

# Adaptive Wireless Multimedia Services

XIAOKUN YI



**KTH Information and  
Communication Technology**

Master of Science Thesis  
Stockholm, Sweden 2006

COS/CCS 2006-12

# **Adaptive Wireless Multimedia Services**

Xiaokun Yi

9<sup>th</sup> May, 2006

Master of Science Thesis performed at  
Wireless Center, KTH  
Stockholm, Sweden

Advisor and Examiner: Prof. Gerald Q. Maguire Jr.

Department of Communication Systems (CoS)  
School of Information and Communication Technology (ICT)  
Royal Institute of Technology (KTH), Stockholm, Sweden

# Abstract

Context-awareness is a hot topic in mobile computing currently. A lot of importance is being attached to facilitating the user of various mobile computing devices to provide services that are more “user-centric”. One aspect of context-awareness is to perceive variations in available resources, and to make decisions based on the feedback to enable applications to automatically adapt to the current environment.

For Voice over IP (VoIP) software phones (softphones), variations in network performance lead to fluctuations in the quality of the communication. Therefore, by making these softphones more adaptive to the network environment will, to some extent, mask such fluctuations. Dynamic voice and video adaptation derives from the fact that different coder-decoders (CODEC) have different characteristics, even the same CODECs with a different configuration can behave quite differently, in terms of bandwidth consumption, packet size, etc.

Minisip is a VoIP client application which was implemented on and targeted for a Linux platform. One of my tasks was to port Minisip to Microsoft’s Windows Mobile operating system, running on an HP IPAQ Pocket PC H5550. Such handheld computer enables the user to communication while they are moving about, thus increasing the probability that the characteristics of the network connection will change. Building upon this port, the next task was to add dynamic voice and video CODEC adaptation. Dynamic voice and video CODEC adaptation on Minisip poses several challenges, for example, in what way can the network performance be determined and what adaptation strategy can achieve high call quality while making efficient utilization of available network resources.

In order to make the proper design choices, several estimation models will be discussed, these are used to determine an efficient, un-intrusive, and light weight means of dynamic CODEC selection within Minisip. This thesis only implemented audio CODEC adaptation of Minisip, and the evaluation of the resulting prototype shows that such dynamic adaptation is **both** feasible and practical; further more, video CODEC adaptation would be a more significant extension to this work in the future.

# Sammanfattning

Context-awareness är ett hett i den nuvarande mobila datavärlden. Det finns ett stort värde i att facilitera användare av olika mobila dator anordningar för att kunna förse branschen med användarvänligare tjänster. En aspekt på Context-awareness är att uppmärksamma variationen i de tillgängliga medel som finns tillhanda, och att ta beslut som är baserade på feedback för att applikationen automatiskt ska anpassa sig till den nuvarande miljön.

Variationer i nätverksprestanda påverkar kvaliteten på Voice over IP (VoIP), som är en typ utav softwaretelefon, i hög grad. Dessa kvalitets svängningar kan stabiliseras och döljas i högre grad om softwaretelefonen anpassas till nätverksmiljön. Dynamisk voice och video adaptation härleds från faktum att olika coder-decoders (CODEC) har olika karaktärer, även samma CODEC med en annan konfiguration kan bete sig olikt sig själv om vi talar om bandbredds förbrukning och packet storlekar, etc.

Minisip är en VoIP klient som är framtagen för Linux plattformen. En av mina huvuduppgifter var att port Minisip till Microsoft's Windows Mobila operativsystem genom att köra en HP IPAQ Pocket PC H5550. En sådan bärbar dator möjliggör för användaren att kommunicera fastän denne rör på sig, fastän risken finns för att nätverks kontakten ändras. Baserat på denna port, blev min nästa uppgift att anpassa denna CODEC till dynamiskt ljud och bild. Att anpassa denna CODEC till dynamiskt ljud och bild på Minisip medför många utmaningar t.ex. hur nätverks prestandan kan bestämmas och vilken anpassningsstrategi som kan bidra till högkvalitativa samtal samtidigt som nätverks tillgångarna nyttjas på ett effektivt sätt.

Denna tes kan endast genomföras på ljud CODEC anpassning av Minisip, och utvärderingen utav prototypen resulterade i att sådan dynamisk anpassning är både genomförbar och praktisk, en video CODEC anpassning skulle bli ett perfekt uppföljningsprojekt till denna studie.

# Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
<b>1.1 Overview of the problem area .....</b>	<b>1</b>
<b>1.2 Problem Specifications .....</b>	<b>2</b>
1.2.1 Network Performance.....	2
<b>1.3 Proposed Solution .....</b>	<b>3</b>
1.3.1 Network Performance Estimation .....	3
1.3.2 Dynamic CODEC Switching .....	4
<b>2. Related Work .....</b>	<b>6</b>
<b>2.1 Background .....</b>	<b>6</b>
2.1.1 Voice over Internet Protocol .....	6
2.1.2 Voice over Wireless LAN .....	6
2.1.3 Voice and Video Coder-Decoders .....	6
2.1.4 Session Initiation Protocol (SIP) .....	7
2.1.5 Session Description Protocol .....	8
2.1.6 Real-time Transport Protocol.....	9
2.1.7 Real-time Transport Control Protocol.....	9
2.1.8 Minisip.....	10
<b>2.2 Related Research and Work .....</b>	<b>11</b>
2.2.1 Multicodec MINISIP.....	11
2.2.2 Techniques for Estimation of End-to-end Available Bandwidth .....	12
2.2.3 Measurements of End-to-end Packet Delay and Packet Loss.....	14
2.2.4 Ethereal .....	14
2.2.5 NIST Net .....	14
<b>3. Design and Implementation.....</b>	<b>15</b>
<b>3.1 Design Issues .....</b>	<b>15</b>
3.1.1 Development Environment.....	15
3.1.2 Code Structure of Minisip .....	15
3.1.3 Media System of Minisip .....	16
3.1.4 Multi-Session Support of Minisip .....	16
3.1.5 Network Performance Estimation Limitations.....	17
<b>3.2 Implementation .....</b>	<b>18</b>
3.2.1 QoS Feedback using RTCP .....	18
3.2.2 RTCP Feedback Module.....	20
3.2.3 Multiple CODECs .....	23
3.2.4 Audio Adaptation .....	24

<b>4. Test and Evaluation</b> .....	<b>27</b>
<b>4.1 CODEC Adaptation Test</b> .....	<b>27</b>
<b>4.2 Competing Traffic (TCP) Test</b> .....	<b>32</b>
<b>5. Conclusions and Future Work</b> .....	<b>33</b>
<b>5.1 Conclusions</b> .....	<b>33</b>
<b>5.2 Future Work</b> .....	<b>33</b>
<b>References</b> .....	<b>35</b>
<b>Appendix</b> .....	<b>38</b>
<b>A. UML Description of Related Classes</b> .....	<b>38</b>
<b>B. Unix Script of NIST Net Configuration</b> .....	<b>44</b>

## Table of Figures

Figure 1: Voice and video adaptation based upon feedback .....	3
Figure 2: State Machine Controlling CODEC .....	4
Figure 3: SIP Session Setup example .....	7
Figure 4: RTP Header .....	9
Figure 5: RTCP header .....	10
Figure 6: Minisip's CODECs.....	11
Figure 7: Packet Pair Measurement on a path .....	13
Figure 8: Code Structure of Minisip .....	15
Figure 9: Audio Media System of Minisip .....	16
Figure 10: Example of Multi-Session Call .....	17
Figure 11: RTCP Sender Report Structure.....	18
Figure 12: Class Diagram of RTCP .....	20
Figure 13: Audio Media System with RTCP .....	20
Figure 14: Flowchart of RTP Receiver with RTCP Sender .....	21
Figure 15: Flowchart of RTCP Receiver .....	22
Figure 16: SIP/SDP INVITE message from X-Lite to Minisip.....	23
Figure 17: SIP/SDP answer from Minisip to X-Lite .....	24
Figure 18: G.711 CODEC, 1 x 20 ms sample per packet -95.200 kbps over Ethernet	25
Figure 19: State Transition of CODEC Switching .....	26
Figure 20: Experimental Setup .....	27
Figure 21: Bandwidth Utilization with CODEC switching (configuration N1).....	28
Figure 22: Bandwidth Utilization with CODEC switching (configuration N2).....	29
Figure 23: Bandwidth Utilization with CODEC switching (N3, N4, N5, and N6).....	30
Figure 24: Media Flow with CODEC Switching .....	31

## List of Tables

Table 1: Basic Features of SIP .....	8
Table 2: Available CODEC states in Minisip .....	25
Table 3: NIST Net Configurations (Path A to B).....	28
Table 4: CODEC Risk Threshold .....	28
Table 5: Fraction loss and CODEC switching (configuration N1).....	29
Table 6: Fraction loss and CODEC switching (configurations N3, N4, N5, and N6) .	30
Table 7: Packet Loss of G.711 and GSM encoded data with effect of TCP .....	32



# 1. Introduction

## 1.1 Overview of the problem area

With the increasing integration of computing and wireless communications, the number of the mobile device users has expanded rapidly, since the late 1990s. Nowadays, a mobile device can range from a laptop computer to a web-enabled phone. Additionally there are considerable differences in the computing power and user interface capabilities of these devices. Besides the computing power, network performance is another important issue which determines the usability of such mobile devices and the software running on them.

For the current mobile multimedia applications and services, the main objective is to facilitate user mobility users, to enhance this mobility, and to provide high quality of service (QoS) to the best extent possible. However, QoS control and management at the client side are non-trivial due to the basic nature of mobile devices using wireless networks—with users increasingly expecting communications anywhere and at anytime. One way to realize QoS control in mobile multimedia applications is to make the device and applications aware of their current context and make adjustments accordingly. Context-awareness as a concept aims to adapt behavior according to the “environment”.

In recent years, context-awareness is a frequent topic of the mobile computing research community. Several definitions [1] of it have been proposed. The main problem that context-awareness is trying to solve is to obtain and filter useful external or internal information, thus allowing the system to automatically adapt to the environment –therefore reducing the need for user interactions while delivering better service(s). Particularly, in a network environment, the network’s resources should be considered a source of contextual information.

Compared with desktop computers connected to the fixed network, mobile devices face a variable environment due to their mobility, this leads to a higher variation in the availability and stability of network resources. Variations, such as changes in network interfaces, access and authentication delays, and lack of coverage or low link-speed are especially troublesome for real time communications. To deal with such situations, an adaptive model could be applied in mobile real-time applications (such as conversational or streaming voice and video), in order to hide or smoothen the negative effects imposed by the changing network environment.

## 1.2 Problem Specifications

Network performance and choice of Coder-Decoder (CODEC) determine the voice and video quality experienced between two or more end users of a softphone (i.e. a software voice over IP (VOIP) application). To make such applications more robust to changes of the network environment changes and to provide relatively stable communication quality. In this thesis, I have focused on adaptation of the softphone's real-time functionality, specifically the choice of voice and video CODECs. A method for dynamic voice and video adaptation is presented and evaluated in this thesis.

### 1.2.1 Network Performance

In a mobile environment, there are four main factors about the network that constraint the performance of a real-time application.

- **Link usability**

The link usability usually involves two issues, one is the presence of the link, and the other is the permission to use this link. The former is related to the relative position or distance between a mobile device and wireless interface, while the latter mainly involves issues, such as authorization, authentication and accounting (link availability to mobile users was examines in [36].).

- **Available end-to-end bandwidth**

The available end-to-end bandwidth is usually restricted by the network interface, competing traffic, link errors (and link layer transmissions), and possibly purposely restricted to be fair to other traffic (for example, using TCP Friendly Rate Control [41]) particularly for the bottleneck link between the end points. These processes are the underlying causes of packet delay and packet loss.

- **Packet delay**

Packet delay causes voice quality degradation, which in the limit leads to a breakdown of interactive conversation. ITU standard G.114 [33] specified that less than 150ms of one-way, end-to-end delay ensures user satisfaction for telephony applications. Delay variation (i.e., jitter) can be solved by use of a de-jitter buffer. The size of this buffer can be adapted based upon estimates of the jitter.

- **Packet loss**

Packet loss usually happens on the bottleneck link of a path due to heavy competing traffic, but may also occur due to link errors or lack of connectivity during handoffs. According to ITU standard G.114 [33], packet loss causes voice clipping and skips. Some CODECs have their own methods to do packet loss concealment (PLC), a technology to mask the effects of lost or discarded voice packets.

### 1.2.2 Dynamic Voice and Video Adaptation

To realize dynamic voice and video adaptation, two interrelated issues should be addressed. The first step is acquiring sufficient accurate knowledge about the network conditions; the second step is to apply an effective strategy to adapt the media based on these network conditions.

## 1.3 Proposed Solution

### 1.3.1 Network Performance Estimation

The network performance can be measured based on packet probing techniques (see sections 2.2.2 and 2.2.3). For real-time media applications, the Real-time Transport Control Protocol (RTCP) provides feedback about the quality of real-time media delivery to each end user, thus allowing us to calculate the network's performance. A suggested means to gain additional information is to use back-to-back RTCP packets to measure the available end-to-end bandwidth on a path; however, this method can only be applied when the traffic load is relatively light (i.e., low packet loss). The reason for this limitation is that once the link is congested with competing traffic, the possibility of RTCP packets being dropped or discarded by the intermediate nodes on the transmission path increases, thus, the back-to-back packet pairs might be lost or face different loads and the desired estimation correctness and accuracy can not be achieved. However, at these higher traffic loads, the RTP packets carrying the media data themselves can act as the probes, thus providing feedback on the packet loss rate. In order to utilize the proper type of the probes, a threshold should be identified to distinguish between light traffic and heavy traffic (Threshold 1 in Figure 1). Of course these two kinds of probes could work together to guarantee a more accurate and effective estimation, but this may not be suitable for an application on mobile devices due to their limited battery power (Power consumption is not explicitly considered in this thesis, but has been addressed in other theses, such as [36, 37].).

