INFLUENCE OF CODECS ON ADAPTIVE JITTER BUFFER ALGORITHM

I have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a major in Electrical Engineering.

__________________________
Ravi Pendse, Committee Chair

We have read this thesis and recommend its acceptance:

__________________________
John Watkins, Committee Member

__________________________
Krishna K. Krishnan, Committee Member
DEDICATION

To my parents
ACKNOWLEDGEMENTS

This thesis would not have been possible without the support and assistance of many people. Foremost, I would like to thank my advisor Dr. Ravi Pendse for his invaluable guidance and support throughout my Master’s program.

I would also like to thank my committee members for taking time to review my thesis report and for their suggestions.

I express my gratitude to doctoral student Mr. Amarnath Jasti for his constant support, extensive help, ideas, and suggestions throughout my thesis work.

I am sincerely grateful to Dr. Shankar Narayan, Director of Centillium Communications, for his valuable guidance during my thesis. I would also like to thank my friend Mr. Prashant N. Lodhia for his criticism and assistance.

I really appreciate the suggestions given by Mr. Prasad Calyam, a doctoral student from Ohio State University, on my thesis.

It is an immense pleasure to thank my parents, brother, and sister for their help and support. They were always there to guide and encourage me.

I would like to acknowledge all my friends who helped and supported me in completing this thesis.
ABSTRACT

Transmitting real-time audio or video applications over the Internet is a challenge in the current technology. The motivation for deploying this technology is the reduction in voice communication overheads and the enhancement of services. Voice over Internet Protocol (VoIP) provides improved features like flexible call routing, unified messaging and call center and network multimedia applications which in turn provide reduced costs and improvised services for distance learning, customer support, and remote sales presentations.

The integration of voice, video, and data encounters a variable amount of jitter and delay. Typical packet loss ranges from 0 to 20 percent and one-way delay from 5 to 500 milliseconds. Reducing jitter delay involves buffering of audio packets at the receiver so that the slower packets arrive sequentially on time at the destination. Adaptive jitter buffering at the receiver improves the quality of voice connections on the Internet.

In this thesis, the existing jitter buffer model was further enhanced by proposing a model to change the codecs from a higher bit rate to a lower bit rate during an established call session thus reducing the packet loss and improving the call performance. A simulation model is shown to support this proposal, leading to the development of a new protocol. Various tests were conducted to analyze the performance of a number of calls and bandwidths by varying one and keeping the other constant.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Thesis Organization</td>
<td>4</td>
</tr>
<tr>
<td>2. VOICE OVER INTERNET PROTOCOL SYSTEM</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Real-Time Transport Protocol (RTP)</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Real-Time Transport Control Protocol (RTCP)</td>
<td>9</td>
</tr>
<tr>
<td>2.4 H.323 Protocol</td>
<td>10</td>
</tr>
<tr>
<td>2.4.1 H.323 Protocol Stack</td>
<td>11</td>
</tr>
<tr>
<td>2.5 Audio Codecs</td>
<td>14</td>
</tr>
<tr>
<td>2.5.1 G.711</td>
<td>14</td>
</tr>
<tr>
<td>2.5.2 G.723.1</td>
<td>15</td>
</tr>
<tr>
<td>2.6 Jitter and Packet Loss</td>
<td>16</td>
</tr>
<tr>
<td>2.7 Voice over IP system with Playout Buffers</td>
<td>17</td>
</tr>
<tr>
<td>3. LITERATURE REVIEW</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>19</td>
</tr>
<tr>
<td>3.2 Algorithm 1 (Exponential Average)</td>
<td>20</td>
</tr>
<tr>
<td>3.3 Algorithm 2 (Fast-Exponential Average)</td>
<td>21</td>
</tr>
<tr>
<td>3.4 Algorithm 3 (Minimum Delay)</td>
<td>22</td>
</tr>
<tr>
<td>3.5 Algorithm 4 (Spike Detection)</td>
<td>22</td>
</tr>
<tr>
<td>3.6 Algorithm 5 (Window)</td>
<td>23</td>
</tr>
<tr>
<td>3.7 Algorithm 6 (Gap-Based)</td>
<td>25</td>
</tr>
<tr>
<td>4. H.323 CALL SETUP</td>
<td>27</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Establishing Call Between Two Terminals</td>
<td>27</td>
</tr>
<tr>
<td>4.3 Initializing Call</td>
<td>28</td>
</tr>
<tr>
<td>4.4 Establishing Control Channel</td>
<td>29</td>
</tr>
<tr>
<td>4.4.1 Master/Slave Determination</td>
<td>30</td>
</tr>
<tr>
<td>4.5 Beginning of Call</td>
<td>30</td>
</tr>
<tr>
<td>4.6 Dialogue</td>
<td>32</td>
</tr>
<tr>
<td>4.7 Ending of Call</td>
<td>33</td>
</tr>
<tr>
<td>4.8 OpenH323 Source Code</td>
<td>33</td>
</tr>
<tr>
<td>4.8.1 OpenH323 Class Architecture</td>
<td>34</td>
</tr>
<tr>
<td>4.9 Proposed Test Model</td>
<td>37</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (Cont.)

5. RESULTS 42

5.1 Introduction 42
5.2 Ethereal Packet Analyzer Tool 42
5.3 Arena 7.01 Simulation Software 43
5.4 Experimental Setup 44
5.5 Performance of Call Establishment with Change in Bandwidth 45
5.6 Test Using Arena Simulation Tool 46
5.7 Proposed Test Model of Changing Codec with Existing Adaptive Jitter Buffer 48
5.8 Comparison of Packet loss with Proposed Model 49

6. CONCLUSIONS AND FUTURE WORK 51

6.1 Conclusions 51
6.2 Future Work 52

LIST OF REFERENCES 54

APPENDIX A 58
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Basic concept of VoIP</td>
<td>5</td>
</tr>
<tr>
<td>2. VoIP protocol structure</td>
<td>6</td>
</tr>
<tr>
<td>3. RTP header packet format</td>
<td>7</td>
</tr>
<tr>
<td>4. H.323 Protocol Stack</td>
<td>12</td>
</tr>
<tr>
<td>5. VoIP system with playout buffers</td>
<td>17</td>
</tr>
<tr>
<td>6. Playout time estimation</td>
<td>20</td>
</tr>
<tr>
<td>7. Spike detection algorithm</td>
<td>23</td>
</tr>
<tr>
<td>8. Window algorithm</td>
<td>24</td>
</tr>
<tr>
<td>9. Gap-based algorithm</td>
<td>25</td>
</tr>
<tr>
<td>10. Initializing call</td>
<td>28</td>
</tr>
<tr>
<td>11. Establishing control channel</td>
<td>29</td>
</tr>
<tr>
<td>12. Beginning of call</td>
<td>31</td>
</tr>
<tr>
<td>13. Dialogue during call</td>
<td>32</td>
</tr>
<tr>
<td>14. Relationship of OpenH323 and PWLIB libraries</td>
<td>34</td>
</tr>
<tr>
<td>15. OpenH323 architecture</td>
<td>35</td>
</tr>
<tr>
<td>16. Experimental setup</td>
<td>44</td>
</tr>
<tr>
<td>17. Plot of average packet loss in real-time system</td>
<td>45</td>
</tr>
<tr>
<td>18. Plot of call establishment</td>
<td>46</td>
</tr>
<tr>
<td>19. Test model with G.711 codec</td>
<td>47</td>
</tr>
<tr>
<td>20. Plot of average packet loss with G.711</td>
<td>47</td>
</tr>
<tr>
<td>21. Test model for adaptive codecs</td>
<td>48</td>
</tr>
</tbody>
</table>
LIST OF FIGURES (Cont.)

22. Plot of average packet loss with adaptive codecs 49
23. Plot of comparison of packet loss 50
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNAME</td>
<td>Canonical Name</td>
</tr>
<tr>
<td>CRV</td>
<td>Call Reference Value</td>
</tr>
<tr>
<td>FEC</td>
<td>Forward Error Correction</td>
</tr>
<tr>
<td>GPL</td>
<td>General Public License</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunications Union</td>
</tr>
<tr>
<td>kbps</td>
<td>Kilo bits per second</td>
</tr>
<tr>
<td>MC</td>
<td>Multipoint Controller</td>
</tr>
<tr>
<td>MCU</td>
<td>Multipoint Control Unit</td>
</tr>
<tr>
<td>MOS</td>
<td>Mean Opinion Score</td>
</tr>
<tr>
<td>MP</td>
<td>Multipoint Processor</td>
</tr>
<tr>
<td>msec</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>NeVoT</td>
<td>Network Voice Terminal</td>
</tr>
<tr>
<td>PCM</td>
<td>Pulse Code Modulation</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>PSTN</td>
<td>Public Switched Telephony Network</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-Time Protocol</td>
</tr>
<tr>
<td>RTCP</td>
<td>Real-Time Control Protocol</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over Internet Protocol</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

1.1 Overview

Attaining high quality telephony over Internet Protocol (IP) networks is one of the essential steps in the convergence of voice, fax, video, and data communication services. Transporting voice packets on a network is not only technically difficult but also leads to complexities, requiring a better understanding of dealing with challenges of the system, such as interoperability, packet loss, delay, scalability, and reliability. Voice over IP (VoIP) is one of the emerging trends proving to be feasible for carrying voice and call signaling messages over the Internet by adopting standards like H.323, SIP, etc., designing terminals and gateways, and function globally.

