/dev/dsp' Linux device directory. In order to define the type of codec we have used ‘audio setting’ module.

This module is called by the ‘main’ module. In each execution of the module, it returns a buffer that contains digitized audio and its length to the “main module”.

3.2.1.5 Audio Setting
This function is responsible for defining the type of audio codec used in encoding. As we discussed in the above sections, we use G.711 codec. G.711 codec takes 8,000 samples per second. Each of these samples is represented in 8 bit [16].

We use ‘ioctl’ system call to define the type of audio codec. The operation consists of three system calls:

- The first system call set the sampling rate, which is 8000 samples per second.
- The second system call sets the sampling size to 1 byte (8 bits).
- The last system call sets the channel of the sound device to ‘mono’ mode.
3.2.1.6 Build Packet Module
This module creates a new iRTP header. First, it initializes the iRTP header and the encapsulated RTP header and returns a pointer to iRTP header. The following parameters are assigned in this module:

- Set the version if iRTP. The current value is 1.
- Set the payload type for the iRTP packet.
- Set the payload size for the iRTP packet. This is usually computed by the ‘main’ module.
- Initialize the RTP header which contains information about the audio payload.

3.2.1.7 Send Packet Module
This module is responsible for sending the packet to the recipient. The module sends packets through an unconnected socket as the destination socket is specified. The destination IP address and port number are defined earlier in the socket module, so this module will get the destination socket address from ‘main’ module.

The following parameters are required to send the data:

- The destination socket address to which the data is to be sent.
- The buffer containing the data to transmit.
- The length of the buffer.
- The socket descriptor.

3.2.2 Retransmission Module
This module is responsible for retransmission of lost packets to the receiver. Unlike TCP [17], the sender does not require acknowledgment for each sent packet. If the packet does not reach its destination, the receiver will send acknowledgment to inform packets are lost in the transmission. The sender recognizes which packets are lost via acknowledgment. Then the sender will retransmit the lost packet to the receiver.

This module is a concurrent program that runs in parallel with the main thread. The module starts execution when the program starts and ends when the session ends. The flow chart of this module is shown in Figure 7.
The following process is required to perform the retransmission:

- The receiver code makes a system, which receives acknowledgment packet from a socket. The acknowledgment packet holds list of sequence numbers that represent lost packet.
- Extract the received acknowledgment packet to get the list of sequence numbers.
- According to the list of sequence numbers, we build a retransmitted packet that holds list of lost packets.
- The sending system call is used to send the retransmitted packet to the receiver. The same socket descriptor is used to send packet and receive acknowledgment packet.

3.3 Receiver
The receiver is a network end-point. It is implemented in the recipient side, and it acts as a server. The main objective of the receiver is to receive audio and data from a network. To
carry out this task, different kinds of modules are implemented. These are opening a socket, receiving a packet from the socket, extracting the data from the packet, decoding the voice, displaying the data, and sending acknowledgment packets if the packets get lost in the network.

3.3.1 Flow Chart
Figure 8 presents the flow chart showing the flow of the receiver. It also shows how different modules interact with each other to achieve the objective of the sender. Modules depicted in the flow chart are described in the next subsections.

![Flow Chart](image)

3.3.1.1 Main Module
The implementation of this module is similar to the sender ‘main’ module. This module has two command line arguments. When the program starts execution, the user is required to input a port number for the receiver and the input data type.
3.3.1.2 Receiver Socket Module
The module creates a receiver socket. The receiver socket is used to receive packets coming from the sender host. It is also used to send acknowledgment packets to the sender host.

3.3.1.3 Receive Packet Module
This module is responsible to receive data from a network connection and it also captures the address from where the data was sent.

The following parameters are required to receive the data:

- The source socket address in which the data was received from remote host. This address is used to send acknowledgment packet to the sender if lost packet happens.
- The socket descriptor.
- A buffer that will hold the received data. This buffer can store the maximum length of iRTP payload.
- The length of the buffer.