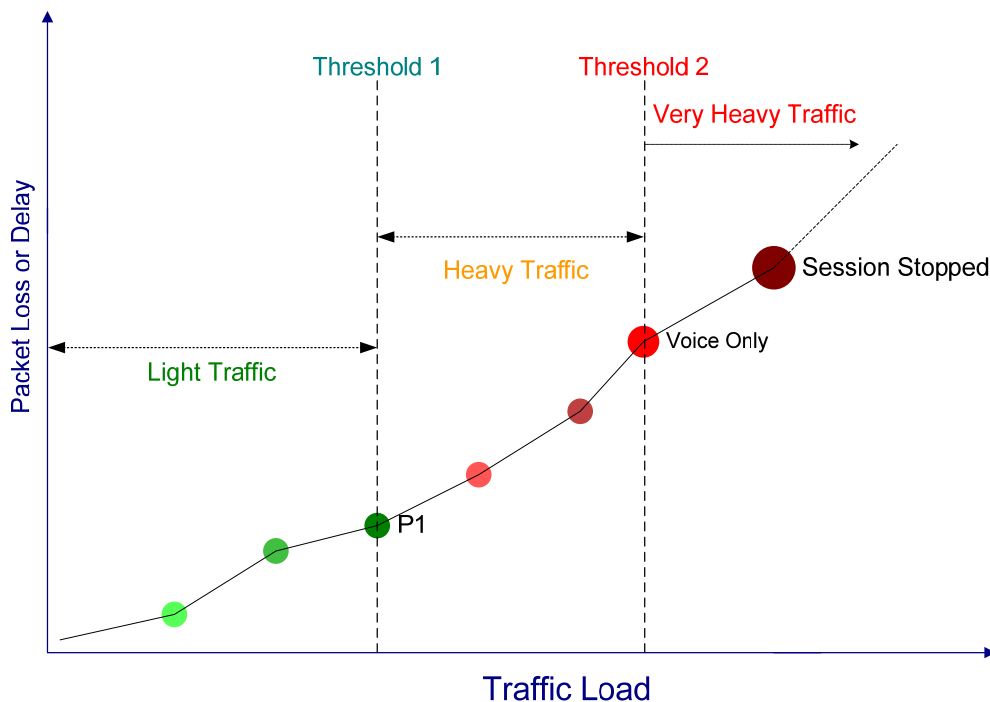


Figure 1: Voice and video adaptation based upon feedback (each dot represents a different combination of CODEC parameters)

### 1.3.2 Dynamic CODEC Switching

Based upon network feedback, applications should adaptively switch between CODECs or adjust the parameters of the current CODEC. The criteria for CODEC switching and parameter adjustment are based on the desirable overall conversation or video quality and the available network resources. For the voice CODECs integrated in Minisip (see section 2.2.1), the sampling rate and choice of CODEC are the primarily parameters determining the bandwidth required. In some circumstances, instead of changing the CODEC, simply adjusting the sampling rate might lead to a significant improvement of the perceived speech quality or reduce the use of the network resources. As the video CODEC's traffic consumes more bandwidth than the voice traffic, changing the frame's resolution or frame rate should be the first parameter to adjust when the network performance is poor. In Figure 1, each colored point on the curve represents a possible CODEC change or parameter adjustment. There is a critical value "threshold 2" that should be identified, at this particular point, the available network resources can not support both voice and video data transmission, thus, to guarantee suitable quality of voice transmission, the video should be turned off. If despite this, the situation continues to get worse, then the whole media session should be terminated when "the loss rate is persistently unacceptably high relative to the current sending rate and the best-effort application is unable to lower its sending rate" [25].

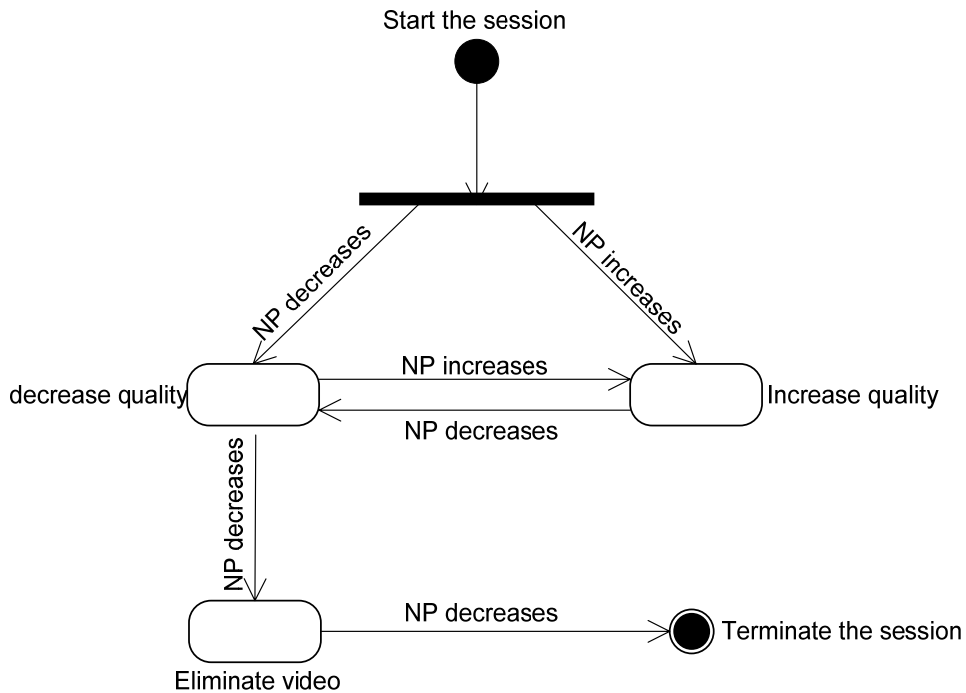


Figure 2: State Machine Controlling CODEC (NP = Network Performance)

From a general point of view, an adaptation model could be modeled as a finite state machine. In Figure 2, each state represents a tuple of {voice CODEC, video CODEC, each with their respective parameters}; the transition between states is based on the network performance (NP).

Some of the proposed solutions stated in this section were implemented and evaluated. Chapter 2 will give a general description of previous related work. The implementation and evaluation will be presented in Chapters 3 and 4.

## 2. Related Work

### 2.1 Background

The traditional telephony service has been carried over the dedicated Public Switch Telephone Networks and has not changed substantially since it was first deployed in the 1800s (i.e., it has remained a circuit switched system with dedicated resources for each call). In the late 1990s, a new technology—voice over IP (VoIP) started to emerge, based on a totally different mode. Today, traditional telephony providers face stiff competition against this new intruder, which appear to be the prototype of tomorrow's telephony. However, today this new technology still has a lot of challenges to face, providing a suitable quality of service is one of them. For the consumer, VoIP provides new services which were difficult or very expensive for the user in traditional telephony systems.

#### 2.1.1 Voice over Internet Protocol

Voice over Internet Protocol (VoIP) [4] is a term used in IP telephony for a set of facilities for managing the delivery of voice information using the Internet Protocol. An application sends voice data transmitted in digital form and in discrete packets rather than via the traditional circuit-switched protocols of the public switched telephone network (PSTN). One of the most important features of VoIP is that *“VoIP is an end-to-end architecture which exploits processing in the end points, while for the traditional PSTN; the processing is inside the networks.”*[5], where *“convergence of services”* [5, 6] can be achieved. While for many, the most attractive advantage of VoIP services over traditional PSTN is that toll charges can be avoided, i.e., machine-to-machine calls are completely free, increasingly VoIP service providers are introducing new services, such as enabling calling fixed or mobile telephones at a very low rate (especially for overseas calls), voice mail, calls from fixed or mobile phones to VoIP users, mobile presence information, etc.

#### 2.1.2 Voice over Wireless LAN

VoIP over Wireless LAN (VoWLAN) is a technology which enables IP voice to be sent over a wireless LAN (such as IEEE 802.11). VoWLAN systems usually work in two different ways. One way is to route calls from the VoIP phone to a WLAN access point and then to a VoIP gateway (where they are connected to the PSTN), this can be used to deliver VoIP over traditional telephone networks. Another way is based on softphones, which route calls directly over the internet in an end-to-end fashion. Softphones can be installed on PDAs, mobile phones, or laptops and calls can be established anywhere WLAN connectivity to the internet is available.

#### 2.1.3 Voice and Video Coder-Decoders

Voice and Video Coder-Decoders (CODECs) are used to convert an analog voice or video signal to a digitally encoded version and vice versa. They vary in quality, bandwidth required, computational requirements, etc. Different CODECs have different properties and parameters, as they were designed to

provide optimal voice and video quality under different conditions. We will introduce the specific CODECs used in section 2.2.1

### 2.1.4 Session Initiation Protocol (SIP)

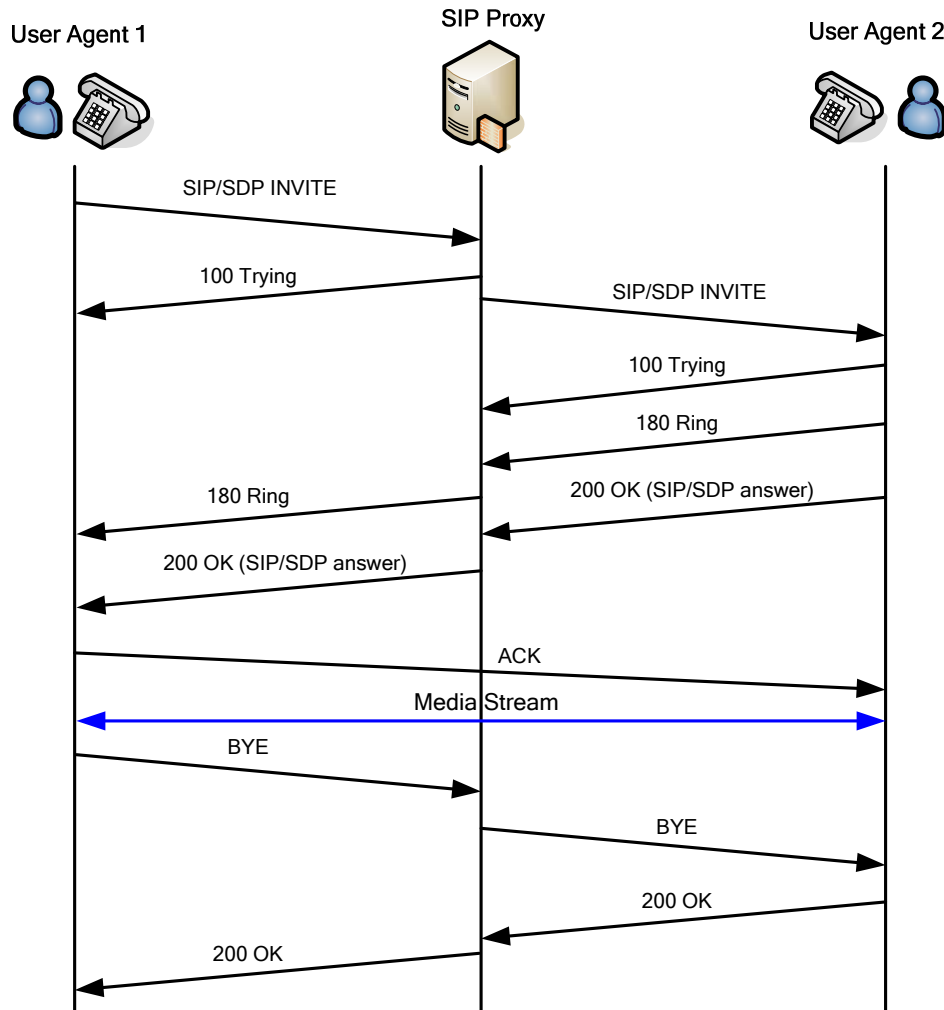


Figure 3: SIP Session Setup example

The Session Initiation Protocol (SIP) [5, 12] was developed by the IETF MMUSIC Working Group. It is a proposed standard for initiating, modifying, and terminating an interactive session that involves multimedia elements such as video, voice, instant messaging, etc. SIP is an application-layer protocol whose messages are textually encoded. It can utilize User Datagram Protocol (UDP), Transmission Control Protocol (TCP), Stream Control Transmission Protocol (SCTP), and so on for its underlying transport—this transport is separate from the media transport and make take a completely different path. A goal of SIP was to provide a superset of call functions and features present in the PSTN. Using SIP, a user

can invite participants to existing sessions, such as (multicast) conferences; additionally media can be added to or removed from an existing session. SIP transparently supports name mapping and redirection services, which supports personal mobility, thus users can maintain a single externally visible identifier regardless of their network location. An example of SIP session setup is shown in Figure 3.

Five types of services that SIP supports are specified in RFC 3261 [12]:

User Location	To determine the location of the end systems to communicate with
User Availability	To determine if the callee is willing to engage in the communication
User Capabilities	To determine the media and media parameters to be used
Session Setup	To establish session parameters at both called and calling party and start calling
Session Management	To transfer and terminate sessions, as well as, to modify the session parameters and invoke services

SIP system consists of two components [22]:

User Agents	an end system acting on behalf of a user, which can be further divided into User Agent Client (UAC) and User Agent Server (UAS) due to their different roles in the system.
Network Servers	registration server, proxy server, or redirect server.

Table 1 below shows the basic features of SIP [11].

Table 1: Basic Features of SIP

<b>Encoding</b>	Textual
<b>Use in 3GPP</b>	Yes
<b>Complexity</b>	Moderate: HTTP-like protocol
<b>Extensibility</b>	Open to new extensions
<b>Architecture</b>	Modular
<b>Addressing</b>	a URL
<b>Transport Protocol</b>	UDP (for most implementations), TCP, and SCTP

### 2.1.5 Session Description Protocol

The Session Description Protocol (SDP) [23] describes the multimedia sessions giving information about session announcement, session invitation, and other types of multimedia session information to the potential participants.

However, SDP only defines the format of the session descriptions; it doesn't specify the protocol for the data transport, although it states what protocols might be used for the media session. Therefore, a number of different transport protocols can be used for carrying the session descriptions, such as Session



Announcement Protocol (SAP) [23], SIP, and Real Time Steaming Protocol (RTSP) [37].