VoIP is defined as the routing of voice signals over any IP-based network. It has the ability to make telephone calls like in the Public Switched Telephony Network (PSTN), and to send imitations over IP-based data networks with a suitable Quality of Service (QoS) with more advanced benefits such as cost reduction, simplification, and advanced applications. Voice communication is required in various fields from inter-office intercom to multi-point teleconferencing background. VoIP applications are used in some of the areas like inter-office trunking over the corporate intranet, remote access from a branch office, voice calls from a mobile PC via the Internet, Internet call-center access, and a PSTN gateway, which interconnects the Internet to the PSTN using a gateway. Transportation of real-time voice on the Internet is one of the immediate and demanding applications for IP networks. Real-time applications include IP telephony, voice conference, Internet radio, and video-on-demand.
Internet telephony has evolved as an important service on the edge of replacing circuit-switched telephony service in the future. The best-effort Internet service model was never engineered to handle voice traffic, since it experienced variable loss rates and delays. Numerous studies have tried to evaluate these constraints [1]. Results indicate significant variability, which depends on the position of senders and receivers, with a typical packet loss rate of 0% to 20% and one-way delay ranging from 5 to 500msec. If the packet loss rate exceeds 10% and one way delay exceeds 150msec, then speech quality deteriorates. Many mechanisms have been employed to deal with these problems. Techniques like error control and Forward Error Correction (FEC) are designed to handle packet loss and route control to playout scheduling methods, which reduces the delay variations and improves the performance of VoIP.

Integrating data and real-time multimedia traffic on packet networks continues to be a significant challenge. One of the tests of transmitting real-time voice on IP networks is to overcome the variable inter-packet delay (jitter) encountered as the packets traverse the network. Packet transmission features transmission delays, jitters, and packet losses varying dynamically, which make it difficult to handle real-time traffic in the current Internet scenario. In User Datagram Protocol (UDP) based real-time applications to compensate for variable delays, a smoothing buffer is typically used at the receiver. Received packets are first queued into the smoothing buffer to decode the influence of the network’s delay variations, which can be minimized. This is referred to playout delay or playout buffer, a mechanism used to receive uninterrupted data over IP-based networks by scheduling the packets playout time.
Typically, a VoIP application buffers incoming packets and delays their playout in order to compensate for variable network delay (jitter). Playout delay is important because it directly affects the communication quality of the application. If the delay is too short, the client treats the packet as lost, even though it eventually arrives, and if the delay is too large, it may be unacceptable to the client users. This makes it difficult to determine the existing playout delay. The playout time can be adaptive or fixed. Adaptive playout time is more flexible compared to fixed playout time. In time-varying networks, playout buffer algorithms govern the adaptive playout buffering. Adaptive buffering is necessary when the receiver is not able to select appropriate playout times, thus the end-to-end delay is high. Adequate playout delay depends on the variation of packet transmission delays between the server and client, depending on Internet network conditions.

The conflicting aspects of minimizing buffer time and delayed packet loss have resulted in various playout algorithms [1] [2] [3] for controlling playout buffers in order to enhance voice performance. These existing adaptive buffer algorithms have emphasized between delay and loss in-order to maximize the perceived speech quality. In highly variable networks, delay adjustments were not properly considered due to small variations that affected performance with respect to type of networks. Compensating for jitter is accomplished primarily through adaptive playout buffer algorithms [2] [3], which generally work by measuring the delays experienced by the packets, and updating playout delay on a talkspurt-by-talkspurt basis.
The motivation for this thesis was to enhance the existing adaptive jitter buffer algorithm [4] to perform better with minimum packet loss. The objectives of this thesis were:

1. To understand the proposed adaptive buffer algorithms [1] [4] that influences the quality of speech.
2. To enhance the existing jitter buffer algorithm [4] to change the codecs from a higher bit rate to a lower bit rate during an established call session, in order to minimize packet loss and maintain call performance.

1.2 Thesis Organization

This thesis is organized into five chapters. Chapter 2 gives an outline of the Voice over IP (VoIP) system, its advantages, and algorithms related to the topic. Chapter 3 presents the existing adaptive buffer algorithms and their analysis. Chapter 4 presents the basic H.323 call setup and the proposal of a new algorithm to achieve better performance, comparing it to the existing one with an appropriate simulation model. Chapter 5 describes the implementation, using the experimental setup, which was employed for this research and the results obtained for different scenarios. Conclusions and its future work are discussed in Chapter 6.
CHAPTER 2
VOICE OVER INTERNET PROTOCOL SYSTEM

2.1 Introduction

Voice over Internet Protocol (VoIP) is a revolutionary technology of routing voice packets over any IP-based network. Voice data flows over a general-purpose packet-switched network, instead of a traditional circuit-switched voice network. The basic concept of the VoIP application involves converting the voice signal to a digital signal that travels through the Internet, and then converting it back from a digital to an analog signal at the destination.

![Diagram of VoIP](image_url)

ADC – Analog to Digital Converter
DAC – Digital to Analog Converter

Transmitting a voice packet is accomplished by implementing certain protocols or standards, as illustrated in TCP/IP [5] layer distribution. H.323 is a popular standard in the application layer for sending voice and video over IP. This comprises of two extensively used protocols for handling audio and video signals, the Real-Time Protocol (RTP) and the Real-Time Control Protocol (RTCP).

VoIP consists of different applications. These applications include branch office use, inter-office trunking, cellular network inter-networking, and the PSTN. Fig. 2 illustrates the network protocols that are currently used to implement VoIP.
Audio (H.323)
RTP, RTCP, RSVP
UDP/TCP (Transport) Layer
IP (v4, v6) (Network) Layer
Data Link Layer
Ethernet (Physical) Layer

Fig. 2 VoIP protocol structure [9]

2.2 Real-Time Transport Protocol

Real-time transport protocol [6] transmits real-time data applications (audio, video or simulation data) by offering end-to-end network transport functions through unicast or multicast network services. In RTP, there is no address resource reservation and there is no guaranteed QoS for real-time services. The data transport is improved by a real-time control protocol. This allows supervising of data delivery in a way which is scalable to large multicast networks moreover providing minimal control and identification functionality. The RTP and RTCP are designed in such a way that it is apart from the transport and network layers which are laid below.

The RTP distinguishes between multiple senders in a RTP multicast stream, preserves the timing relationship between packets, enables timing synchronization between media streams, sequences the data to identify lost packets, identifies media types and does not provide or ensure QoS. The format of the RTP is shown Fig. 3 and the header explanations follow:

Version (V): Specifies the version of RTP.

Padding (P): Is needed by some encryption algorithms with a fixed block size or for carrying several RTP packets at a lower-layer data unit. The packet may hold one or more
additional padding octets at the end that are not part of payload when a padding bit is placed.

<table>
<thead>
<tr>
<th>V=2</th>
<th>P</th>
<th>X</th>
<th>CC</th>
<th>M</th>
<th>Payload Type</th>
<th>Sequence number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Timestamp</td>
</tr>
</tbody>
</table>

Synchronization Source Identifier (SSRC)
Contributing Source Identifier (CSRC)
Profile Dependent | Size

Data

0  2  4  6  8  10 12 14 16 18 20 22 24 26 28 30

Fig. 3 RTP header packet format

**Extension (X):** Extension of header, when set, the fixed header is followed by exactly one header extension with a defined format.

**CSRC:** Identifies the contributing sources for payload contained in the packet.

**CSRC Count (CC):** Indicates the number of CSRC header fields followed by the fixed byte header.

**Marker (M):** Identifies the beginning of the “speech burst” for VoIP applications. The interpretation of the marker is defined by a profile.

**Payload Type (PT):** Identifies the format of RTP payload and determines its interpretation by the application.

**Sequence Number:** Detects packet loss at the receiver, which is used to restore packet sequence. Increments by one for each RTP data packet sent.

**Timestamp:** Reflects the sampling instant of the first octet in the RTP data packet. Increments by one for each codec sample interval and enables correct timing of playout.