3.3.1.4 Parse Module
After the packet arrives at the recipient host, the voice and the text data have to be extracted from the packet. The iRTP header is extracted and the value stored in the ‘main’ module. The size of iRTP packet and payload are mandatory parameters for this module.

The module extracts the data from iRTP payload. For each iterate, the data is processed according to the behavior of the data. If the payload is voice, we call ‘Decode’ module so that the voice will be decoded and played. If the payload is data, ‘Display’ module will be called. The ‘Display’ module gets the text data from ‘parse’ module and then the data will be displayed on a screen.

3.3.1.5 Decode Voice Module
The received audio packet is represented in digital form. In order for the audio to be listened and understood by humans, it has to be decoded back to its original voice (an analog signal). This process is called decoding. This module is responsible to decode the audio to its original analog signal and play back.

Since there are different types of codec available, the sender and the receiver must agree on the type of codec used in the session. Usually both the sender and the receiver exchange the codec information through RTP. The RTP payload type defines the type of codec. The ‘audio setting’ module is responsible to define the type of the codec.

This module receives a character buffer array that holds a sequence of samples taken from the audio signal and the size of the buffer. The ‘write’ system call is used to send audio data to the DSP sound device and this will result a voice that will be heard by humans.

3.3.2 Send Acknowledgment Module
This module is responsible for handling the packet lost. As we discussed in the ‘iRTP header’ section, iRTP is used on top of UDP for better performance. UDP provides no recovery for
lost packets. To overcome this problem, we implemented a reliable functionality of the system that detects lost packet in the transmission. The receiver is responsible for detecting packet lost and informs the sender about the packet lost via acknowledgment.

We took inspiration from TCP, and implemented in an improved way. Unlike TCP, the receiver will send acknowledgment to the sender only if packets are lost in the transmission. The system we implemented is selective acknowledgment (SACK), where the receiver can recognize which packet is lost as well as which packet is received last. The receiver requests retransmission of only lost packet. This implementation avoids re-sending of packets that are not lost. The flow chart of this module is shown in Figure 9.

The following procedure is required to perform the sending acknowledgment:

- When packets reach their destination, their sequence number is stored.
- The module runs a ‘while’ loop to check whether the data is received or lost in the network. This is detected by looking at their sequence number. If data is not received, it creates a list of lost packet.
- Create an SACK packet. The SACK packet will contain list of sequence numbers that represents lost packet.
- The module uses ‘sendto’ system call to send the acknowledgment packet to the sender. Then the sender recognizes the lost packet and retransmits.
4 Network Topology and System Specification
This chapter gives an overview of the network topology, the software and the hardware used to perform the experiments.

4.1 Network Topology
We used hardware components depicted in the Figure 10 below. The VoIP sender client is installed on Sender PC and the VoIP receiver server is installed on Receiver PC. On both PCs we have installed Iperf [18] to overload the network. We used Wireshark [19] and TCPdump [20] to monitor the traffic, capture and analyses the packets.

4.2 System Specification
This section will describe the system specification of the sender, the receiver and the router used in this experiment. Both the sender and receiver are connected to the switch by a 100 Mbps wired connection to their Network Cards. Table 1 gives the specification of the sender PC, the receiver PC and the router used in network set-up.

Table 1: System specification.