Usually SDP serves in one of two roles; one is to announce the existence of a session and relevant information by multicast, the other is to help participants to join an existing session (probably by unicast).

To insure that sufficient information is conveyed, an SDP message usually includes:

- Session name and purpose
- Time the session is active
- Media comprising the session
- Information needed to receive the media, such as address, etc

### 2.1.6 Real-time Transport Protocol

The Real-time Transport Protocol (RTP) [24] provides end-to-end delivery services for data with real-time characteristics and it is frequently used in streaming media systems (e.g., in conjunction with RTSP) as well as audio/video conferencing and push-to-talk (p2t) [32] systems (in conjunction with H.323 or SIP), making it the technical foundation of the voice over IP industry (It should be noted that there are *proprietary* VoIP systems which do not use RTP.).

Applications typically run RTP on top of UDP to make use of UDP's multiplexing and checksum service. However, RTP may be used with other suitable underlying network or transport protocols. RTP supports data transfer to multiple destinations using multicast distribution if provided by the underlying network.

An RTP packet usually has a fixed header (Figure 4). Because RTP does not provide any mechanism to ensure timely delivery or other QoS guarantees, the sequence number and timestamp fields in the header allow the receiver of the RTP packets to reconstruct the sender's packet sequence and perform synchronization and de-jittering. Details of the RTP packet format can be found in [24].

2	3	4	8	9	16bit
V	P	X	CSRC count	M	Payload Type
Sequence number				Timestamp	
SSRC				CSRC (variable 0 - 15 items, 2 octets each)	

Figure 4: RTP Header

### 2.1.7 Real-time Transport Control Protocol

The Real-time Transport Control Protocol (RTCP) is a sister protocol of RTP and also defined in [24]. It partners with RTP's packaging and delivery of multimedia data, but does not transport any media data itself. Rather, it is used to (somewhat periodically) transmit control packets to participants in a multimedia

session. The primary function of RTCP is to provide feedback on the quality of service being provided by the associated RTP stream.

RTCP packets carry statistics concerning a media stream, such as number of bytes sent, packets sent, lost packets, jitter, feedback, and round trip delay. An application may use this information to increase the quality of service perhaps by limiting the flow, or perhaps using a low compression CODEC instead of a high compression CODEC. RTCP is used for QoS reporting.

The fixed part of the RTCP header is shown in Figure 5. Details of the RTCP packet format are defined in [24].

<b>2</b>	<b>3</b>	<b>8</b>	<b>16bits</b>
V	P	RC	PT
Length			

Figure 5: RTCP header

Fields in RTCP header are described below.

- Version (V) identifies the version of RTP, which is the same in RTCP packets as in RTP packets
- Padding (P) indicates whether additional padding octets are contained at the end of this RTCP packet
- Reception Report Count (RC) the number of the reception blocks contained in this RTCP packet
- Packet Type (PT) Indicates of the type of this RTCP packet
- Length the length of this RTCP packet in 32-bit words minus one, including the header and any padding

### 2.1.8 Minisip

Minisip [7] is a SIP user agent based on an open source project primarily carried out by doctoral and masters students at KTH. It was initially targeted to the Linux platform and provides an alternative to other softphones, such as Skype [29], Vbuzzer [30], etc.

Minisip has some attractive features; it uses Transport Layer Security (TLS) [31] for the secure signaling, as well as MIKEY [38] together with Secure Real-time Transport Protocol (SRTP) [39] for key management and media encryption respectively. Video support, p2t [32], support for multiple sessions at the same time [42], and full mesh conferencing [44] are additional features that make Minisip an interesting base for this project.

Currently, there are 4 libraries (libmutil, libmnetutil, libmikey, and libmsip) and an application layer module (minisip) comprising Minisip. Research and development of Minisip is still on-going, a key aspect of this thesis work is to make Minisip more versatile and to provide better QoS.

## 2.2 Related Research and Work

Research related to network performance estimation started over ten years ago. This section presents some related work which could be relevant to this thesis.

### 2.2.1 Multicodec MINISIP

“Multicodec MINISIP” [10] was a project suggested by Minisip’s lead developer, doctoral student Erik Eliasson and undertaken by several KTH masters students. The goal of this project was to enable and integrate a number of voice CODECs into Minisip, and to verify correct session set-up with multiple CODECs.

In this project, two additional voice CODECs were integrated into Linux version of Minisip: internet Low Bitrate CODEC (iLBC) and Speex (both are described below), plus the default codec ITU G.711. Figure 6 illustrates the voice and video CODECs available in the Linux version of Minisip.

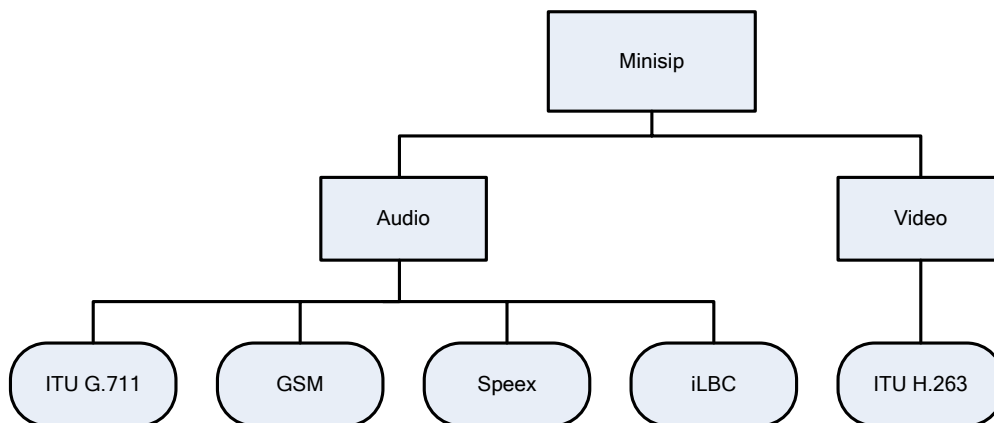


Figure 6:Minisip’s CODECs

The basic features of these audio CODECs are:

#### **G.711** [13]:

- High bitrate: 64kbps
- Two different versions:  $\mu$ -Law (indigenous to the T1 standard used in North America and Japan), A-Law (indigenous to the E1 standard used in the rest of the world)
- Sampling rate: 8000 samples per second
- Theoretical maximum voice bandwidth: 4000Hz

**iLBC [13]:**

- Bitrate 13.33 kbps (399 bits, packetized in 50 bytes) for a frame size of 30 ms and 15.2 kbps (303 bits, packetized in 38 bytes) for a frame size of 20 ms
- Basic quality is higher than G.729A (another audio data compression algorithm) [13], robust to packet loss
- Similar computational complexity to G.729A
- Royalty Free

**Speex [14]:**

- Narrowband (8 kHz), wideband (16 kHz), and ultra-wideband (32 kHz) compression in the same bit-stream
- Intensity stereo encoding
- Packet loss concealment
- Variable bitrate operation (VBR)
- Voice Activity Detection (VAD)—detecting when the user is not speaking
- Discontinuous Transmission (DTX)—avoids sending packets when there is no voice activity

**GSM [35]:**

- Full-Rate(FR) GSM CODEC uses Regular Pulse Excitation Long-Term Prediction (RPE-LTP) compression, 13kbps
- Enhanced-Full-Rate (EFR) GSM CODEC uses Algebraic Code Excited Linear Prediction (ACELP) compression, 12.2kbps
- Half-Rate(HR) GSM CODEC uses Code Excited Linear Prediction - Vector Sum Excited Linear Prediction (CELP-VSELP) compression, 5.6kbps

**2.2.2 Techniques for Estimation of End-to-end Available Bandwidth**

Currently, measurement of network bandwidth is crucial for many Internet applications and protocols, especially those involving the transfer of large files and those involving the delivery of content with real-time QoS constraints, such as conversational and streaming media. To measure the end-to-end bandwidth in a simple way, special hardware or software deployed in the core of the network must **not** be required; therefore, operations should only need to be made at the end points. Curtis and McGregor [2] give a survey of some currently used techniques to determine the end-to-end network bandwidth. We will examine some of these in more detail below.

**2.2.2.1 Single Packet Techniques**

Single packet techniques are usually used to estimate an individual link's bandwidth as opposed to end-to-end properties. These techniques are based on the observation of differences in transmission time between packets sent on a faster link and a slower link.

In a computer network, several factors should be taken into account, such as link bandwidth and latency.

For the single packet techniques, latency, which varies on each link, must be calculated. Latency depends on the packet's size and the bandwidth of that link, but here we are only concerned with the time the packets travel from one end of a path to the other. The transmission time can be calculated based on the packet size, link bandwidth, plus a fixed latency which is unique for a particular link. Accordingly, if the transmission time, packet size, and (often fixed) latency are known, the bandwidth can be determined. Curtis and McGregor [2] described a method that takes advantage of the IP *time-to-live* (TTL) field. When the TTL of a packet becomes zero, the router will send back an ICMP packet to the packet's source, by increasing the TTL value, properties of each successive link along a path can be measured.

In practice, there are several problems when conducting these measurements in the real world [2]. For example, if the ICMP packet is sent back on a different path, due to asymmetric routing, we will have an incorrect estimate. Even if the ICMP packet is sent back via exactly the same path, due to differences in the traffic congestion on the forward and reverse paths, we still can not distinguish the difference in delays (i.e., we see only the sum of the forward and reverse path delay). Therefore, exploiting the *shorted observed round trip time* (SORTT) [15] is the key to solve these problems. Today, there are a number of implementations of single packet techniques, such as Jacobson's *pathchar* [16].

#### 2.2.2.2 Packet Pair Techniques

Unlike single packet techniques, packet pair techniques are usually used to measure the bandwidth of a path, instead of the bandwidth of a link [2, 18]. A packet pair is two identically sized packets sent immediately one after the other, i.e. back-to-back. The idea behind this method is that the bandwidth can be derived from the inter-arrival time of these two packets at the destination.

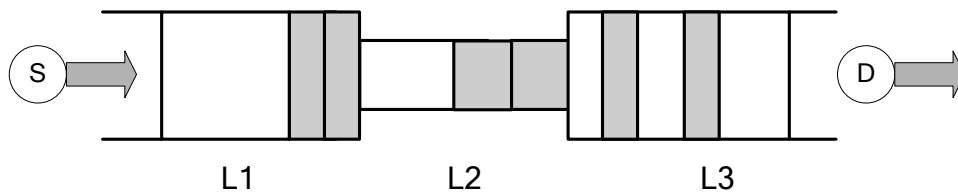


Figure 7: Packet Pair Measurement on a path

Figure 7 illustrates the packet pair measurement on a path. For simplicity, cross-traffic is ignored. Here, we can see that when the packet pair travels through a link (L2) with limited bandwidth, spacing between these two packets is observed on L3. Link bandwidth can be estimated based on the packet size and spacing between them at the destination.

When cross traffic is added, additional issues should be taken into account. Cross traffics affect the packet pair model in two ways—*spacing compression* and *spacing expansion* [2]. Spacing compression is caused by the cross traffic delaying the first packet of the pair, which rarely happens due to the queuing mechanism of the routers. While spacing expansion is caused by competing traffic filling in between the

packet pair, splitting them apart. Therefore, filtering out the compressed and expanded measurements becomes the key to extracting information. Statistical methods were introduced by Lai, *et al.* [18] and Carter, *et al.* [4] with the assumption that compressed and expanded delays are random variables.

### **2.2.3 Measurements of End-to-end Packet Delay and Packet Loss**

Besides the available bandwidth, some other aspects relevant to network performance are packet delay and packet loss rate. These two factors directly affect the way a real-time multimedia application works. Considering the CODECs that are currently available on softphones, they will only work acceptably up to some bounded packet delay. Additionally these CODECs can endure a certain amount of packet loss, which depends on a particular CODEC.

In packet-switched networks, each packet is generated at the source and routed to the destination via a sequence of intermediate nodes (routers, switches). As described by Bolot [20], the end-to-end delay is the sum of the delays accumulated at each hop on the way to the destination, and such delay consists of two parts, a fixed part including the transmission delay at a node and the propagation delay between the neighboring nodes, and a variable part including queuing and processing delay at each intermediate a node; whereas, packet loss is usually caused by buffer overflow in one of the intermediate nodes.

D. Sanghi, *et al.*[21], have experimentally measured the end-to-end path characteristics and found that the variability of packet round-trip-time (RTT) was not based on cross traffic alone, as a lot of duplications and even some reordering of packets happened under light traffic loads. J.C. Bolot [19] has analyzed the end-to-end packet delay and loss using the measurement of the round trip delays of small UDP packets. By analyzing these packet delays, he obtained a relatively accurate value of the bottleneck bandwidth on a transatlantic link. To increase the accuracy of the packet loss measurement, J. Sommers *et al.* [20] have developed and implemented a prototype tool—BADABING which could achieve a more accurate measurement than other existing methods and tools.

### **2.2.4 Ethereal**

Ethereal [26] is a network protocol analyzer. It can capture packets from live network traffic or read data from a file containing previously captured traffic. It can be used to troubleshoot the network, detect network intrusions, locate network bottlenecks, etc. There are several powerful features in Ethereal, such as support for browsing the captured packet details via a graphical user interface (GUI), a filter for data display refinement, and viewing the ASCII content of a TCP connection.

### **2.2.5 NIST Net**

NIST Net [27, 28] is a network emulator package for used on Linux. It allows a Linux PC to work as a variable performance link, enabling us to emulate performance dynamics of an IP network. By adjusting the packet delay, packet loss, packet duplication, and bandwidth constraints, NIST Net can emulate the end-to-end network performance, thus enabling reproducible laboratory settings under which network applications can be experimented with.

## 3. Design and Implementation

### 3.1 Design Issues

This section presents some of the design issues of dynamic CODEC-switching in the current implementation of Minisip. Concerns described in this section were taken into account in our implementation which is also described.