**SSRC:** Identifies the synchronization source. This identifier is chosen randomly, with the intent that no two synchronization sources within the same RTP session will have the same SSRC identifier.
The RTP is designed to allow receivers to compensate for jitter and unorganized data packets introduced in the IP networks. It defines a way to format IP packets carrying isochronous data and includes the following:

- Information on the type of data transported.
- Timestamps.
- Sequence numbers.

The RTCP used with the RTP conveys some feedback on the quality of the transmission and information on the identity of the participants. Both protocols are primarily used on top of a UDP network layer because the Transmission Control Protocol (TCP) re-transmission scheme is not adapted for data that needs to be carried out with a very low latency.

The RTP carries a sequence number and a timestamp. An audio application uses the sequence number and the timestamp to manage a reception buffer. When a new RTP packet arrives, it is placed in the buffer in the appropriate position depending on its sequence number. If the packet does not arrive on time and is still unavailable at playback time, the application makes a copy of the last played out frame of the packet and plays it back long enough to catch up with the timestamp of the next received packet.

The payload of each RTP packet is the real-time information contained in the packet. The format must be specified by the application or the profile of the RTP in use. Instead of analyzing the content of the payload to distinguish the particular format, the header of each RTP packet contains a payload-type identifier.
2.3 Real-Time Control Protocol

The RTCP is based on the sporadic transmission of control packets to all users in
the session who are employing the same distribution method as the data packets [6] [7].
This protocol offers multiplexing of the data and control packets by using different port
numbers with the UDP. The main function of the RTCP is to provide feedback on the
quality of data distribution related to flow and congestion control functions of other
transport protocols. It also transmits a transport-level identifier for an RTP source called
Canonical Name (CNAME) which maintains a track of every user.

RTCP packets vary for each type of information, including the following:

- **SR** – Sender Reports containing information of transmission and reception
  statistics for active senders.
- **RR** – Receiver Reports containing reception information statistics for listeners
  who are not active senders.
- **SDES** – Source Descriptions of various parameters about the source,
  including CNAME.
- **BYE** – Indicates end of participation when one or more sources become
  inactive.
- **APP** – Functions specific to application.

Multiple RTCP packets are compounded in a single transport protocol packet like
UDP, where each RTCP “message” of several RTCP messages contains enough
information to be decoded properly. This is helpful for saving overhead bandwidth used
by the transport protocol header.
2.4 H.323 Protocol

H.323 is an International Telecommunications Union (ITU) recommended set of protocols for voice, video, and data conferencing over packet-based networks [8] [9]. It can be used in addition to any packet-based network transport mechanism like Ethernet, ATM, and Frame Relay to provide real-time multimedia communication. It uses IP for inter-network conferencing.

H.323 is a broad and flexible signaling protocol, which specifies for real-time point-to-point audio communication between two terminals on a packet-based network which does not guarantee QoS. The scope of H.323 is broader and includes inter-network multipoint conferencing among terminals that supports not only audio but also video and data communication. The framework of H.323 is complete and includes specification of the following:

- Video-conferencing terminals.
- Gateways between an H.323 network and other voice and video networks.
- Gatekeepers, the intelligent part of the H.323 network, perform registration of terminals, call admission, etc.
- Multipoint Control Unit (MCU), Multipoint Controller (MC), and Multipoint Processor (MP) functional blocks, which are used for multiparty conferencing.

H.323 defines a number of functional or logical components:

1. **Terminal:** Terminals are H.323-capable endpoints, implemented in software on workstations. They are allocated to one or more aliases which could be a user’s name and/or telephone numbers.
2. **Gateway**: Interconnects H.323 entities (such as endpoints, MCUs, or other gateways) to other network or protocol environments, allocated to one or more aliases.

3. **Gatekeeper**: The core management entity in an H.323 environment that is responsible for access control, address resolution, and H.323 network management. Gatekeepers also provide supplementary call services and are well prepared with proxy features, enabling the routing of RTP traffic (audio and video) through the gatekeeper and T.120 traffic (data) so that no traffic is directly exchanged between endpoints.

4. **Multipoint Controller**: A logical entity that interconnects call signaling and conference control channels of two or more H.323 entities in a star topology. It cascades to allow conferences spanning across multiple MCs.

5. **Multipoint Processor**: Used to receive a media stream from an individual endpoint, and then combines them through some mixing or switching technique and transmits the resulting media stream back to the endpoint.

6. **Multipoint Control Unit**: A combination of an MC and an MP in a single device.

2.4.1 **H.323 Protocol Stack**

H.323 dwells above the basic Internet protocols (IP, IP Multicast, TCP, and UDP) which make use of integrated and differentiated services [8] along with resource reservation protocols. They interact using these control protocols for basic call signaling and conference control interactions with H.323. The H.323 protocol stack shown in Fig. 4
illustrates the H.323 services bundled into user applications, which include mandatory audio and optional video and data-sharing capabilities.

The H.323 protocol stack channels include the following:

1. **H.225 Registration, Admission, and Status (RAS):** The RAS channel is used for communication between H.323 endpoints and their gatekeeper and
for some inter-gatekeeper communication. Endpoints use RAS to register with
their gatekeeper, to request permission to utilize system resources, to have
addresses of remote endpoints resolved, etc. Gatekeepers use RAS to keep
track of the status of their associated endpoints and to collect information
about actual resource utilization after call termination. RAS offers methods
for user or endpoint authentication and call authorization

2. **H.225 Call Signaling:** The call signaling channel is used to signal call setup
intention, success, and failure, and to carry supplementary services like
multiple calls preventing the use of a dedicated TCP connection for each call
gateway built to handle multiple calls on each connection and maintain
connection. The need to open a new TCP connection is reduced providing a
better call setup time for the next call, as the endpoint may decide to maintain
the TCP connection. Call signaling messages are obtained from Q.931. The
call signaling channel is used end-to-end between caller and callee, which
voluntarily run through one or more gatekeepers.

3. **H.245 Conference Control:** The conference control channel is used to
establish and control two or more party calls. Its functions are to determine
possible modes for media exchange and to configure actual media streams.
H.245 could be used to carry user input (ex. DTMF) and facilitates exchange
of confidential media which defines the syntax and semantics for multipoint
conference operations. It creates a large volume of maintenance messages
during the call. It is a logical channel which preferably runs through one or
more gatekeeper, or directly between caller and callee.
2.5 Audio Codecs [9] [11]

Codec is a combination of coder and decoder. The coder encodes the digitized signal into more efficient coding techniques for voice into waveform coding and voice coding. In waveform coding, the amplitude samples of voice signals are encoded into binary form. With the voice coding technique, voice coders apply the properties of human speech by using a model or an algorithm. Voice coders compare each sample against a model and produce a set of parameters that achieves the original signal. These parameters are transmitted to reconstruct a voice signal at the receiver using a similar algorithm. The most popular voice coding ITU-T standards for telephony and packet voice include the following:

a. G.711 – Standardized for the 64 Kbps Pulse Code Modulation (PCM) voice coding technique.


c. G.728 – Standardized for 16 kbps voice coding, also known as Code Excited Linear Prediction (CELP).

d. G.729 – A type of CELP coding that encodes voice into an 8 kbps bit stream.

e. G.723.1 – A compression technique to condense speech or other audio signal components of multimedia service at a low bit rate, standardized for 5.3 and 6.3 kbps.

2.5.1 G.711

G.711 is the grandfather of digital audio codecs, an international standard approved in 1965. It encodes an audio stream on a 64 kbps channel with a frame length of
125μs. It is also known as companded PCM operating at an 8 kHz sample rate with 8 bits per sample. This has two forms, A-law and μ-law. An A-law PCM encoder converts 13 bit linear PCM samples into 8 bit compressed logarithmic form samples. The μ-law encoder converts 14 bit linear PCM samples into 8 bit compressed PCM samples. G.711 operates at 64 kbps (standard), 56 kbps and 48 kbps (non-standard). It has a very good Mean Opinion Score (MOS) of 4.2.

2.5.2 G.273.1 [7]

G.723.1 has been chosen as the baseline codec for narrowband H.323 communications by the International Multimedia Teleconferencing Consortium (IMTC) VoIP forum. This codec was approved in November 1995. It encodes the audio signals in a 30 msec frame with a look-ahead of 7.5 msec. G.723.1 is associated with two bit rates, 5.3 kbps (each frame requires 158 bits aligned to 20 octets) and 6.4 kbps (each frame requires 189 bits aligned to 24 octets). The higher bit rate gives a higher quality, while the lower bit rate gives good quality and provides additional flexibility for system designers. Both rates are a mandatory part of the encoder and decoder. It can change dynamically between two rates at each frame. It is also possible to use discontinuous transmission and noise filled during non-speech intervals for variable rate operation.