<table>
<thead>
<tr>
<th></th>
<th>Sender PC</th>
<th>Receiver PC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Product Name</strong></td>
<td>Acer Aspire 5738</td>
<td></td>
</tr>
<tr>
<td><strong>Processor</strong></td>
<td>Intel(R) Pentium(R) 4 CPU 2.2GHz</td>
<td></td>
</tr>
</tbody>
</table>
## Receiver PC

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Acer Aspire 5738ZG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel(R) Pentium(R) 4 CPU 2.0GHz</td>
</tr>
<tr>
<td>Number of cores</td>
<td>Dual-core</td>
</tr>
<tr>
<td>RAM</td>
<td>3GB</td>
</tr>
<tr>
<td>System Architecture</td>
<td>32 bit</td>
</tr>
<tr>
<td>Network Card</td>
<td>Fast Ethernet 100 Mbps</td>
</tr>
<tr>
<td>Operating System</td>
<td>Ubuntu 10.04</td>
</tr>
</tbody>
</table>

## Wireless/Wired Router

<table>
<thead>
<tr>
<th>Product Name</th>
<th>D-Link DIR-65</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless</td>
<td>Wireless: 802.11 b/g/n</td>
</tr>
<tr>
<td>Device Interface</td>
<td>4 10/100 LAN Ports&lt;br&gt;1 10/100 WAN Port</td>
</tr>
</tbody>
</table>
Chapter-5

5 Results

5.1 Isolated Scenario
In this scenario, we remove all external factors that affect the bandwidth of the network. Since we used LAN wired connection, we have a full control of the test environment. Based on this scenario, the total available network bandwidth is used by iRTP and RTP traffics, no external traffic affect. This gives the finest result shown in Table 2 and Table 3.

   a) Sending audio and data

Table 2: Isolated scenario audio and data.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Type of data</th>
<th>Average back to back delay in millisecond</th>
<th>Number of sockets used</th>
</tr>
</thead>
<tbody>
<tr>
<td>iRTP Audio, data and control information</td>
<td>20.471815</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>RTP, RTCP and TCP  Audio, control information and data</td>
<td>41.240729</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

   b) Sending only audio

Table 3: Isolated scenario only audio.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Type of data</th>
<th>Average back to back delay in millisecond</th>
<th>Number of sockets used</th>
</tr>
</thead>
<tbody>
<tr>
<td>iRTP Audio and control information</td>
<td>20.471815</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>RTP and RTCP Audio and control information</td>
<td>21.022893</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

5.2 Active Traffic Load Scenario
In this scenario, we have overloaded the network by 50% and by 90 % with background traffic to examine the behavior of the iRTP and RTP traffic. Iperf open source traffic generator is used to generate the background traffic. This scenario has two sections, first we generate 50Mbps traffic which consumes almost half of the total available network and then 90 Mbps traffic is generated.
a) Sending audio and data

Table 4: 50% background overloaded traffic audio and data.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Type of data</th>
<th>Average back to back delay in millisecond</th>
<th>Number of sockets used</th>
</tr>
</thead>
<tbody>
<tr>
<td>iRTP</td>
<td>Audio, data and control information</td>
<td>20.506266</td>
<td>1</td>
</tr>
<tr>
<td>RTP, RTCP and TCP</td>
<td>Audio, control information and data</td>
<td>41.316670</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 5: 90% background overloaded traffic audio and data.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Type of data</th>
<th>Average back to back delay in millisecond</th>
<th>Number of sockets used</th>
</tr>
</thead>
<tbody>
<tr>
<td>iRTP</td>
<td>Audio, data and control information</td>
<td>20.507350</td>
<td>1</td>
</tr>
<tr>
<td>RTP, RTCP and TCP</td>
<td>Audio, control information and data</td>
<td>41.431085</td>
<td>3</td>
</tr>
</tbody>
</table>