#### 3.1.1 Development Environment

Minisip was originally developed on Linux operating systems. Now, thanks to the work of a number of contributors<sup>1</sup>, Minisip is available on several other platforms, such as Embedded Linux, Windows XP/2000, and Pocket PC. However, the only stable version of Minisip is on Linux, others are still under development. Due to my knowledge of Microsoft Windows and some of the advanced features of Microsoft's Visual Studio 2005, I chose Windows XP as the operating system for my development, and Microsoft Visual Studio 2005 as the integrated development environment (IDE).

#### 3.1.2 Code Structure of Minisip

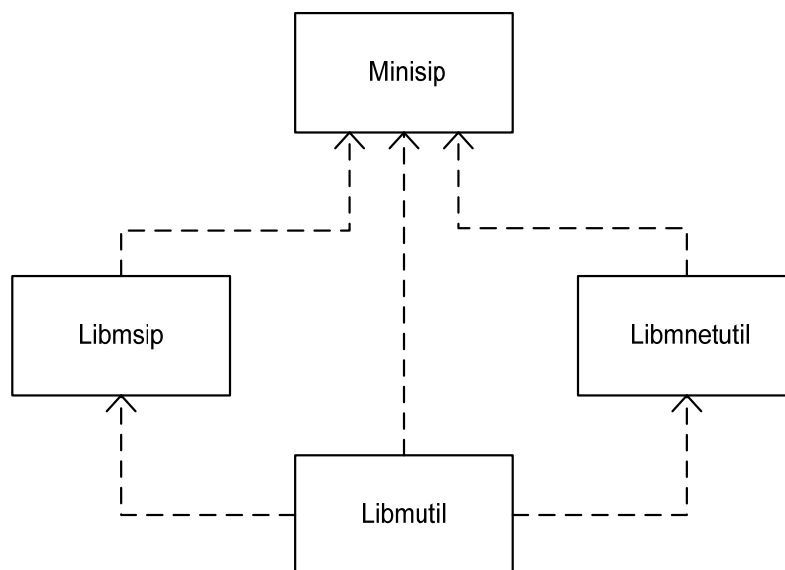


Figure 8: Code Structure of Minisip

The current code structure of Minisip is divided into 4 modules—three libraries and one application module (Figure 8). The libraries provide functions via their APIs to the application module, for example,

<sup>1</sup> Erik Eliasson (minisip, libmsip, ...), Jon-Olov Vatn (minisip/security), Israel Abad (SRTP), Johan Bilien (minisip, SRTP, libmikey, ...), Florian Mauer (p2t), Joachim Orrblad (MIKEY, IPSEC), Ignacio Sanchez Pardo (spatial audio), Cesc Santasusana, Xiaokun Yi and Ali Nesh-Nash (Windows XP port, low-level Audio I/O [43])

Libmnetutil provides network APIs, and Libmsip provides SIP related APIs. To minimize changes of the original code, most of the modifications should be made in the application module, these changes should be modularized, and are designed to be easily extended.

### 3.1.3 Media System of Minisip

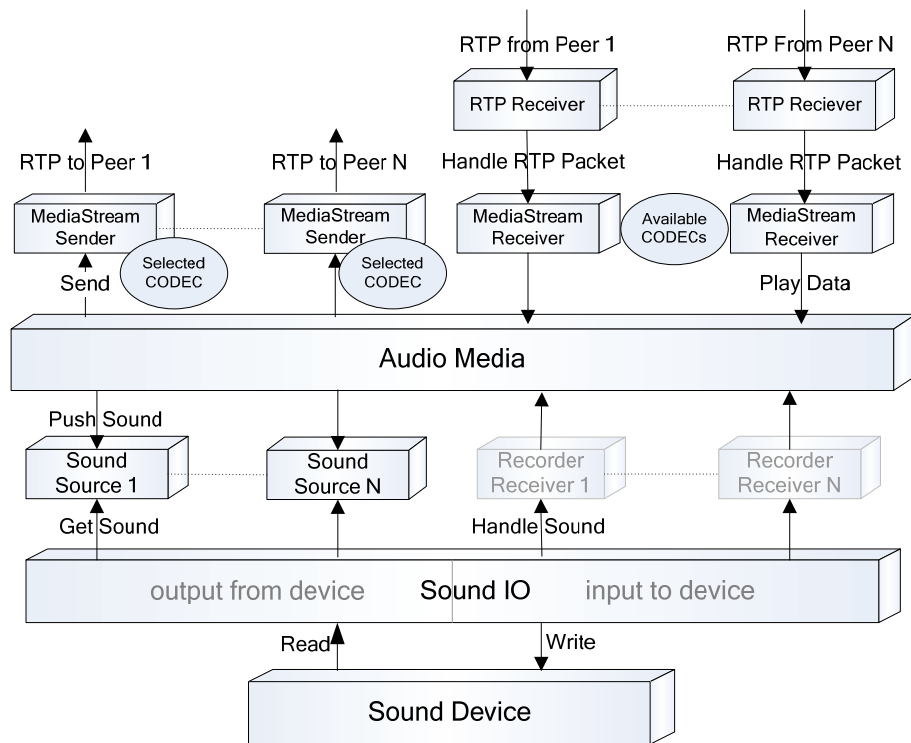


Figure 9: Audio Media System of Minisip

Minisip's media system was implemented in several layers (an audio only illustration of this is shown in Figure 9). The lowest layer deals with the input and output to/from the sound and video devices, and the highest layer deals with receiving and sending RTP packets. Here, issues, such as on which layer my implementation should reside, require careful deliberation due to possible inter-layer access and data sharing. Since this thesis has focused only on the adaptation of audio CODECs, we will focus on the audio part of the Minisip media system.

### 3.1.4 Multi-Session Support of Minisip

Support for conference calls is one of the most distinctive features of Minisip. A conference call usually involves several participants, leading to creations of multiple media sessions on each user agent. Minisip implements sophisticated mixing (including spatial audio [42]) for such conferences. So, there should be a corresponding estimate of network performance for each session, thus I sought to minimize dependencies



between my implementation and the lower-layer Minisip multimedia system. An example of a multi-session call using Minisip is shown in Figure 10 (Note that the figure does not show the communication between the other two parties -- as this also occurs directly between them in a third session.).

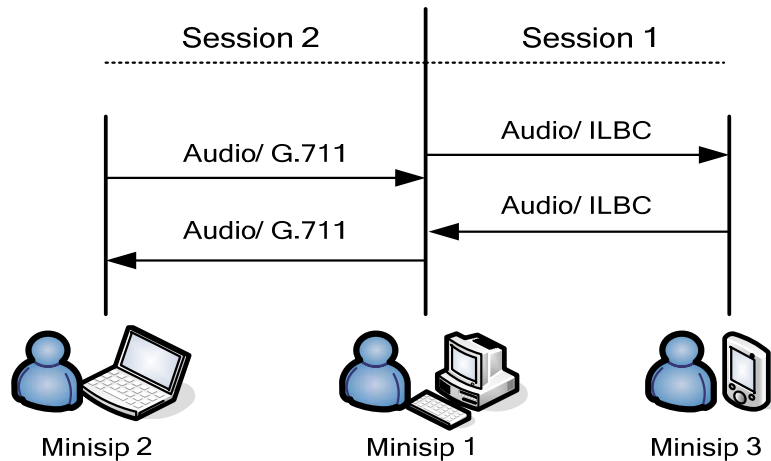


Figure 10: Example of Multi-Session Call

### 3.1.5 Network Performance Estimation Limitations

As was discussed in sections 2.2.2 and 2.2.3, current network performance estimation techniques for measuring available end-to-end bandwidth as well as packet delay and loss have certain restrictions which prohibited their integration into Minisip. Because these techniques usually generate a considerable number of probing packets, this would lead to competing traffics which would cause the application's performance to decrease. Additionally, accurate available bandwidth estimations are usually based on statistical models, which require special algorithms, and additional computation resources (which might not be available on a portable platform).

Therefore network performance estimation module to be integrated into Minisip should meet the following two requirements:

- Un-intrusive the number of the additional feedback packets or probes should be relatively small compared with the number of the media packets; so as to lessen the communication overhead due to the estimation module.
- Light weight the amount of the computational resources required by the estimation module should not significantly degrade service quality at the end point, for example, it should not introduce additional jitter, delay, etc.

## 3.2 Implementation

This section will give a general description of the implementation of a proposed CODEC Adaptation Module. Note that in this thesis the estimation and adaptation are implemented as a single CODEC Adaptation Module (consisting of an RTCP Feedback section, itself consisting of three submodules, and a CODEC Manager). There are clear interfaces between these two parts as described in section X.Y.

### 3.2.1 QoS Feedback using RTCP

As described in section 2.1.7, the primary function of RTCP [24] is to provide feedback on the quality of data received. A compound RTCP packet contains several types of RTCP packets, such as Sender Report (SR), Receiver Report (RR), Source Description Items (SDS), etc. For Minisip CODEC adaptation, the most important information and criterion is packet loss, which is carried in both RTCP Sender and Receiver Reports. The RTCP Sender Reports are sent by the RTP receivers who are also senders, by contrast, RTCP Receiver Reports are usually sent by the ones who only receive. To simplify our analysis and implementation, we assume that participants in calls using Minisip are all active senders, therefore, only the RTCP Sender Report was implemented and integrated into Minisip. The adequacy of this report is described below.

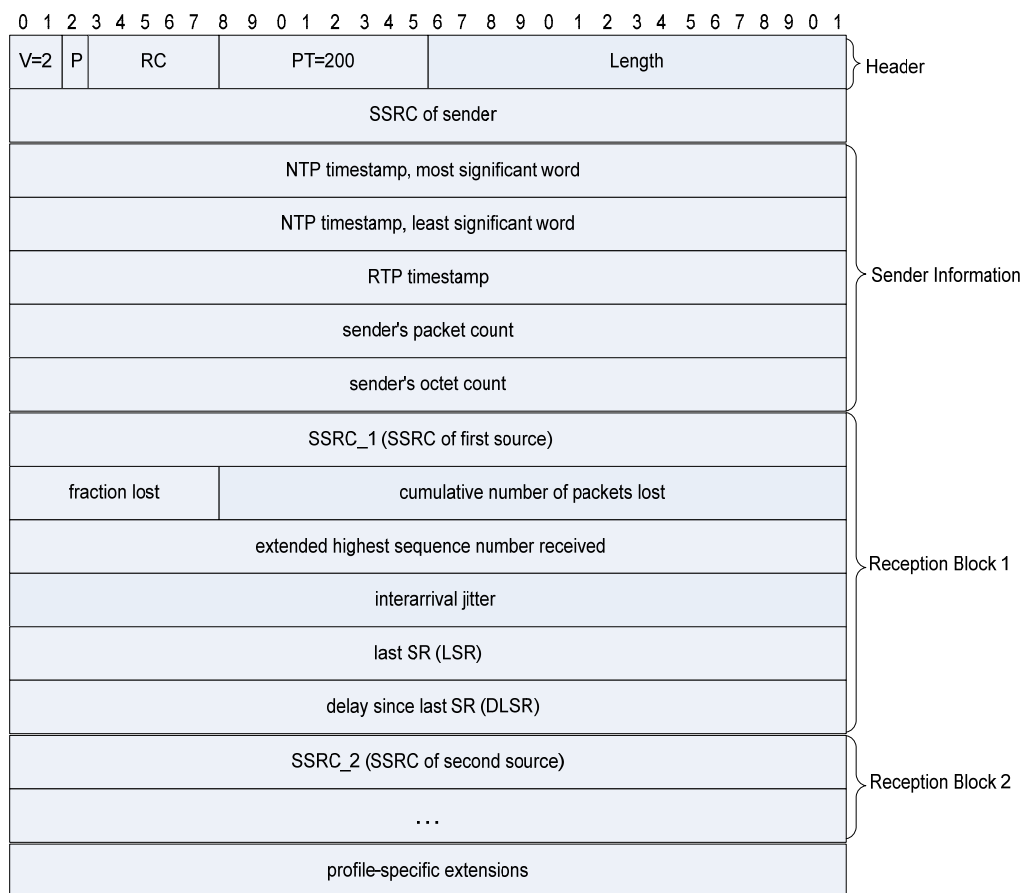


Figure 11: RTCP Sender Report Structure

As we can see in Figure 11, the RTCP Sender SR consists of three sections, and possibly a fourth section of profile-specific extension if defined. The sender information section must be presented in an RTCP SR, used by the active senders to do synchronizations and so on. The number of the reception report blocks depends on the number of the sources heard by this sender. The only difference between RTCP SR and RTCP RR is that RR does not have the sender information section. The UML class diagram of RTCP report is shown in Figure 12. Two important statistics in these reception reports—fraction loss and jitter are described below.

- Fraction loss

The fraction loss specifies the fraction of the RTP packets lost from a particular source since the last sender report or receiver report was sent. The value is the number of the RTP packets lost in the last RTCP reporting interval divided by the number of the RTP packets expected in the last RTCP reporting interval. Once the loss is negative due to duplicates, the value is set to zero. This value defines as the average packet loss rate in each RTCP reporting interval. This is the most important parameter used for the CODEC adaptation decision making as implemented in this thesis.