G.723.1 is optimized to represent speech with high quality at the specified rates with limited complexities. Music and other audio signals are not represented as effectively as speech, but can be compressed and decompressed using this coder. G.723.1 has MOS of 3.9 for 6.4 kbps and 3.7 for 5.3 kbps.
2.6 Jitter and Packet Loss

Jitter is introduced as a result of internal operations of components in the network. Queuing and buffering of the data in the network, packet re-routing, packet loss, network multiplexing, and other similar factors can cause jitter. Jitter can also be introduced at the end-user system, which is the source of network traffic. This jitter is called inclusion jitter that is introduced when certain packets are delayed before placing them in transmission slots because of the previous transmission being incomplete. Jitter needs to be regulated as the network tends to amplify it. Packet sizes also influence the magnitude of jitter. Long packet sizes increase the overall delay due to the packet-processing overhead. This is one of the reasons that multimedia applications have characteristically small packet sizes. To alleviate some of the sender side jitter, playback buffer devices can be used at the endpoints.

Packet drop can be caused by change in the inter-arrival times of the audio packets due to the intermediate router processing along the path of the packets. The packet drop value is negligible for smaller changes in the packet inter-arrival times. At the receiving end, when buffers are used for reproducing the data units, buffer overflow or buffer refreshing frequency can cause packet drop. The impact of packet drop depends on the application. For a multimedia application, dropping of some important frames might be disturbing for the end user. Selective discard of packets on the receiver end can help applications to maintain their QoS to the user.

Jitter buffer (playout buffer) is created to eliminate the aftermath of jitter from a decoded voice stream and to buffer each incoming packet for a small period before playing it out. This substitutes an extra delay and packet loss for the jitter. A fixed jitter
buffer retains a constant size, whereas an adaptive jitter buffer has the ability of adjusting its size dynamically in order to optimize the delay [12]. Adaptive jitter buffers react to an increase in jitter level or discarded events. If a discarded event is detected, then the jitter buffer size increases; otherwise, it will decrease the size of the jitter buffer. The adaptive playout mechanism makes it possible to balance the length of the buffer with the possibility of packet loss.

2.7 Voice over IP System with Playout Buffers

![VoIP system with playout buffers](image)

Fig. 5 VoIP system with playout buffers

Fig.5 represents the complete process of voice transferred from sender to receiver through the represented VoIP system. To obtain better and accurate transmission results, mechanisms such as codec, playout buffer, and voice or silence detector are set up. The challenge in transmitting real-time voice on a packet network is overcoming the variable inter-packet delay (jitter) encountered as the packet traverses the network. Networks that are originally designed to carry data cannot easily distinguish between different types of traffic. In the case of congestion, voice packets are equally delayed, just like data packets. As shown in Fig 5, voice packets are periodically generated at the sender’s codec and sent to the network, which introduces random amounts of inter-packet delay due to queuing and routing in the intermediate nodes [13]. At the receiver, a playout buffer is
used to delay the received packet in order to remove jitter and reconstruct the
deterministic inter-arrival characteristics required by the receiver’s codec.

The playout buffer at the receiver end queues and holds each received packet by
an amount of buffer time that compensates for the network delay variance without
excessively delaying the playout. If the inter-packet delay exceeds the buffer time, the
buffer will starve and the decoder will not have any packet to play. This results in an
irregularity of the voice playout. Those packets arriving later are considered to be lost.
Buffer time can be manually set or adaptively determined. Clearly, the longer the delay,
the more packets arriving before the scheduled playout time, there is a better jitter
compensation. However, long delays are undesirable since they impair human
conversation. Human conversation is tolerant of a maximum end-to-end delay of between
150 and 400 msec [3]. Different encoding schemes have varying error tolerances. A good
playout scheme should be able to trade-off playout delay and packet loss rates in order
for the voice packet to be successful.

This chapter laid the background for understanding the concept of a VoIP system
with the protocol structure. The basic protocols RTP and RTCP, which formed the basis
of this research, were discussed. H.323, the signaling protocol for real-time transmission
of voice, video, and data communication over packet-based networks, was detailed in this
section. A review of the playout buffer and its mechanism also was introduced.
CHAPTER 3
LITERATURE REVIEW

3.1 Introduction

Packets gain random delays while traversing the network. To smooth out delay jitter, the receiving host can delay the initiation of periodic playout of received packets for some time. Buffering delay is re-adjusted on a talkspurt basis:

a) Too little delay will result in packet loss.

b) Too much delay will result in unnecessary delay.

c) High jitter—buffering delay must be increased.

d) Low jitter—small amount of buffering delay is enough.

Packet loss is based on packet drop within the network and, also from packets discarded by the jitter buffer due to the arrival time exceeding the playout time. Packets might be lost due to talk burst during transit, or network congestion and insufficient buffer capacity on the path. Packet loss due to bursts is rare but might be due to network congestion since the packet loss empties the window. The network is congested if a packet is frequently lost or if timeout occurs due to packet loss.

Playout buffers are set to remove jitter in packets, which can be adaptive or fixed. If a network delay changes frequently, fixed buffers cannot adapt to it easily resulting in bad quality of speech. For each talkspurt, buffer is adjusted and set at the beginning as in [2] [3].

Jitter is caused when packets experience unequal delays when transiting a network. Fig. 6 represents the time between voice packets at the sender when they are first transmitted and at the receiver after the packets have experienced variable delays.
while transiting the network. If all packets experience the same amount of delay, then the delay among the packets are equal.

As shown in Fig. 6, X1 and X2 are much closer together at the receiver side than the sender side because X2 experiences less delay than X1 while transiting the network. Similarly, X2 and X3 are further apart at the receiver side than the sender side, since X3 experiences more delay than X2 while transiting the network. \( D_{1,2}, D_{2,3}, D_{3,4}, D_{4,5} \) are delays. The end-to-end delay is \( d_i \). Here, \( t_i \) is the time of sending \( i^{th} \) packet, \( \alpha_i \) is the packet arriving time, \( p_i \) is the playout time, and \( n_i \) is the packet’s network delay. The buffer delay is \( b_i \).

3.2 Algorithm 1 (Exponential-Average)

In this algorithm, the total delay of \( i^{th} \) audio packet is calculated based on RFC 793 [5], and one-way delay \( n_i \) is introduced by the network. The estimated packet delay
\( \hat{d}_i \) and variance \( \hat{v}_i \) during the flow are used to estimate the playout delay \( \hat{p}_i \) of the incoming packet as 

\[
\hat{p}_i = \hat{d}_i + 4\hat{v}_i
\]  

(3.1)

Estimated packet delay (mean) \( \hat{d}_i \) is given as 

\[
\hat{d}_i = \alpha \times \hat{d}_{(i-1)} + (1 - \alpha) n_i
\]  

(3.2)

Variance \( \hat{v}_i \) is given as 

\[
\hat{v}_i = \alpha \times \hat{v}_{(i-1)} + (1 - \alpha) \times | \hat{d}_i - n_i |
\]  

(3.3)

The estimation is computed each time a packet arrives but only used when a new talkspurt is initiated. This algorithm is basically a linear recursive filter and is characterized by weighting factor \( \alpha \). The value of \( \alpha \) is defined as 0.998002 according to [2]. The variation coefficient 4 controls delay or packet loss ratio.

### 3.3 Algorithm 2 (Fast-Exponential Average)

The Fast-Exponential algorithm is alike to the first algorithm but adjusts more rapidly due to small bursts of packets gaining extended delays. Here, the current packet network delay \( n_j \) is larger than \( \hat{d}_{(i-1)} \) the equation of \( \hat{d}_i \) is set as 

\[
\hat{d}_i = \beta \hat{d}_{(i-1)} + (1 - \beta) n_i,
\]  

(3.4)

Where \( \beta = 0.75 \) according to [2] [14].