b) Sending only audio

Table 6: 50% background overloaded traffic only audio.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Type of data</th>
<th>Average back to back delay in millisecond</th>
<th>Number of sockets used</th>
</tr>
</thead>
<tbody>
<tr>
<td>iRTP</td>
<td>Audio, data and control information</td>
<td>20.506266</td>
<td>1</td>
</tr>
<tr>
<td>RTP and RTCP</td>
<td>Audio and control information</td>
<td>21.051658</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 7: 90% background overloaded traffic only audio.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Type of data</th>
<th>Average back to back delay in millisecond</th>
<th>Number of sockets used</th>
</tr>
</thead>
<tbody>
<tr>
<td>iRTP</td>
<td>Audio and control information</td>
<td>20.507350</td>
<td>1</td>
</tr>
<tr>
<td>RTP and RTCP</td>
<td>Audio and control information</td>
<td>21.100579</td>
<td>2</td>
</tr>
</tbody>
</table>
The graph in Figure 11 above has been constructed based on Table 2, Table 4 and Table 5. The shows the performance comparison between RTP and iRTP protocols when multiple data is used in a session. The results show that iRTP has less back to back delay than RTP. iRTP improves the performance by 100% which is very significant for multimedia applications. On the other hand, iRTP reduces the number of sockets from three to one. iRTP can carry multiple data at the same time on single packet, so in this case one socket for one protocol is required to make the communication. However, in RTP at least three sockets for RTP, RTCP and TCP protocols are needed to send audio, data and control information.
The important factors behind this performance improvement are sending multiple data in single packet and parallel code design which increase the performance on multi-core hardware. As illustrated in Figure 12, the total back to back delay for audio and data is the sum of D1 and D2 which gives D3. However, D4 is the back to back delay for iRTP which is less than D3. With D4 time delay, we can transmit multiple data (audio and data) at the same time. The value of Dx (which is in the range of 11 µs) is negligible compared to the values of the other delays, namely D1, D2, and D4 that are in the millisecond range. Hence, we can neglect adding Dx in our calculations without having any effect on the results.

![Figure 12: Back to back delay of RTP and iRTP](image)

**Figure 12: Back to back delay of RTP and iRTP**

In Figure 13, the total back to back delay for audio and data is the sum of D1 and D2 which gives D3. However, D4 is the back to back delay for iRTP which is less than D3. With D4 time delay, we can transmit multiple data (audio and data) at the same time. The value of Dx (which is in the range of 11 µs) is negligible compared to the values of the other delays, namely D1, D2, and D4 that are in the millisecond range. Hence, we can neglect adding Dx in our calculations without having any effect on the results.

![Figure 13: Performance of iRTP and RTP only audio.](image)
The graph in Figure 12 above has been constructed based on Table 3, Table 6 and Table 8. As depicted in Figure 12, we got a minimum performance when only audio packets are received. On average RTP has 0.56 milliseconds more delay than iRTP (iRTP protocol has 2.5% performance improvement).
Conclusions and Future Work

6.1 Conclusions
According to the goal of this project mentioned in the first chapter, the following tasks have been presented:

- An enhanced real-time protocol has been designed and implemented. This protocol can carry audio, data, and control information in a single packet. This enhanced protocol works on behalf of RTP, RTCP and TCP protocols. The enhanced protocol sends voice and data successfully in a single packet. It has a mechanism of retransmitting lost packets.
- A VoIP client program has been designed and implemented. The Sender captures voice and data from input, build a packet and send the packet over data network. The Receiver receives a packet from network, parses the packet, decodes the voice, and requests retransmission of lost packets.
- The presented VoIP client has been designed in parallel way so that each module can execute concurrently in different processors or cores.
- The results show that the presented VoIP client doubles the performance when we use multiple data at the same time.

6.2 Future Work
- The presented software has to be tested on a multi-core server, which was not possible to achieve in this project due to the scope of the project and time limitation.
- Due to the fact that each packet may follow a different network path, packet may arrive at the receiver in diverse time delay. To overcome this jitter problem, voice packets need to be buffed in correct timing order.
- The presented VoIP client sends audio and data. Sending video packet is a good idea to implement for future work.
7 References


[18]. Iperf, Network bandwidth measurement and performance tool, URL: http://iperf.sourceforge.net

[19]. Wireshark, Network protocol analyzer for Unix and Windows, URL: www.wireshark.org

[20]. Tcpdump, Powerful command-line packet analyzer, URL: http://www.tcpdump.org