- Interarrival jitter

The jitter is the variation in the transit delay that packets experience while traversing a network. It is caused by queuing and serialization effects on the packet path. The Interarrival jitter value in an RTCP Sender Report is measured in timestamp units and is defined as the smoothed absolute value of the deviation in packets spacing at the receiver compared to the sender for pairs of a packets. The deviation  $D$  between two successive incoming RTP packets is calculated in the following equation, where  $S_a$  is the timestamp of packet  $a$ ,  $R_a$  is the arrival time of packet  $a$  expressed in timestamp units.

$$D(a, b) = (R_a - R_b) - (S_a - S_b) = (R_a - S_b) - (R_b - S_a)$$

The interarrival jitter is calculated continuously as each packet is received from each particular source. The value of interarrival jitter  $J$  is sampled according to the formula below, where  $J(i-1)$  is the jitter value calculated when the previous packet  $i-1$  was received.

$$J(i) = J(i-1) + (|D(i-1,i)| - J(i-1))/16$$

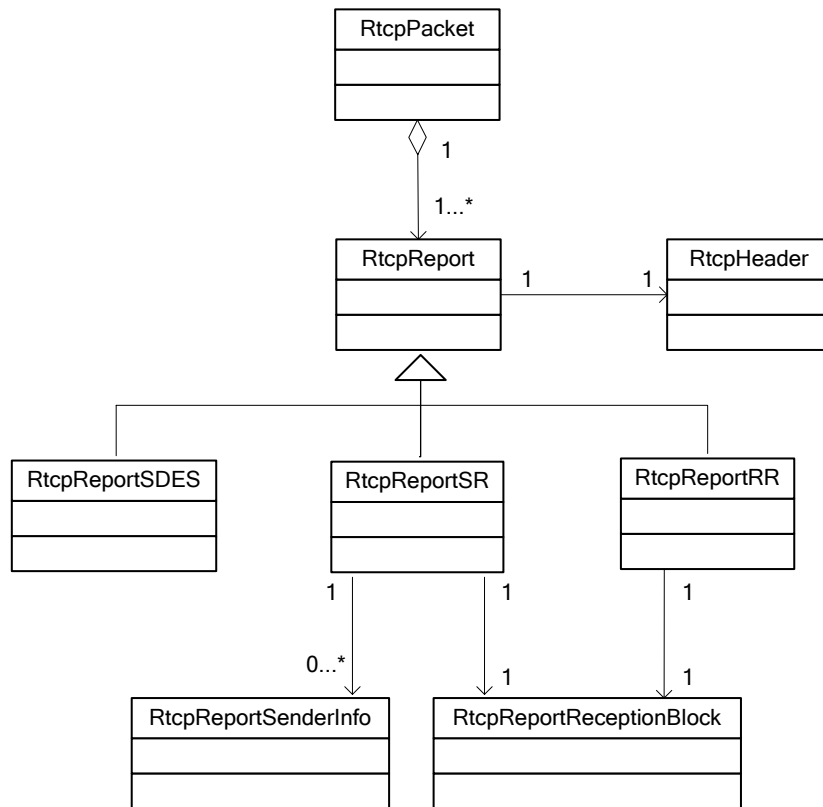


Figure 12: Class Diagram of RTCP

### 3.2.2 RTCP Feedback Module

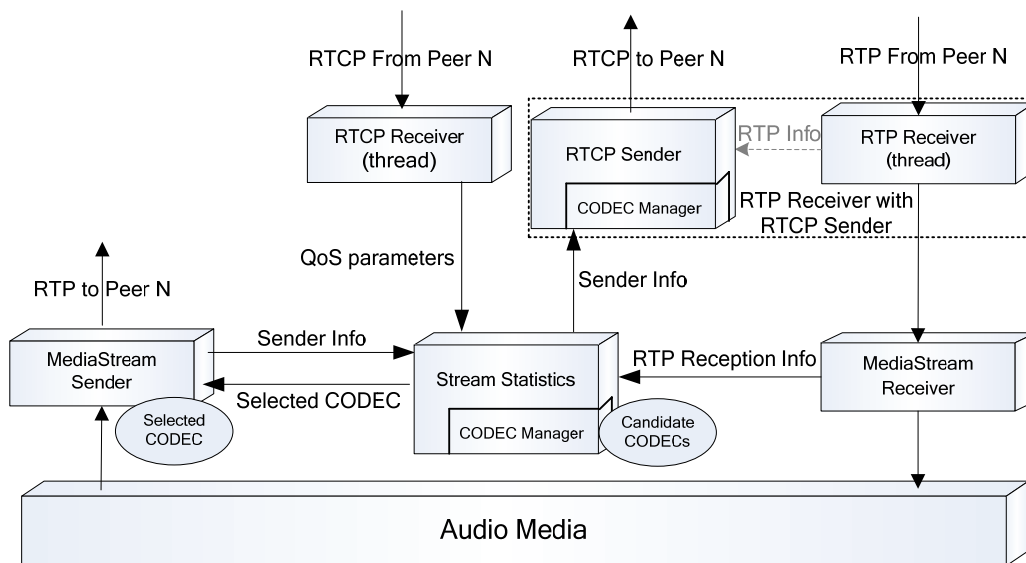


Figure 13: Audio Media System with RTCP

The RTCP Feedback module consists of three submodules (see Figure 13)—RTCP Sender, Stream Statistics, and RTCP Receiver. General descriptions about the implementation of this module are listed below.

### 3.2.2.1 RTCP Sender

The responsibility of the RTCP Sender is to create RTCP compound packets and send them. Some of the important information carried in an RTCP Sender Report is calculated based upon RTP packets received, so, it is obvious that the RTCP sender submodule should be placed in the RTP receiver. However, some other issues were exposed due to the structure of the RTCP Sender Report and the structure of Minisip's media system. As is shown in Figure 11, the sender information such as RTP timestamp, sender's packet count, and sender's octet count carried in the Sender Report should be retrieved from the Mediastream Sender, while from Figure 9, we can see that there is no direct connection between the RTP receiver and a Mediastream Sender, therefore, a new Stream Statistics submodule was implemented to bridge these two blocks.

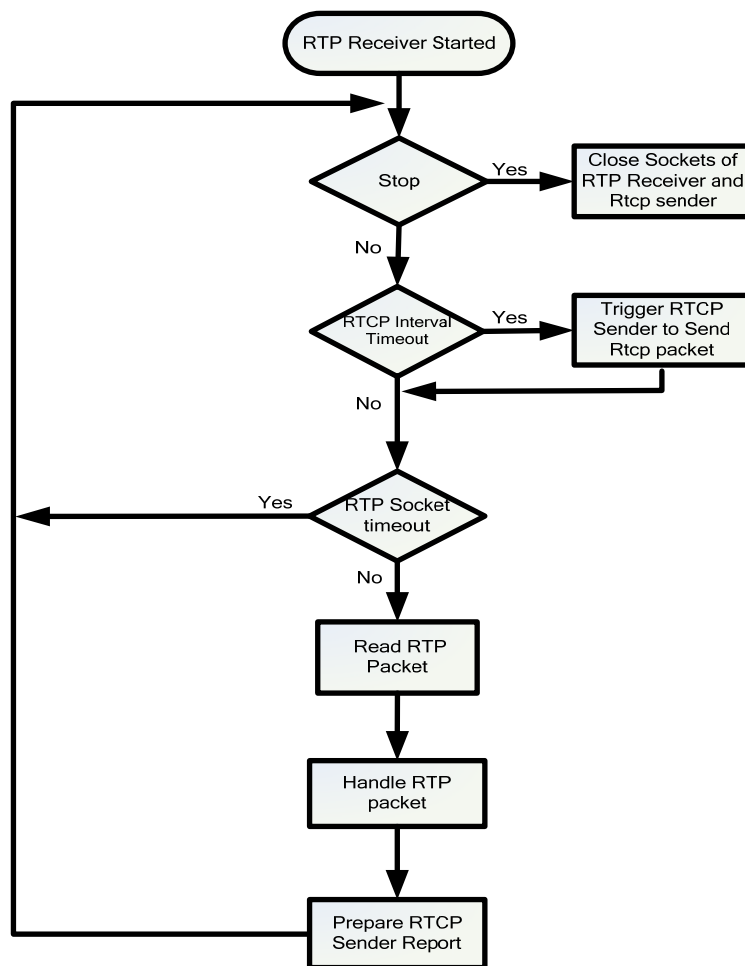


Figure 14: Flowchart of RTP Receiver with RTCP Sender

The RTCP Sender sends RTCP packets triggered by the expiration of a timer in the RTP Receiver (Figure 14). Currently, this RTCP implementation is not applicable for audio conference, because in a conference, a single session usually involves several participants leading to the creations of the same number of the RTCP modules as the number of the participants, which is considered inefficient, therefore, the RTCP transmission interval [24] in the current implementation is fixed and RTCP reports will not be sent for conferences.

### 3.2.2.1 Stream Statistics

The Stream Statistics submodule was implemented to collect and maintain statistics about each incoming and outgoing media stream, the important information includes:

- SSRC the synchronization source identifier of sender's
- Sender's Packet Count: the total number of the packets sent since the call has started
- Sender's Octet Count: the total number of the octets sent since the call has started
- Receiver's Packet Count the total number of the packets received since the call has started
- Receiver's Octet Count the total number of the octets received since the call has started
- Fraction Lost the one-way fraction loss from sender to receiver in a RTCP transmission interval

### 3.2.2.3 RTCP Receiver

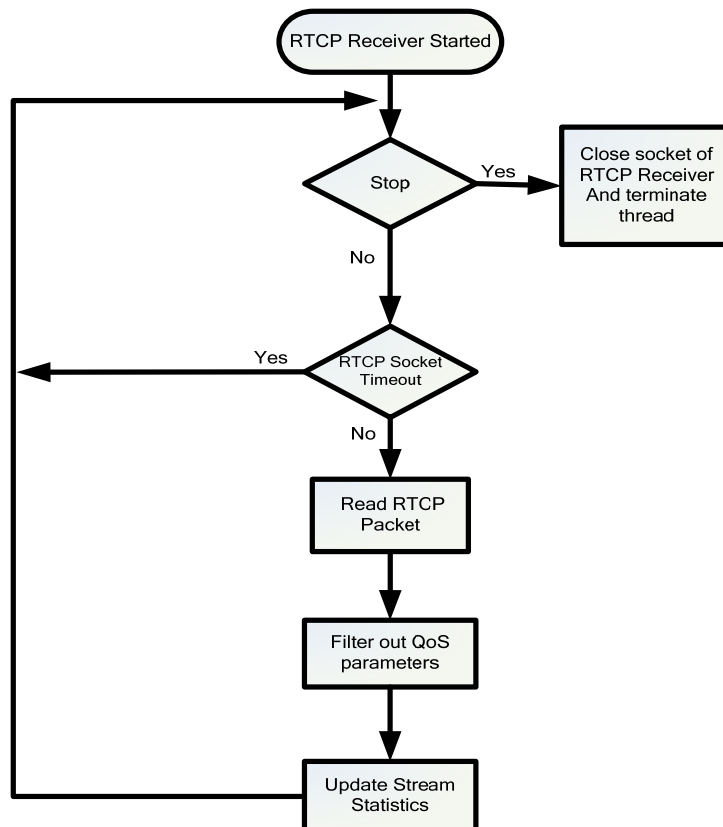


Figure 15: Flowchart of RTCP Receiver

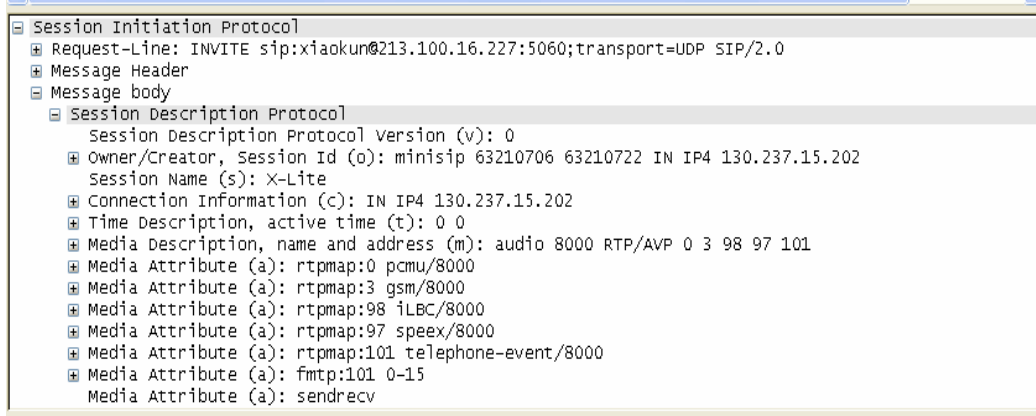
The RTCP Receiver is a thread, which continuously listens for incoming RTCP packets, parses the RTCP Sender Reports received, and filters out the QoS parameters (i.e., fraction loss and jitter). All the QoS parameters are then saved in the corresponding Stream Statistics. The RTCP Receiver thread is started during the initialization of the Mediastream Sender and stopped when Mediastream Sender is terminated. Figure 15 shows the flow chart of RTCP Receiver.

### 3.2.3 Multiple CODECS

At present, there are three audio CODECs built for the Windows version of Minisip: G.711 ( $\mu$ -Law), GSM (FR) CODEC, and Speex (Narrowband). All these CODECs are very different with regard to their bandwidth requirements, resilience to packet loss, and some special features such as Speex's variable bit-rate (VBR) and voice activity detection (VAD).

Originally, Minisip only created an object corresponding to the first matching CODEC in the SIP/SDP negotiation, and only this matching CODEC would appear in the SIP/SDP answer. So, modifications were made to include media information for all the matching CODECs in the SIP/SDP answer. Figure 16 and Figure 17 show the SIP/SDP INVITE message from X-Lite [34] to Minisip and the SIP/SDP answer from the reverse direction as captured by Ethereal. Here we can see that X-Lite has five audio CODECs, and Minisip has three audio CODECs that match the SIP invitation from X-Lite.