The weighting factor \( \alpha \) uses two values in this algorithm. A smaller value is used for increasing trends in the delay and a larger value is used for decreasing trends. The variation in delay is the same as in the first algorithm.
3.4 Algorithm 3 (Minimum-Delay) [2]

This algorithm is based on the delay adaptation algorithm that is used in Network Voice Terminal (NeVoT) 1.6 [15]. The main aim of this algorithm is to minimize delays. Minimum delay of all packets received in the current talkspurt is used as $\hat{d}_i$ to calculate the playout delay of the next talkspurt. If $S_i$ is the set of all packets received during the talkspurt, the delay estimate $\hat{d}_i$ is computed as

$$\hat{d}_i = \min j \in S_i \{n_j\}$$

(3.5)

3.5 Algorithm 4 (Spike Detection) [2]

This algorithm is based on the spike detection mechanism. A spike is an immediate, huge increase in the end-to-end network delay followed by simultaneously arriving series of packets. The algorithm is shown in Fig. 7 When a packet arrives with a delay bigger than the threshold, the algorithm automatically switches to spike mode. If a spike is detected, the delay estimate follows very closely to adjust the slope of the spike. If the value claims to be small, the algorithm goes back to the normal mode. It would be similar to the first algorithm if it is not in the SPIKE mode, where the $\alpha$ value is set to 0.875 [2].
Fig. 7 Spike-detection algorithm

3.6 Algorithm 5 (Window) [3]

This algorithm collects the network delays of previously received packets and uses them to estimate the playout delay. The delays of the last few packets are recorded and the allocation of delay is updated with every received talkspurt. The playout delay is selected by finding the delay that represents among the last few packets. This is concluded by using a histogram which is updated. The delay of the oldest packet is removed from the histogram and the delay of new packet is added as the new packet arrives. The distribution delay is calculated using a collective sum of the frequencies which is performed only in the beginning of a fresh talkspurt.
The algorithm is shown in Fig. 8.

```c
IF (mode = SPIKE)
  IF (d^k_i \leq \text{tail} \times \text{old}_d) \quad \text{//End of spike}
    mode = NORMAL;
  ELSE
    IF (d^k_i > \text{head} \times p_k) \quad \text{//beginning of spike}
      mode = SPIKE;
      old_d = p_k;
    } ELSE
    IF (\text{delays}[\text{curr_pos}] \leq \text{curr_delay})
      count -= 1;
      \text{distr}_\text{fcn}[	ext{delays}[\text{curr_pos}]] -= 1;
      \text{delays}[\text{curr_pos}] = d^k_i;
      \text{curr_pos} = (\text{curr_pos} + 1)\%w;
      \text{distr}_\text{fcn}[d^k_i]+ = 1;
    IF (\text{delays}[\text{curr_pos}] < \text{curr_delay})
      count += 1;
    WHILE (count < w \times q)
      \text{curr_delay} += \text{unit};
      count += \text{distr}_\text{fcn}[\text{curr_pos}];
    WHILE (count > w \times q)
      \text{curr_delay} -= \text{unit};
      count -= \text{distr}_\text{fcn}[\text{curr_pos}];
```

Fig. 8 Window algorithm

This algorithm also detects spikes. When a spike is identified, it ends accumulating packet delays. When a fresh talkspurt starts during the spike, the delay of the foremost packet of a talkspurt is utilized as the playout delay for that talkspurt. The amount of recorded packet’s delay determines the sensitivity of the algorithm to the alteration in network conditions. If the delay is very little, the algorithm is prone to generate a meager assessment of the playout delay. If the delay is too big, huge amount of previous record will be kept needlessly.
3.7 Algorithm 6 (Gap-based) [13]

This algorithm, shown in Fig. 9, objects to any desired loss rate. Buffering delay is adjusted on the basis of incoming packets collected in the earlier talkspurt. The playout delay is obtained from the prearranged list of delays of the same talkspurt.

Even this algorithm operates in two modes, NORMAL and SPIKE. In SPIKE mode, playout delay of a talkspurt is the delay of the new packet of a talkspurt.

![Algorithm 6 (Gap-based)](image)

All algorithms discussed are network-based parameters of prediction of the playout delay. The aim of the existing adaptive algorithms is to reduce the end-to-end delay, and to maintain packet loss rate below 5% except for gap-based algorithm. The challenge lies in the ability to explicitly set different packet-loss tolerances in order to
achieve the desired levels of audio quality. The exponential average, fast-exponential average, and minimum delay algorithms do not adapt to SPIKE detection, and thereby take longer time to increase their delay estimate on detection of a SPIKE and also to decrease their estimate once the SPIKE is over.

The spike-detection and window algorithms are proposed to detect spikes and regulate the playout delay. Spike-detection sets a threshold to detect the beginning and end of a spike. In the window algorithm, the playout delay is the delay of the first packet of a talkspurt, when a fresh talkspurt starts through the spike. For a variable network delay, this algorithm does not adjust spikes well due to the fixed value of the algorithm head thus resulting in small and large playout delays. The insufficient prediction of playout delay may result in packet loss due to late arrival.

The gap-based algorithm computes the difference between the packet’s playout time and its arrival time. It is based on an optimum gap, which is the difference between current and optimum playout delays for the minimum amount of delay required to playout the talkspurt at the desired packet loss rate.

The adaptive jitter buffer algorithms proposed in [2] [4] was summarized in this chapter. The estimate of playout delay mainly focused on the trade-off between loss and delay. This work implemented the design motivation and performance evaluation of a buffer algorithm to perceive better speech quality.
4.1 Introduction

H.323 mainly defines the signaling needed to set up calls and conferences by choosing common codecs. The core of RTP/RCP is used to transport periodic streams to obtain feedback on the quality of the network. H.323 tackles complex problems like the email alias distribution feature of the RTCP. Establishing a simple voice call from terminal A to terminal B is discussed below [16].

4.2 Establishing Call between Two Terminals

Establishing a point-to-point H.323 call requires two TCP connections between two IP terminals for call setup, and for call control and capability exchange. The call signaling channel Q.931 initializes the TCP connection made from the caller to a well-known port on the callee which carries the call setup messages defined by H.225. Call control messages use the second TCP connection. The callee listens for a TCP connection on a dynamic port when an incoming call is received and communicates through it during the call acceptance message. The caller then establishes a second TCP connection to that port, which carries the call control messages defined in H.245. When the H.245 channel is established, the first connection is no longer necessary and can be closed by either endpoint.

The H.245 channel was used by the terminals to exchange audio and video capabilities, and to perform master/slave determination. It sends signals to open the logical channels for audio and video to create RTP sessions for media streams. The H.245
channel remains open for the duration of the conference. It is also used to signal the end of the conference.

4.3 Initializing Call

H.323 uses a subset of the Integrated Services Digital Network (ISDN) Q.931’s user to network interface signaling messages for call control. Messages like Setup, Alerting, Connect, Release Complete and Status Facility belong to the core H.323 and must be supported by all terminals. Other messages like Call Proceeding, Status, and Status Enquiry are optional.

Terminal A makes the call to Terminal B, by knowing B’s IP address (14.1.1.2). Terminal A sends a Setup message on the call signaling channel port (1720 defined by H.225) using a TCP connection to Terminal B. H.225 defines the message that contains a
protocol discriminator field set to 08H and a 2-octet locally unique Call Reference Value (CRV) chosen by the originating side. The CRV will be copied in each future message regarding the call. H.225 also contains a message type (05h for setup), a bearer capability to indicate if the call is audio or video, a called party number and suborders, a calling party number and sub-address, and a user-to-user H.323 Protocol Data Unit (PDU) that encapsulates most of the extended information needed by H.323.

4.4 Establishing Control Channel

Call control and capability exchange messages defined in H.245 are sent on the second TCP connection, where the caller establishes contact on the callee’s terminal via a dynamic port in Fig. 11.
The caller opens this H.245 control channel immediately after receiving the Alerting, Call Proceeding or Connect message. It uses a TCP connection which must be maintained throughout the call. If there is an indication of an H.245 transport address in the Setup message by the caller, the callee could have alternatively set up this channel. The H.245 control channel is unique for each call between two terminals, even if several media streams are involved for audio, video, or data. This channel is also known as logical channel 0. The first message sent over the control channel is the Terminal Capability Set, which contains information like sequence number; capability table, which is an ordered list of codec configurations that the terminal can support; and capability descriptors. The terminals send this message to each other and acknowledge it with a TerminalCapabilitySetAck message.

4.4.1 Master/Slave Determination

A master/slave determination is useful when the same function or action is performed by two terminals during a conversation and is necessary to change only one. The master is responsible for distributing the encryption keys for media channels to other terminals in H.235. Exchange of the masterSlave determination messages determines who will be the master and contains a terminalType value reflecting the terminal capabilities and a random number. The terminal-type values specified in H.323 prioritize MCUs over gatekeepers, gateways and terminals, and MC- and MP-capable units with only MC units but not with either MC or MP units.