No. -	Time	Source	Destination	Protocol	Info
1317	27.117988	195.37.77.99	213.100.16.227	SIP/SDP	Request: INVITE sip:xiaokun@213.1
1319	27.120182	195.37.77.99	213.100.16.227	SIP/SDP	Request: INVITE sip:xiaokun@213.1
1857	33.085519	213.100.16.227	195.37.77.99	SIP/SDP	Status: 200 OK, with session desc

```

Session Initiation Protocol
  Request-Line: INVITE sip:xiaokun@213.100.16.227:5060;transport=UDP SIP/2.0
  Message Header
  Message body
    Session Description Protocol
      Session Description Protocol version (v): 0
      Owner/Creator, Session Id (o): minisip 63210706 63210722 IN IP4 130.237.15.202
      Session Name (s): X-Lite
      Connection Information (c): IN IP4 130.237.15.202
      Time Description, active time (t): 0 0
      Media Description, name and address (m): audio 8000 RTP/AVP 0 3 98 97 101
      Media Attribute (a): rtpmap:0 pcmu/8000
      Media Attribute (a): rtpmap:3 gsm/8000
      Media Attribute (a): rtpmap:98 iLBC/8000
      Media Attribute (a): rtpmap:97 speex/8000
      Media Attribute (a): rtpmap:101 telephone-event/8000
      Media Attribute (a): fmp:101 0-15
      Media Attribute (a): sendrecv
  
```

Figure 16: SIP/SDP INVITE message from X-Lite to Minisip

No. -	Time	Source	Destination	Protocol	Info
1317	27.117988	195.37.77.99	213.100.16.227	SIP/SDP	Request: INVITE sip:xiaokun@213.1
1319	27.120182	195.37.77.99	213.100.16.227	SIP/SDP	Request: INVITE sip:xiaokun@213.1
1857	33.085519	213.100.16.227	195.37.77.99	SIP/SDP	Status: 200 OK, with session desc

User Datagram Protocol, Src Port: 5060 (5060), Dst Port: 5060 (5060)

- [-] Session Initiation Protocol
  - [-] Status-Line: SIP/2.0 200 OK
  - [-] Message Header
  - [-] Message body
    - [-] Session Description Protocol
      - Session Description Protocol Version (v): 0
      - [-] Owner/Creator, Session Id (o): - 3344 3344 IN IP4 213.100.16.227
      - Session Name (s): Minisip Session
      - [-] Connection Information (c): IN IP4 213.100.16.227
      - [-] Time Description, active time (t): 0 0
      - [-] Media Description, name and address (m): audio 32398 RTP/AVP 0 3 97
      - [-] Media Attribute (a): fmtp:0 pcmu/8000
      - [-] Media Attribute (a): rtpmap:0 pcmu/8000
      - [-] Media Attribute (a): fmtp:3 gsm/8000
      - [-] Media Attribute (a): rtpmap:3 gsm/8000
      - [-] Media Attribute (a): fmtp:97 speex/8000
      - [-] Media Attribute (a): rtpmap:97 speex/8000

Figure 17: SIP/SDP answer from Minisip to X-Lite

### 3.2.4 Audio Adaptation

#### 3.2.4.1 VoIP CODEC Bandwidth

The amount of the bandwidth required to carry voice over an IP network is dependent upon a number of factors: choice of CODEC and sample period, IP header, transmission medium, etc. The voice data payload is usually wrapped in successive layers of headers in order to deliver it to its destination. In IP networks (on top of links such as: IEEE 802.3, Ethernet, IEEE 802.11) the common layers for VoIP data transmission include IP, UDP, and RTP, so the bandwidth needed for each CODEC can be calculated according to the payload size plus the IP overhead. As is shown in Figure 18, a G.711 20 ms sample period payload size is 160 octets, and a fixed 40 octets of IP/UDP/RTP are added, which means that a 64 kbps CODEC bitrate plus 16 kbps IP overhead, results in 80 kbps in total. For different transmission media, additional medium-specific overhead should be taken into account. Over an IEEE 802.3 Ethernet, a fixed 38 octets overhead includes an Ethernet Preamble, Ethernet header, Ethernet CRC, and an inter-frame gap, introduces effectively an additional 15.2 kbps extra throughput (as shown in Figure 18). While in IEEE 802.11, the situation is more complicated, usually the IEEE 802.11 overhead is 70 octets, including Physical Layer Convergence Protocol (PLCP) header, 802.11 packet header, 802.11 packet checksum, as well as an inter-frame gap.



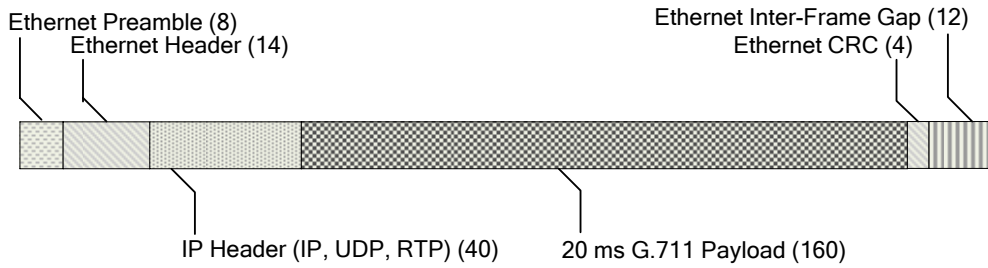


Figure 18: G.711 CODEC, 1 x 20 ms sample per packet results in 95.200 kbps over an Ethernet

### 3.2.4.2 CODEC Switching

The CODEC adaptation was realized by switching between the CODECs or changing the CODEC parameters according to the one way packet loss feedback carried in the RTCP Sender Reports from the calling parties. This idea is based on the assumption that with heavy competing traffic, big packets in a congested router might be dropped with higher probabilities than small ones, and in wireless environment, big packets have higher probabilities for errors. There are six CODEC states (Figure 19) enabled in the Windows version of Minisip, and each of them provides different call qualities and transmission features.

Table 2: Available CODEC states in Minisip (20ms frame, 1 frame per packet, 50 packets per second)

CODEC	bytes/frame	bits/sec	bytes/packet ( +IP overhead )	bits/sec (+IP overhead )
<b>G.711</b>	160	64000	200	80000
<b>GSM</b>	33	13000	83	29000
<b>Speex-24k</b>	62	24600	102	40600
<b>Speex-18k</b>	46	18200	86	34400
<b>Speex-11k</b>	28	11200	68	27200
<b>Speex-8k</b>	20	8000	60	24000

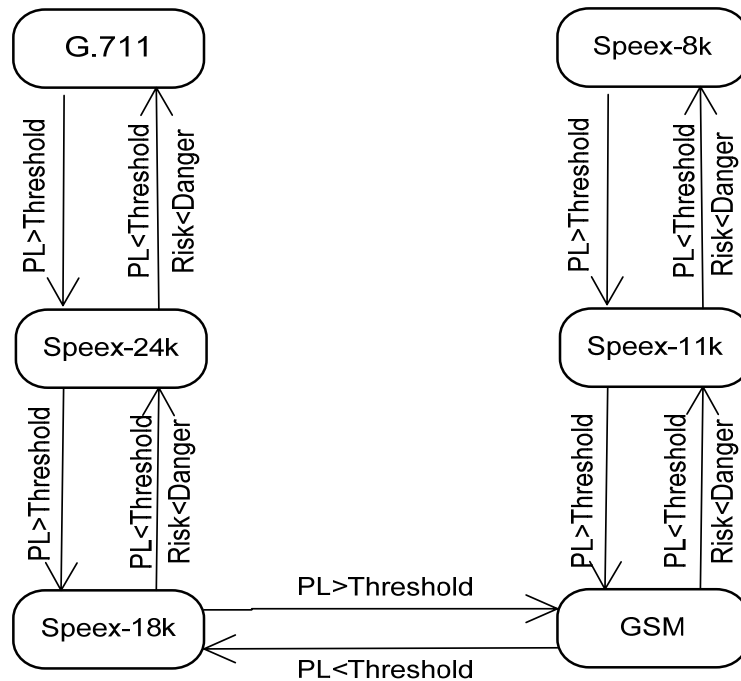


Figure 19: State Transition of CODEC Switching

The CODEC states are ranked by their bitrates. Once the packet loss rate is found higher than the specified threshold, the CODEC state will be changed to a lower bitrate if one is available. In contrast, if the packet loss rate goes below a threshold, then the CODEC state will be switched back to a higher bitrate one. CODEC switching will happen only between two adjacent CODEC states to avoid a sudden change of the call quality perceived. There might be some risks in switching from a lower bitrate state back to a higher bitrate one, because this may lead to frequent CODEC state switching up-and-down, so hysteresis was added to avoid this situation. This hysteresis is implemented by a risk count, if it is too high, no action will be taken, even though the packet loss is lower than the threshold. However, if the network performance recovers from a poor condition, the risk count of each CODEC state will be reset, and it only happens when the packet loss is persistently lower than the packet loss threshold for a certain period of time. Figure 19 shows the state transition between CODECs; any state could be the initial state which is specified as the caller's CODEC preference in their Minisip configuration file.

In Minisip, originally only one CODEC was used to encode voice data in each established call, and an instance of this CODEC was created by Mediastream Sender. To realize dynamic CODEC switching, a CODEC Manager was implemented, which remembers all the matching CODECs specified in the SIP negotiation and acts as a decision agent to select both the proper CODEC and CODEC properties that should be used based on the one-way packet loss feedback. This CODEC switching will only take place when a new RTCP Sender Report is received, at which point the corresponding Stream Statistics is updated, providing new information to make a potentially new CODEC choice.

## 4. Test and Evaluation

The following sections will present the results of some experiments and measurements on the implementation described above. The experiments were conducted in both wired and wireless environments.

### 4.1 CODEC Adaptation Test

The objective of this test is to verify the correctness of the packet loss feedback and the CODEC adaptation mechanism. The experimental setup is shown in Figure 20, where two Windows machines A (desktop computer) and B (laptop) running Minisip, and a Linux machine running NIST Net emulates the path by providing a bounded bandwidth, causing packet loss, adding delay, etc.

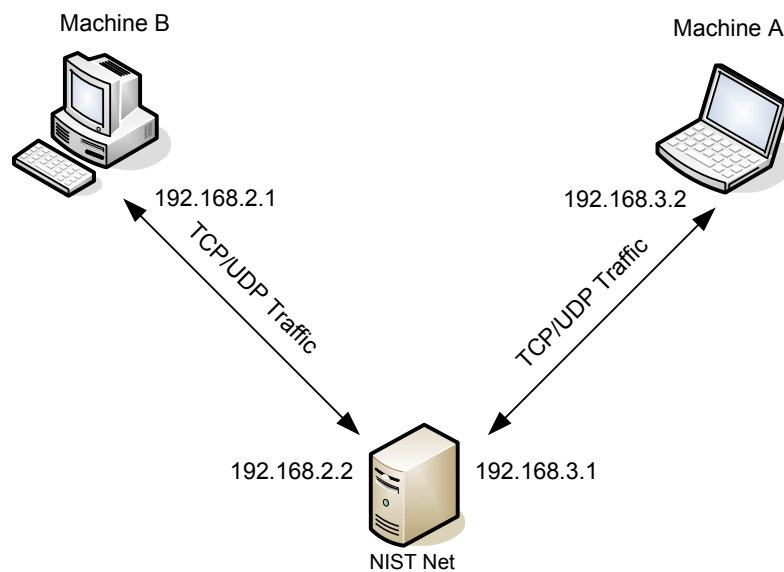


Figure 20: Experimental Setup

Due to the limitation of the current implementation, CODEC switching will only be triggered by packet loss feedback, thus, NIST Net was configured to drop packets through a UNIX shell script, the configuration of NIST Net is shown in Table 3. Minisip's packet loss threshold for CODEC switching was set to 3%, and the feedback interval was 5 seconds. Three audio CODECs (G.711, GSM, and Speex) were used in the test.

Table 3: NIST Net Configurations (Path A to B)

NIST Net Configuration	Packet Drop	Duration (Seconds)
N1	5%	60
N2	0	60
N3	6%	20
N4	1%	30
N5	5%	30
N6	0	60

Table 4: CODEC Risk Threshold

CODEC	G.711	Speex-24K	Speex-18K	GSM	Speex-11K
Risk Threshold	1	2	3	4	5

Table 4 shows the risk threshold of each CODEC state, for example, the risk threshold of G.711 is 1, which means that this CODEC can only be switched back once.

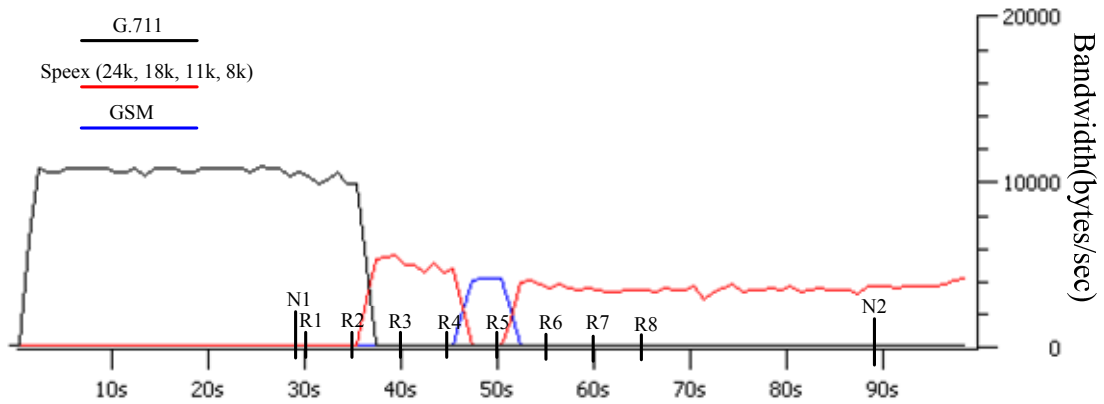


Figure 21: Bandwidth Utilization with CODEC switching (configuration N1 through start of N2)

Figure 21 were obtained from data collected by Ethereal based on the packets captured on B. The plot shows the change of the bandwidth utilization (path A to B) under NIST Net configuration N1 due to CODEC state switching. As we can see in Figure 21, configuration N1 was started at about the 29th second; each  $R_i$  represents an RTCP SR feedback received by A and its corresponding time. The fraction losses carried in each RTCP SR sent by B and corresponding CODEC switching on A are shown in Table 5.

Table 5: Fraction loss and CODEC switching (configuration N1)

Time (Second)	RTCP SR	Fraction Loss Calculated (%)	CODEC Switching
32	R <sub>1</sub>	2	G.711
37	R <sub>2</sub>	8	G.711 → Speex-24K
42	R <sub>3</sub>	6	Speex-24K → Speex-18K
47	R <sub>4</sub>	6	Speex-18K → GSM
52	R <sub>5</sub>	6	GSM → Speex-11K
57	R <sub>6</sub>	4	Speex-11K → Speex-8K
62	R <sub>7</sub>	5	Speex-8K
67	R <sub>8</sub>	6	Speex-8K

In Table 4, we can see that, RTCP SR was sent every 5 seconds by B. The first CODEC switching happened when R<sub>2</sub> was received by A—7 seconds after NIST Net configuration N1 was started, this is because fraction loss value carried in each RTCP SR feedback was calculated based on the observation of RTP packets sent by A within the previous 5 seconds (RTCP transmission interval). We can also see that, when the fraction loss value is above the packet loss threshold (3%), CODEC state switching happens. Once the CODEC state is switched to the lowest bitrate one and the packet loss is still high, this CODEC state will be always used.