4.5 Beginning of Call

At the beginning of a call, terminal A and terminal B must open media channels for voice as shown in Fig. 12. The data for these media channels is carried in several
logical channels which are unidirectional except for T.120 data channels. Terminal A sends an H.245 OpenLogicalChannel message to open the voice channel to B, which contains the number given to the logical channel and other parameters like type of data that would be carried. In the case of audio or video (carried over RTP), the OpenLogicalChannel message also mentions the UDP address and port where B sends RTCP receiver reports, the type of RTP payload, and the capacity to stop sending data during silences. The number of codec frames put in each RTP packet is specified by the sender here. Terminal B sends an OpenLogicalChannelAck for this logical channel as soon as it is ready to receive the data from terminal A. This message contains the UDP port number to which A should send the RTP data, and the UDP port to which A should send the RTCP data. Meanwhile, terminal B may have opened a logical channel to A following the same procedure.

![Diagram of call setup]

Fig. 12 Beginning of call
4.6 Dialogue

Now, terminal A and B can talk, and determine if they have opened video logical channels, as shown in Fig. 13.

The media data are sent in RTP packets. The RTCP SR packets sent by terminal A are used to allow terminal B to synchronize multiple RTP streams to evaluate the expected rate of RTP data and to measure the distance to the sender. RTCP receiver reports sent by terminal B allow terminal A to measure the QoS of the network between the two. The RTCP messages contain the fraction of packets that have been lost since the last RR, the cumulative packet loss, the inter-arrival jitter, and the highest sequence number received. H.323 terminals should respond to increasing packet loss by reducing the sending rate. It is noted that H.323 authorizes the use of only one RTP/RTCP port pair for each session.

There can be three main sessions between H.323 terminals—the audio session (session...
id1), the video session (session id2), and the data session (session id3), but a terminal opening more sessions cannot be prevented from the standard. Only one RTCP port should be used for each session. If there is a simultaneous RTP flow from terminal A to B and from terminal B to A, the RTCP sender and receiver reports for both flows will use the same UDP port.

4.7 Ending of Call

If terminal A ends the call, it must send an H.245 CloseLogicalChannel message for each logical channel opened by terminal A. Terminal B acknowledges those messages with CloseLogicalChannelAck. When all the logical channels have been closed, terminal A sends an H.245 endSessionCommand and waits until it has received the same message from terminal B by closing the H.245 control channel. Finally, terminals A and B must send an H.225 ReleaseComplete message over the call signaling channel if it is still open. Then, the channel is closed and the call is ended.

4.8 OpenH323 Source Code

The OpenH323 project [17] aims to create a full-featured inter-operable open source implementation of the H.323 teleconferencing protocol that can be used by personal developers and commercial users without any cost. OpenH323 development is coordinated by Quicknet Technologies but is open to everyone. The OpenH323 project consists of an OpenH323 Library as a central library, which is based on PWLib, another open source and multi-platform class library. The open source OpenH323 library is used to implement H.323 protocol for packet-based networks in multimedia communication. It is written in C++ and contains nearly 100 classes of more than 350,000 lines of source
code. Some classes represent H323 connection, various types of H323 channels, gatekeeper, and transport protocols.

Many applications have been developed based on OpenH323 library, both within and outside the OpenH323 project. They include command line H.323 Client (OhPhone), H.323 Conferencing Server (OpenMCU), H.323 Answering Machine (OpenAM), H.323 Gatekeeper (OpenGK), and H.323 Call Generator (CallGen323). Fig. 14 illustrates the inter-connection of OpenH323 and PWLib libraries, and the architecture of the applications developed on top of these libraries.

![Diagram of OpenH323 and PWLib libraries](image)

**Fig. 14 Relationship of OpenH323 and PWLib libraries**

### 4.8.1 OpenH323 Class Architecture

OpenH323 constitutes various dynamic link libraries (dlls):

1. OpenH323dll
2. Ptlibdll
3. Ptcilibdl

With these three dlls registered to the system, any of the open source OpenH323 applications, such as openphone, ohphone, openam, opengk, openmcu, and others, can be tested.

The OpenH323 Architecture is shown in Fig. 15. The architecture primarily represents the interaction in the classic “phone” application of the main classes existing in OpenH323 library.

![Fig. 15 OpenH323 architecture [17]](image)

H323Endpoint class is the basic and main object in the OpenH323 architecture hierarchy. One instance of a descendant of H3232 Endpoint class would an application have. This
application defined would set up defaults for various H323 parameters, timeouts, etc. The codecs and channel type of the application is defined in the capability table.

OpenH323 implements the behavior of a H.323 endpoint in a class named H323Endpoint, which has a number of virtual methods that allow the programmer to customize the behavior of the endpoint. The virtual methods also serve as a communication interface through which OpenH323 notifies the application about important events, like arrival of a new call, end of a call, etc.

Instances are produced by the application in H323Endpoint by one or more descendants of the H323Listener class. A descendant of this class is created for each protocol which supports. A thread is generated by each listener that monitors its protocol. When an incoming call is detected which is new, an instance of an H323Transport class descendant is generated.

A call reference recognizes the connection made when the first PDU arrives on an H323Transport using the Q.931 and H.225 protocols. These connections are represented by the H323Connection class, which consists of the information position for a connection between H323 endpoints. They keep a track of these active connections. New connection is created if there is no connection for the call reference number, and establishes H323 signaling negotiations.

A system usually creates an instance of a descendant of the H323Connection class in an application, instead of a class creating the instance. Here, any numerous amounts of virtual methods may be overruled. These methods are called “callback” functions of the library which allows the application to either obtain information or change performance at various phases of protocol negotiations.
The H323Negotiator classes maintain the state and functionality of each command or variable defined by the H.245 protocol. With their existence the scope of h225.h and h245.h files is reduced, which define many hundreds of classes. During every compilation, a user using H323Connection class does not need to include these classes.

Logical channels are created by remote endpoint and local application when H.245 negotiations take place. Encoded audio data is spurge open by one of these classes. H323Capability passed during protocol negotiations is used to create H323Codec by H323Channel.

4.9 Proposed Test Model

An adaptive jitter buffer is a hardware device or a software process that eliminates jitter caused by transmission delays in VoIP networks. It enables a smooth playout of audio packets in transmission. Jitter buffer is a shared data area where voice packets are collected, stored, and sent to the voice processor in evenly spaced intervals. Jitter, which is the variation in packet arrival time, occurs as the result of network congestion, timing drift, route changes, or contention for access to the Local Area Network (LAN), thus leading to loss of speech quality.

Jitter buffer is usually located at the receiving end of the voice connection, which intentionally delays the packet arrival and discards those that arrive too late where the end user can experience a clear connection with little distortion. The size of the jitter buffer depends on the condition of the network, which varies with time. If the packet loss is less than 10%, then the quality of speech is good. There are two kinds of jitter buffer—static and dynamic jitter buffer. Static jitter buffer is hardware-based, configured by the
manufacturer, whereas dynamic jitter buffer is software-based, configured by the network administrator.

A software-based jitter buffer was adapted in this research. Adaptive jitter buffer was implemented to enable smooth playout of voice packets as the packets are received. Depending on the network conditions, an efficient adaptation scheme was used to enable the buffer size to change dynamically, which could be configured to the desired level of packet drop. This increased the end-to-end latency.

Inter-arrival jitter [6] is an estimate of statistical variance of the RTP data packet inter-arrival time, measured in timestamp units and expressed as an unsigned integer. It is defined as the mean difference $D$ in packet spacing between a pair of packets at the receiver compared to the sender for the same. It is equivalent to the difference in relative transit time for two packets, which means the difference between a packet’s RTP timestamp and receiver’s clock at the time of arrival.

Inter-arrival jitter, “$J$” is calculated by

$$J_i = J_{i-1} + ((|D_{(i-1)i}| - J_{i-1}) / 16) \quad (4.1)$$

For two packets $i$ and $j$, $D$ is expressed as

$$D_{(i,j)} = (R_j - S_j) - (R_i - S_i) \quad (4.2)$$

Where $S_i$ and $S_j$ are the RTP timestamps from packets $i$ and $j$, and $R_i$ and $R_j$ are the time of arrivals in RTP timestamp units for packets $i$ and $j$.

The jitter buffer will hold packets in memory. If the value of $J$ is increased, the terminal may have to resynchronize more packets. Packets arriving late will be dropped. If $J$ is too small, then too many packets will be dropped, and if $J$ is too large, then
additional delay will be unacceptable to the user. The jitter buffer size is dynamically resized using heuristics by the terminal.

In this research, the experimental setup consisted of a network with two terminals based on OpenH323 source code, as described in Chapter 5. The terminals were H.323-deferential and could function with other H.323 software like Callgen323 that uses G.711 A-law, G.711 U-law, G.723.1, GSM audio, etc. The application OpenH323 Call Generation [17], also known as CallGen323, allows functions like generation of an adjustable number of simultaneous calls, execution of a previously recorded voice file, hanging up a call after an established time, waiting for a new call during an adjustable time, storing a trace file containing information of a call, selecting a preferred codec, and configuring the number of frames per packet. CallGen323 considered G.711 A-law as the preferred codec during the call setup. The codec could also be changed to G.723.1 or any other codec by setting their priority during the initialization of the call. The voice was extracted from an “ogm.wav” file, which was included in the source code of callgen323. The G.711 A-law codec took frames of 30 msec per packet. The G.711 had 64 kbps as the bit rate and required about 75 kbps of bandwidth including headers (IP/UDP/RTP=40bytes) for each voice call. G.723.1 encoded audio signals in 30 msec frames with look ahead of 7.5 msec. G.723.1 was chosen in addition to G.711 since it is the extensively used audio coder as a part of H.323 protocol.

In a transmission including voice/data and video to enhance the existing model of manipulating the jitter buffer, it was proposed to change the codecs from a higher bit rate to a lower bit rate to decrease the packet loss further. The existing RTP protocol needed enhancements to support changing of the codecs during the established call session. A
simulation model demonstrated the manipulation of both jitter buffer and codec, which led to the development of a new protocol.

The simulation model was built in Arena 7.01 [19]. Initially the default codec G.711 was analogous to the regular call setup of OpenH323 call generation. The results obtained were almost similar to the real-time setup with OpenH323. This model was further extended for G.723.1 codec, in addition to the existing one, by retaining the adaptive nature of the jitter buffer. The lower bit rate codec G.723.1 was chosen when the buffer reached the maximum and was prepared to discard the packets while using G.711 codec. This reduced the packet loss and improved the performance of the call.

In the simulation model, the codec was changed to a lower bit rate codec when the jitter buffer length increased. This was due to the high bit rate and frame size of the existing codec. A lower bit rate codec with less frame size was chosen so that the buffer could hold many packets to compensate for packet loss. The codec was changed only when it reached the threshold of buffer size and was ready to discard the packets arriving into it. Because it cannot be rapidly or slowly changed, this resulted in unexpected end-to-end delays and poor QoS leading to poor establishment of the call. The codec could be changed for multiple codec scenarios to reduce the packet loss in the call setup for better performance.

It was observed that the proposed model performed better compared than the existing model that was analogous to the real-time. After changing the codec from G.711 to G.723.1, it was seen that packet loss decreased by 30% compared to employing a G.711 codec. The proposed simulation model maintained the adaptive nature of the jitter buffer with minimum playout delay.
This chapter described the basic call setup of the H.323 signaling protocol between two terminals. It also gave a brief background of OpenH323 source code implemented in analyzing and testing the call setup. Augmenting the existing jitter buffer using a simulation model to change the codecs and its influence on the jitter buffer was proposed. The test bed scenario and results obtained are discussed in the next chapter.
CHAPTER 5

RESULTS

5.1 Introduction

The application layer RTP protocol supports the transmission of audio packets across the Internet using the best-effort UDP transport protocol. Every header of RTP comprises of a timestamp, sequence number, marker bit, and source ID to recognize diverse features. They are helpful through the synchronization process where the sequence number is essential to identify packet loss, the timestamp is necessary for inter-stream and intra-stream synchronization, and the marker bit specifies the start of a talkspurt.

Terminals were set up constituting OpenH323 source code [17] to inspect the performance of playout algorithms. These terminals ran on PCs with the Red Hat Linux operating system. The experimental setup and other scenarios are discussed here. The data obtained from these experiments were analyzed using OpenH323 trace files and Ethereal. The proposed test model was tested using Arena simulation software.

5.2 Ethereal Packet Analyzer Tool [18]

Ethereal is an Open Source, PC-based protocol analyzer that provides network analysis features to understand the behavior of a VoIP setup. It is a network packet analyzer that captures network packets and displays their details. It can monitor and collect TCP and UDP packets from a network and sort and display them in a comprehensible way. It is available for platforms like UNIX, LINUX, and Windows, and captures live packet data from a network interface. It is an open-source software project, released under GNU General Public License (GPL). The source code is freely available
under GPL and can be modified to suit the requirements. Ethereal was initially developed by Gerald Combs. It is now used by network professionals for troubleshooting, analysis, software, and protocol development and education.

5.3 Arena 7.01 Simulation Software [19]

Arena, from Rockwell software, offers the ease of use, flexibility, and domain expertise required for improving the performance of any aspect of the business enterprise. The Arena product family supports both in-depth analyses of particular functional areas like manufacturing, logistics, and customer service. It is used to avoid risky investments, reduce costs, optimize, and enhance sales. This simulation software is used successfully by organizations to advance the efficiency and productivity of their business. It is designed to provide successful simulation within an easy-to-use modeling environment.

Arena 7.01 is discrete time simulation model software. It provides an intuitive, flowchart-style environment for building a model of any process flow. Real-world data like process times, resource requirements, etc. can be added to Arena’s data forms to refine the model. Simulations are run to verify the model that reflects the actual system. This software provides automatic reports on common decision criteria such as resource utilization and waiting times. Arena models use basic building blocks called modules. The two types of modules are Flowchart and Data modules. Flowchart modules are placed in the model window and connected to form a chart describing the dynamic process of the model; they are edited by double-clicking to get a dialog. Data modules are edited using a spreadsheet interface and display similar kinds of modeling elements simultaneously.
This project used Flowchart modules to simulate the experimental scenario. Arena is a true Windows application and compatible with other windows software like word processors, spreadsheet, etc. It also models using different modules like the Create module, Entity Data module, Process module, Resource data module, Queue data module and Dispose Flowchart module.

The Create module creates a birth node for the arrival of entities called the inter-arrival time. The inter-arrival time between packets is based on a Normal distribution [20]. The Process module represents the queue and entity delay time. The entity delay time is represented by Weibull distribution [20]. A Queue data module represents the discipline used to operate, and a First-in First-out (FIFO) queue is selected by default. A Dispose flowchart module represents entities outside the model boundaries.

5.4 Experimental Setup

Fig. 16 shows the test model for the VoIP. The setup consisted of two terminals A and B with an OpenH323 source code on a Red Hat LINUX operating system to place the voice calls. The source and destination distance was considered two hops. The experiment was carried out for two different scenarios:

- Bandwidth kept constant and varying number of calls.
- Calls kept constant and varying bandwidth.
Calls were initiated from terminal A to terminal B for a bandwidth of a 10 Mbps link. The calls followed the basic call setup of an H.323 protocol. For a bandwidth of 10 Mbps, the results were recorded for a various number of calls placed. The number of calls used was 5, 10, 15, 20, 25, and 30. It was noted that until there were 20 calls, there was no call loss. After 20 calls, some calls were lost during transmission. This was due to the serial link of 1.5 Mbps used in setup which limits the number of calls to be transmitted. Packet loss during these sessions varied because the packets sent were not constant and the experiment was based on real time.

Fig. 17 represents a plot of average packet loss for the different number of calls placed. The packet sent varied from 400 to 2000 packets.

![Average Packet Loss vs # of Calls Placed](image)

<table>
<thead>
<tr>
<th># of Calls Placed</th>
<th>Average Packet Loss (Units, %)</th>
<th>Calls lost (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>683.8</td>
<td>0.00</td>
</tr>
<tr>
<td>10</td>
<td>845.8</td>
<td>0.00</td>
</tr>
<tr>
<td>15</td>
<td>744.6</td>
<td>0.00</td>
</tr>
<tr>
<td>20</td>
<td>903.5</td>
<td>0.00</td>
</tr>
<tr>
<td>22</td>
<td>825.0</td>
<td>0.00</td>
</tr>
<tr>
<td>24</td>
<td>393.6</td>
<td>29.17</td>
</tr>
<tr>
<td>25</td>
<td>478.3</td>
<td>20.00</td>
</tr>
<tr>
<td>30</td>
<td>558.8</td>
<td>16.67</td>
</tr>
</tbody>
</table>

Fig. 17 Plot of average packet loss in real-time system

### 5.5 Performance of Call Establishment with Change in Bandwidth

In this scenario, the total number of calls was kept constant at 60, and the bandwidth was varied from 1 Mbps to 20 Mbps. The playout buffer was still adaptive in nature and did not affect any other parameters. As the bandwidth increased, it was
observed that there were fewer number of calls dropped, and most of the calls were established. Fig. 18 represents this discussion.