Figure 23 shows the bandwidth utilization and CODEC switching during the 60 seconds of NIST configuration N2. During this 60 seconds, NIST Net was configured with no packet drop on path A to B, therefore, as we can see, the CODEC state was switched back from Speex-8k to G.711, leading to an increasing bandwidth utilization.

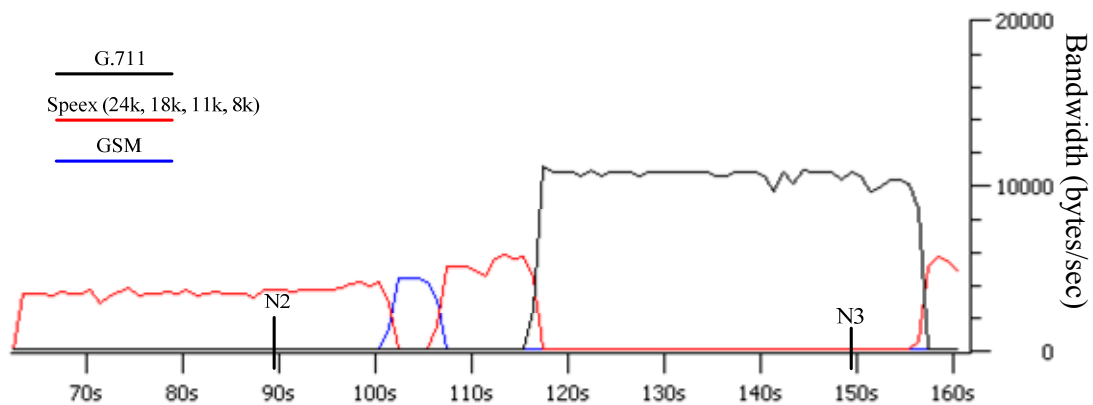


Figure 22: Bandwidth Utilization with CODEC switching (N2)

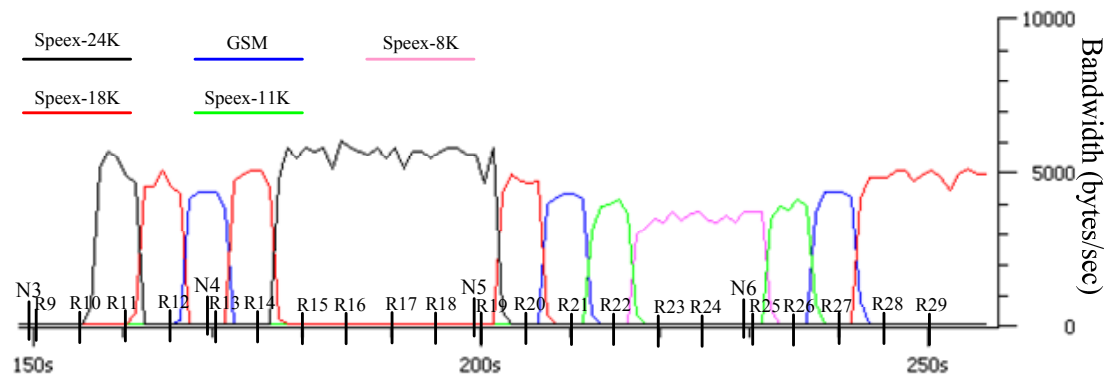


Figure 23: Bandwidth Utilization with CODEC switching (configurations N3, N4, N5, and N6)

Table 6: Fraction loss and CODEC switching (configurations N3, N4, N5, and N6)

Time (Second)	RTCP SR	Fraction Loss Calculated (%)	CODEC Switching
152	R <sub>9</sub>	2	G.711
157	R <sub>10</sub>	5	G.711 → Speex-24K
162	R <sub>11</sub>	7	Speex-24K → Speex-18K
167	R <sub>12</sub>	6	Speex-18K → GSM
172	R <sub>13</sub>	1	GSM → Speex-18K
178	R <sub>14</sub>	1	Speex-18K → Speex-24K
182	R <sub>15</sub>	1	Speex-24K
187	R <sub>16</sub>	0	Speex-24K
192	R <sub>17</sub>	0	Speex-24K
197	R <sub>18</sub>	1	Speex-24K
202	R <sub>19</sub>	3	Speex-24K → Speex-18K
207	R <sub>20</sub>	5	Speex-18K → GSM
212	R <sub>21</sub>	3	GSM → Speex-11K
217	R <sub>22</sub>	4	Speex-11K → Speex-8K
222	R <sub>23</sub>	5	Speex-8K
227	R <sub>25</sub>	4	Speex-8K
232	R <sub>25</sub>	1	Speex-8K → Speex-11K
237	R <sub>26</sub>	0	Speex-11K → GSM
242	R <sub>27</sub>	0	GSM → Speex-18K
247	R <sub>28</sub>	0	Speex-18K
252	R <sub>29</sub>	0	Speex-18K

The CODEC state switching and bandwidth utilization during NIST Net configurations N3, N4, N5, and N6 is shown in Figure 23, and the corresponding fraction loss values carried in RTCP SR feedbacks received by A are shown in Table 6. In this table, there are two specific points to note, the first one is RTCP SR  $R_{15}$ , where we can see that the fraction loss value in  $R_{15}$  was below the packet loss threshold, but the CODEC did not switch from Speex-24K back to G.711, this is because G.711 was already switched back once during configuration N2, and the risk count of G.711 went above its risk threshold. The other one is  $R_{28}$ —the fraction loss was 0 and the CODEC state was still Speex-18K, this is because Speex-24K was switched back twice during configurations N2, and N3. From these obtained results, we can see that, if one-way packet loss changes frequently below and above the packet loss threshold, Minisip will tend to select the lower bitrate CODEC to encode voice data and this is determined by the risk threshold of each CODEC state. (Note that the CODECs' risk count values will not be reset until 500 successive RTCP Sender Reports with fraction loss lower than the packet loss threshold are received)

Time(s)	(Port) A	B (Port)	Comments
207.970	(30006)	(30444)	Payload type=speex, SSRC=26500, Seq=16498, Time=1645569
207.973	(30006)	(30444)	Payload type=ITU-T G.711 PCMU, SSRC=14718, Seq=29767, Time=1664314
207.990	(30006)	(30444)	Payload type=speex, SSRC=26500, Seq=16499, Time=1645729
207.993	(30006)	(30444)	Payload type=ITU-T G.711 PCMU, SSRC=14718, Seq=29768, Time=1664474
208.010	(30006)	(30444)	Payload type=speex, SSRC=26500, Seq=16500, Time=1645889
208.013	(30006)	(30444)	Payload type=ITU-T G.711 PCMU, SSRC=14718, Seq=29769, Time=1664634
208.030	(30006)	(30444)	Payload type=speex, SSRC=26500, Seq=16501, Time=1646049
208.033	(30006)	(30444)	Payload type=ITU-T G.711 PCMU, SSRC=14718, Seq=29770, Time=1664794
208.050	(30006)	(30444)	Payload type=speex, SSRC=26500, Seq=16502, Time=1646209
208.057	(30006)	(30444)	Payload type=ITU-T G.711 PCMU, SSRC=14718, Seq=29771, Time=1664954
208.070	(30006)	(30444)	Payload type=GSM 06.10, SSRC=26500, Seq=16503, Time=1646369
208.073	(30006)	(30444)	Payload type=ITU-T G.711 PCMU, SSRC=14718, Seq=29772, Time=1665114
208.089	(30006)	(30444)	Payload type=GSM 06.10, SSRC=26500, Seq=16504, Time=1646529
208.093	(30006)	(30444)	Payload type=ITU-T G.711 PCMU, SSRC=14718, Seq=29773, Time=1665274
208.109	(30006)	(30444)	Payload type=GSM 06.10, SSRC=26500, Seq=16505, Time=1646689

Figure 24: Media Flow with CODEC Switching

Another result observed from this test is shown in Figure 22, from the media flow captured by Ethereal, we can see that in two directions A to B and B to A, voice data was encoded by different CODECs. This result is really expected, because usually in end-to-end communications, the network conditions can vary in each direction, so, instead of using the same CODEC or the same CODEC settings in each Minisip user agent, CODECs can be selected dynamically according to the one-way network performance, thus, to separately provide optimal service quality to each Minisip user.

Finally, as observed from Ethereal, the average RTCP throughput from A to B is about 19 bytes/second, while the RTP throughput on the same path is about 10700 bytes/second. Even if the RTCP transmission interval is set to 1 second, the RTCP throughput is only about 100 bytes/second. So compared with the RTP

traffic, the generated RTCP traffic is negligible.

## 4.2 Competing Traffic (TCP) Test

The objective of this test is to verify the relation between RTP packet size and packet loss behavior with heavy TCP traffic. In this test, machine A was connected to the WLAN via the wireless access point in the lab. Calls were established between A and B, and in the meantime, several TCP connections were created between A and B and TCP packets were sent from A to B to generate competing traffic. Two CODECs—G.711 and GSM were used in this test separately with CODEC switching disabled.

Table 7: Packet Loss of G.711 and GSM encoded data with effect of TCP

<b>Number of TCP</b>	<b>Packet loss(G.711)</b>	<b>Packet loss(GSM)</b>
10	1.6%	1.1%
15	2.6%	2.5%
20	3.3%	3.1%

As we can see in Table 7, although the packet size (including IP overhead) of G.711 encoded voice data is almost three times larger than the packet size of GSM encoded one, the packet loss rates (each packet loss rate is the average value obtained from 5 tests) are similar when facing the same amount of TCP competing traffic. The reason for this result could be that all these TCP connections were not synchronized [40], and they tried to use up all the available bandwidth limited only by their flow control mechanism. Therefore, even if the RTP transmission rate decreases by switching the CODECs, the resulting available bandwidth will be consumed promptly by TCP connections, and this will not contribute to improving the call quality.



# 5. Conclusions and Future Work

## 5.1 Conclusions

The objective of this thesis was to create and evaluate an adaptive mechanism by means of CODEC adaptation to achieve QoS control for the SIP user agent Minisip.

The tests of the implemented prototype show that using RTCP can be a very good way to provide QoS feedback between Minisip user agents. By switching the CODEC or adjusting the CODEC's parameter(s) (bitrate) according to the RTCP feedback, the one-way throughput can be changed dynamically at runtime of Minisip, which could potentially improve the call quality and is especially useful in a wireless environment due to the variations of the network performance, especially as a user moves about in such an environment. Also, we can see that, this implementation allowed the utilization of different CODECs in different directions, which can optimize the call quality perceived by each Minisip user during a call.

## 5.2 Future Work

This thesis work is just a beginning of a new focus on the QoS control of the SIP user agent Minisip, so there is a lot of work that could be done to extend the current implementation. Some suggestions are:

- In this thesis work, only the audio CODEC adaptation of Minisip was implemented, therefore, adding video CODEC adaptation will be a significant extension (section 1.3.2). Unlike audio communication, the support for video requires an additional device, such as a web camera, so, this means that not every Minisip user can send video streams, but can definitely receive such a stream if the calling part has such a device. In this case, RTCP Receiver Reports should be used and added to the RTCP feedback.
- Packet loss is not the only factor that reflects a poor network condition; therefore, future work could perform the CODEC adaptation based on a combination of several QoS parameters, such as delay and jitter. The relations between these QoS parameters and the corresponding network performance, as well as the thresholds described in section 1.3.2 should be explored and exploited.
- As noted in section 1.3.2, when the loss rate is persistently high the session will automatically be terminated - so as to give network resources to others. Because this thesis did not consider the question of when to restart a terminated session, this remains as important future work. This is particularly important because unlike a traditional phone call where the user would need to dial the number (or manually hit the re-dial button) the computer could persistently attempt to re-establish the connection; thus when the network resources are unavailable this could lead to new session establishment attempts -- which are not congestion controlled (e.g., the attempted TCP connection establishment to the SIP server is **not** congestion controlled -- only the transfer of data over this connection is controlled, but since the amount

of data to be sent is very small - there is effectively no control).

- As described in section 2.2.2.1, the current RTCP feedback module is not applicable to Minisip's conference call, however, this could be realized by re-implementing this module to create per conference state instead of per incoming/outgoing media stream, and the RTCP packets should be broadcast/multicast/unicast to all the conference participants. Some of the related problems, such as the calculation of the RTCP transmission interval (this calculated interval must be able to avoid synchronization with other participants, and be able to limit the total throughput of RTCP traffic) should be explored.
- The result of the test shown in section 4.2 gives some implications about future work which could combine the CODEC adaptation with TFRC (TCP-Friendly Rate Control) [41] to achieve fairness with TCP, meanwhile, rate control could be done by switching CODECs instead of changing the transmission rate. Further studies should focus on the computational load of this combined mechanism and the possibility to integrate it into Minisip that running on a handheld device.