Fig. 18 Plot of call establishment

5.6 Test Using Arena Simulation Tool

The tests using Arena were initially done to verify the model which was analogous to the real-time system. In OpenH323, the G.711 codec was used by default for all calls placed. Primarily, only one call was placed. The inter-arrival time and delay between the packets were analyzed using Ethereal software. In the Arena simulation, the packet sent was chosen to be the same as the real-time test, and it was proved that the simulation model was analogous to the real-time test. The adaptive jitter buffer was set to vary from 20 msec to 1000 msec in real-time to maintain its adaptive nature. From the traces of Ethereal and OpenH323, it was noted that one packet size was of 240 bytes for G.711. For 1000 msec, it took approximately 7920 bytes or 33 packets. From this, it was inferred that when the buffer is full it can take approximately a maximum of 33 packets when G.711 codec is used. Based on these criteria, a model was built in Arena, where the
packets were being played out at constant interval of 40 msec. Fig. 19 shows the basic model for G.711. In the Arena simulation, instead of placing a number of calls like in the real-time system, tests were run with replications of 5, 10, 15, 20, 25, and 30, analogous to placing simultaneous calls through Callgen323 in OpenH323. In these simulations, the packet sent was kept constant at 1500 packets for each replication. The simulation model had a Normal distribution for the inter-arrival time of packets, with 37 as the mean and a standard deviation of 2.01. A Weibull distribution was adopted for the delay between the packets, with $\alpha$ of 7.33 and $\beta$ of 4.36.

![Fig. 19 Test model with G711 codec](image_url)

![Average Packet Loss with G711 Codec](image_url)

Fig. 20 Plot of average packet loss with G711 codec
Fig. 20 shows a graph of the number of calls versus average packet loss for the above test model in Arena software. In this model, the packet loss was approximately 5% and was thus verified with the original real-time system. Since the packet sent was kept constant, the resultant packet loss was also approximately similar. The average packet loss for all the call was 81.36.

5.7 Proposed Test Model of Changing Codec with Existing Adaptive Jitter buffer

Fig. 21 shows a schematic representation of the proposed model. In this model, to reduce packet loss during transmission, the codec was changed from a higher bit rate codec G.711 to a lower bit rate codec G.723.1. During transmission of packets in the network, the buffer size was ensured and the G.711 codec was adapted by default. When the buffer was at threshold, it automatically changed to the lower bit rate codec G.723.1 to hold more number of packets in the buffer.

![Flowchart](image)

Fig. 21 Test Model for adaptive codecs

The size of G.723.1 was 96 bytes for one packet. The jitter buffer dynamically varies from 20 msec to 1000 msec, and G.723.1 held approximately 7968 bytes for 1000
msec. This is approximately equal to 83 packets maximum in the buffer. The buffer playout packet at a constant rate of 40 msec after the codec was changed to G.723.1 from G.711. The packet sent was kept constant at 1500 packets in this scenario and was run for different replications.

The simulation model had normal distribution for inter-arrival time of packets and a weibull distribution for the delay between packets. The normal distribution had a mean 37 and a standard deviation of 2.01. The weibull distribution had $\alpha$ of 7.33 and $\beta$ of 4.36.

Fig. 22 shows a graph of the average packet loss with respect to number of calls, when the codec was changed from G.711 to G723.1 during call transmission. The average packet loss for all the calls was 56.

![Average Packet Loss with adaptive Codecs](image)

**Fig. 22 Plot of average packet loss with adaptive codecs**

### 5.8 Comparison of Packet Loss with Proposed Model

Fig. 23 represents a comparison of test models with only codec G.711 and when the codec was changed from G.711 to G723.1. The proposed test model performed well by ensuring lesser packet loss with optimum playout delay.
This graph clearly indicates that the proposed test model reduced the loss of packets by 30% from the original test model by keeping the adaptive nature of the playout buffer.

This chapter described the basic scenario with an OpenH323 source code which was analyzed using the ethereal packet analyzer tool and OpenH323 trace files. Based on this real-time system, a simulation model was created using Arena software tool used to implement the test model analogous to the basic scenario. Then, the model was modified for changing the codec from the higher bit rate G.711 to the lower bit rate G.723.1, resulting in less packet loss compared to the original test model with only one codec G.711. The next chapter deals with the conclusions from the results and future work.
6.1 Conclusions

The timing between the codec frames at the sender was different from the timing between the codec frames at the receiver. This resulted in an inaccurate representation of the original sound at the receiver. The adaptive jitter buffer stored the codec frames for a brief amount of time to ensure it could playout the packet at a constant rate. The number of frames arriving or leaving changed the buffer size to decrease or increase it accordingly. The delay introduced by the playout buffer also depended on the number of codec frames that were in the buffer at a given time.

During transmission of voice, data and video, the existing model of the RTP protocol required an enhancement in jitter buffer to support the changing of codecs during the established call session. The proposed test model supported this enhancement by changing the codec from a higher bit rate codec G.711 to a lower bit rate codec G.723.1.

The codec was changed from a higher bit rate to a lower bit rate only when the buffer length reached the threshold and the incoming packets would be discarded. This was adapted to reduce packet loss by choosing a lower bit rate codec which had a smaller frame size and could hold many packets in the buffer. The jitter buffer size limit did not change when a codec was changed. It maintained its adaptive nature by checking the size of the buffer when a packet entered. When there was an indication that the buffer was at threshold, the codec was changed.
In this research, the focus was to change from one codec to another codec only. This model could be adapted to change multiple codecs during the call transmission. The order of changing the codecs needs to be considered. Based on the simulation results it is observed, that the rapid change of codecs from one to another led to transmission errors, unexpected end-to-end delay, and poor quality of service, thereby deteriorating the performance of the call established. If a codec was changed very slowly, it also affected the performance of the call established due to high end-to-end delay and heavy packet loss.

Experiments were carried out for different scenarios by keeping the bandwidth constant and varying the number of calls, and by varying the bandwidth and keeping the number of calls constant, in this project. It was observed that as the bandwidth increased, nearly all the calls were established. The simulation model proposed for changing the codec from a higher bit rate to a lower bit rate when the jitter buffer was full resulted in 30% less packet loss compared to when only one codec was employed.

6.2 Future Work

Achieving better performance of voice with different kinds of voice traffic such as ATM, frame relay, etc., could be studied to estimate the quality and quantity of resources to support high quality H.323 audio conferencing. This also could be adapted to implement videoconferencing. Analyzing the sharing of bandwidth between H.323 traffic with other voice traffics could lead to a valuable approach to understanding congestion and other factors, leading to the development of new algorithms. This could also be implemented using SIP protocol.
The jitter buffer could be manipulated by changing multiple codecs, and implementation in the real-world scenario could be developed. The order of changing the codecs while implementing needs to be considered, which could lead to the development of a new protocol. The behavior and the effect of the parameters involved could also be studied.
REFERENCES


http://www.iec.org/online/tutorials/h323/topic01.html


[17] OpenH323 Project, wwwopenh323.org


APPENDIX A
APPENDIX A

Normal Distribution

The Normal Distribution or Guassian Distribution is a model of quantitative phenomena in the natural and behavioral sciences. It is also a model for relative frequency distribution of errors. In theory, a normal distribution occurs as the limiting distribution of several continuous and discrete families of distributions. It was first introduced by Abraham de Moivre in an article in 1734, to approximate certain binomial distributions for large n. Guassian density function proposed by C. F. Guass also used this method to justify the assumption of a normal distribution of errors by 1809.

The normal random variable possesses a density function characterized by two parameters, mean $\mu$ and standard deviation $\sigma$. The distribution is given as

$$f(y) = \frac{1}{\sigma \sqrt{2\pi}} e^{-(y-\mu^2)/(2\sigma^2)} \quad (A.1)$$

The parameters $\mu$ and $\sigma^2$ are the mean and variance, respectively, of the normal random variable $y$.

Mean $\mu$ 37 and standard deviation $\sigma$ 2.01 were used in the test model for all simulations in this research.

Weibull Distribution

The weibull distribution is a model of distribution of the length of life of products measured in hours, miles, cycles, or any other metric for successful operation of a product. Weibull analysis is used to predict the life of all products by fitting a statistical distribution to life data from the sample units. This distribution was initially proposed to
represent the distribution of the breaking strength of materials, but now it is used in reliability and quality control.

The Weibull density function contains two parameters, shape parameter $\alpha$ and scale parameter $\beta$. The distribution is given as

$$f(y) = \frac{\alpha}{\beta} y^{\alpha-1} e^{-y^{\alpha}/\beta}, \text{ if } 0 \leq y \leq \infty; \alpha > 0; \beta > 0$$  \hspace{1cm} (A.2)

The scale parameter $\beta$ reflects the size of the units in which the random variable $y$ is measured. The shape parameter $\alpha$ generates a widely varying set of curves to model real-life failure time distributions by changing its value. In this research, $\alpha$ was set to 7.33 and $\beta$ to 4.36 for all simulations using Arena software.

Weibull is very flexible due to its ability to model a wide range of failure rates and shape. It also has justified many applications and is easy to use.