## References

1. A. K. Dey and G.D. Abowd, "Toward a better understanding of context and context-awareness", Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology, 1999.
2. James Curtis and Tony McGregor, "Review of bandwidth estimation techniques", in New Zealand Computer Science Research Students' Conference, University of Canterbury, New Zealand, April 2001.
3. K. Harfoush, A. Bestavros, and J. Byers, "Measuring bottleneck bandwidth of Targeted path segments", Technical Report BUCS-TR-2001-016, Boston University, Computer Science Department, July 2001
4. R. L. Carter and M. E. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks", *performance evaluation*, Vol. 27, 28, pp: 297-318, 1996
5. G. Q. Maguire Jr., "Voice over IP: Security and Mobility", Lecture notes, IVA Syd, Lund, Sweden, March 2004
6. Debashish Mitra: "Network convergence and voice over IP", Tata Consultancy Services, March 2001
7. Erik Eliasson and Johan Billien, Minisip, KTH, Stockholm, 2004, <http://www.minisip.org/>
8. The Infocomm Development Authority of Singapore (IDA), "WoWLAN—A Brief Introduction", Singapore, 16 April 2004
9. K. J. Khan and M-S Lang, "Voice over Wireless LAN analysis of MiniSIP as an 802.11 phone", a report submitted for course: Practical Voice over IP: SIP and related protocols, 29<sup>th</sup> June 2004
10. P. Nyberg, A. Bararsani, and A. Tong, "Multicodec Minisip, Final Report", KTH, Stockholm, Sweden May 2005 <http://csd.ssvl.kth.se/~csd2005-team15/mcms/>
11. Fraunhofer Fokus, "SIP versus H.323" <http://www.iptel.org/info/trends/sip.html> (Last access in November, 2005)
12. J. Rosenberg, H. Shulzrinne, G. Camarillo, et al., RFC 3261 "SIP: Session Initiation Protocol", June 2002
13. VoIP WiKi—a reference guide to all things VoIP, GSM CODEC(last update in August 2004), ILBC(last update in November 2005), homepage: <http://www.voip-info.org/wiki/>
14. J-M. Valin, "Speex CODEC manual (version 1.0)", March 2003. <http://www.speex.org/manual.pdf>

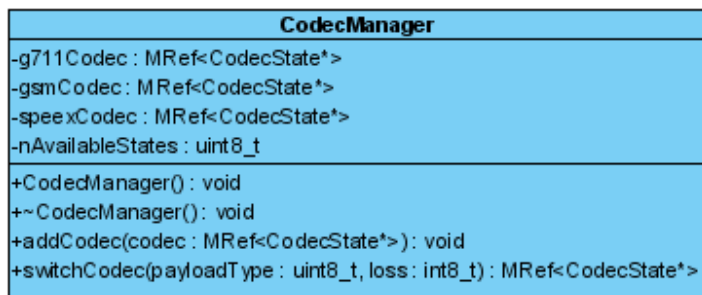
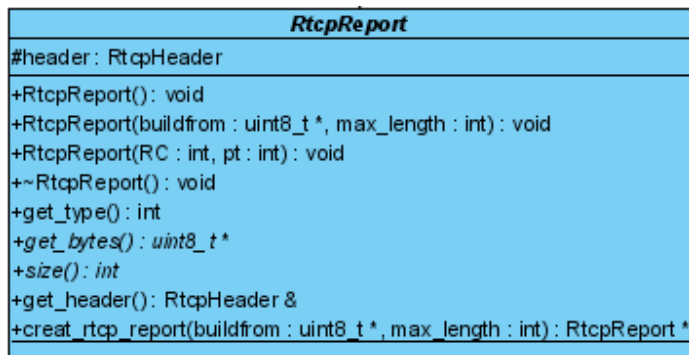
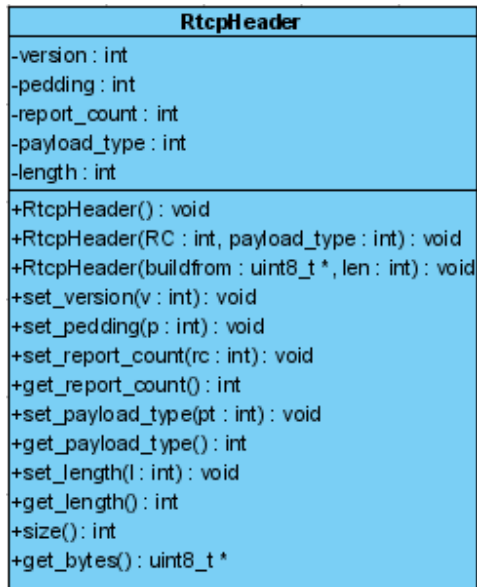
15. A. B. Doney, "Using pathchar to estimate internet link characteristics", in *Proceeding of ACM SIGCOMM*, 1999
16. V. Jacobson, pathchar—a tool to infer characteristics of internet paths, Presented at Mathematical Sciences Research Institute, 1997
17. S. Keshav, "Congestion Control in Computer Networks. PhD Thesis", University of California at Berkeley, September, 1991
18. K. Lai and M. Baker, "Measuring Link Bandwidth Using Deterministic Model of Packet Delay", in *Proceeding of ACM SIGCOMM*, 2000
19. Jean-Chrysotome Bolot, "End-to-end Packet Delay and Loss Behavior in the Internet", in *proceeding of ACM SIGCOMM*, 1993
20. Joel Sommers, Paul Barford, Nick Duffield, and Amos Ron, "Improving Accuracy in End-to-end Packet Loss Measurement", in *proceeding of ACM SIGCOMM*, 2005
21. D. Sanghi, A. K. Agrawala, and B. Jain, "Experimental assessment of end-to-end behavior on Internet", *Proc. IEEE Infocom '93*, San Francisco, CA, pp. 124-131, May 1993
22. Rakesh Arora, "Voice over IP: Protocols and Standards", [http://www.cse.ohio-state.edu/~jain/cis788-99/ftp/voip\\_protocols/index.html#3.-Session-Initiation-Protocol\(SIP\)](http://www.cse.ohio-state.edu/~jain/cis788-99/ftp/voip_protocols/index.html#3.-Session-Initiation-Protocol(SIP)), May 20, 2004
23. M. Handley and V. Jacobson, RFC 2327 "SDP—Session Description Protocol", April 1998
24. H. Schulzrinne, S. Cansner, R. Frederik, V. Jacobson, RFC 3550 "RTP—A Transport Protocol for Real-Time Applications", July 2003
25. G. S. Floyd and J. Kempf (Editors), "IAB Concerns Regarding Congestion Control for Voice Traffic in the Internet", IETF, RFC 3714, Network Working Group, March 2004.
26. G. Combs, et al., Ethereal-The world's most popular network protocol analyzer, homepage: <http://www.ethereal.com/>
27. M. Carson, the NIST Net, Homepage: <http://snad.ncsl.nist.gov/itg/nistnet/> (Last update in July, 2005)
28. S.Parker and C. Schmechel, RFC 2398"Some Testing Tools for TCP Implementers", August 1998

29. Skype Technology S. A., Skype-The whole world can talk for free, homepage: <http://www.skype.com>  
(Last access in May 1, 2006)
30. Softroute Corporation, Vbuzzer, homepage: <http://www.vbuzzer.com/>, 2005
31. The Internet Engineering Task Force (IETF), Transport Link Security (TLS) Charter, April, 2004  
<http://www.ietf.org/html.charters/tls-charter.html>
32. F. Maurer, "Push-2-talk in VoIP, decentralized", September 2004, homepage: <http://push2talk.flohweb.ch/>
33. ITU-T Recommendation G. 114, "One way transmission time", May 2000
34. CounterPath Solutions, Inc, homepage: <http://www.xten.net/> (Last access in 1st May, 2006)
35. Jürgen Morhöfer, "GSM 06.10 lossy speech compression " March 2006,  
<http://kbs.cs.tu-berlin.de/~jutta/toast.html#indices>
36. Liang Huang, "On-line storage versus local storage for mobile users", Master Thesis, KTH, Stockholm April 2006
37. Inmaculada Rangel Vacas, "Context Aware and Adaptive Mobile Audio", Master Thesis, KTH, Stockholm, March 2005
38. H. Schulzrinne, et al, RFC 2326 "Real Time Streaming Protocol (RTSP)", April 1998
39. I. Abad, "Secure Mobile VoIP" Master Thesis, KTH, Stockholm, Sweden, June 2003
40. H. Sawashima, Y. Hori, H. Sunahara, and Y. Oie, "Characteristics of UDP Packet Loss: Effect of TCP Traffic", INET, 1997
41. David Tlahuettl, "Analysis and implementation of TCP Friendly Rate Control in the Context of VOIP", Master Thesis, KTH, Stockholm, Sweden, December 2005
42. Ignacio Sánchez Pardo, "Spatial Audio for the Mobile User", Master Thesis, KTH, Stockholm, Sweden, 2005
43. Ali Nesh-Nash, "VoIP in a Resource Constrained Environment", Master Thesis, KTH, Stockholm, Sweden, March 2006

44. I. Miladinovic and J. Stadler, "Multiparty Conference Signaling using the Session Initiation Protocol (SIP)", Vienna University of Technology, Vienna, Austria, 2002

## Appendix

### A. UML Description of Related Classes



<b>RtcpReportReceptionBlock</b>
-ssrc : unsigned int -fraction_lost : unsigned int -cumulative_n_lost : int -seqHighCycle : uint16_t -seq_high : uint16_t -jitter : unsigned int -last_sr : unsigned int -dlsr : unsigned int
+RtcpReportReceptionBlock() : void +RtcpReportReceptionBlock(buildfrom : uint8_t *, len : int) : v... +set_rbssrc(ssrc : unsigned int) : void +get_rbssrc() : unsigned int +set_flost(flost : unsigned int) : void +get_flost() : unsigned int +set_cumlost(dlost : int) : void +get_cumlost() : int +setSeqHighCycle(seqHC : uint16_t) : void +getSeqHighCycle() : uint16_t +set_seqhigh(seqhigh : uint16_t) : void +get_seqhigh() : uint16_t +set_jitter(jitter : unsigned int) : void +get_jitter() : unsigned int +set_sr(sr : unsigned int) : void +get_sr() : unsigned int +set_dlsr(dlsr : unsigned int) : void +get_dlsr() : unsigned int +size() : int +get_bytes() : uint8_t *

<b>RtcpReportSenderInfo</b>
-ntp_msw : uint32_t -ntp_lsw : uint32_t -rtp_ts : uint32_t -sender_pcount : uint32_t -sender_ocount : uint32_t
+RtcpReportSenderInfo() : void +RtcpReportSenderInfo(timestamp : uint32_t, sender_pcount : uint32_t, se... +RtcpReportSenderInfo(buildfrom : uint8_t *, len : int) : void +set_ntp_tsmw(msw : uint32_t) : void +get_ntp_timestamp_msw() : uint32_t +set_ntp_tslsw(lsw : uint32_t) : void +get_ntp_timestamp_lsw() : uint32_t +set_rtp_ts(rts : uint32_t) : void +get_rtp_ts() : uint32_t +set_packet_count(pc : uint32_t) : void +get_packet_count() : uint32_t +set_octet_count(oc : uint32_t) : void +get_octet_count() : uint32_t +size() : int +get_bytes() : uint8_t *

<b>RtcpReportSR</b>
-sender_ssrc : uint32_t -sender_info : RtcpReportSenderInfo -reception_blocks : std::vector<RtcpReportReceptionBlock*>
+RtcpReportSR() : void +RtcpReportSR(senderInfo : const RtcpReportSenderInfo &) : void +RtcpReportSR(build_from : uint8_t *, max_length : int) : void +~RtcpReportSR() : void +set_sender_ssrc(ssrc : uint32_t) : void +get_sender_ssrc() : uint32_t +getMemObjectType() : std::string +setSenderInfo(senderInfo : const RtcpReportSenderInfo &) : void +get_sender_info() : RtcpReportSenderInfo & +get_reception_block(index : int) : RtcpReportReceptionBlock * +add_reception_block(block : RtcpReportReceptionBlock *) : void +get_bytes() : uint8_t * +size() : int

<b>RtcpPacket</b>
-reports : std::vector<RtcpReport*>
+RtcpPacket() : void +RtcpPacket(buildfrom : uint8_t *, max_length : int) : void +~RtcpPacket() : void +size() : int +get_bytes() : uint8_t * +add_report(report : RtcpReport *) : void +get_reports() : std::vector<RtcpReport*> +readPacket(socket : UDPSocket &, timeout : int) : MRef<RtcpPacket*> +send_to(udp_sock : UDPSocket &, to_addr : IP Address &, port : int) : void

<b>RtcpReceiver</b>
-rtcpSock : MRef<UDPSocket*> -localPort : uint16_t -kill : bool -thread : Thread * -sStt : MRef<StreamStatistics*>
+RtcpReceiver() : void +RtcpReceiver(socket : MRef<UDPSocket*>) : void +RtcpReceiver(socket : MRef<UDPSocket*>, sStt : MRef<StreamStatistics*>) : void +~RtcpReceiver() : void +setSStatistic(ss : MRef<StreamStatistics*>) : void +start() : void +run() : void +stop() : void



<b>RtcpSender</b>
<pre> -rtcpSock : MRef&lt;UDP Socket*&gt; -remoteAddress : IP Address * -localPort : uint16_t -remotePort : uint16_t -ssrc : uint32_t -rtcpSRManager : RtcpSRManager -sStt : MRef&lt;StreamStatistics*&gt; </pre>
<pre> +RtcpSender(localPort : uint16_t) : void +RtcpSender(localPort : uint16_t, sStt : MRef&lt;StreamStatistics*&gt;) : v... +~RtcpSender() : void +setRemoteAddress(ra : IP Address *) : void +getLocalPort() : uint16_t +setRemotePort(port : uint16_t) : void +getRemotePort() : uint16_t +getSocket() : MRef&lt;UDP Socket*&gt; +setSSRC(ssrc : uint32_t) : void +getSsrc() : uint32_t +getRtcpInterval() : uint32_t +getRtcpSRManager() : RtcpSRManager &amp; +sendRtcpPacket(ntp1 : uint32_t, ntp2 : uint32_t) : void </pre>

<b>StreamStatistics</b>
<pre> -ssrc : uint32_t -sPacketCount : uint32_t -soctetCount : uint32_t -rPacketCount : uint32_t -roctetCount : uint32_t -lastRtpTs : uint32_t -ssrc_n : uint32_t -fractionLoss : uint8_t -lossUpdated : bool -sNtpMSW : uint32_t -sNtpLSW : uint32_t -cm : CodecManager </pre>
<pre> +StreamStatistics() : void +~StreamStatistics() : void +incSPacketCount() : void +incSOctetCount(packetSize : uint32_t) : void +incRPacketCount() : void +incROctetCount(packetSize : uint32_t) : void +setSSRC(ssrc : uint32_t) : void +getSSRC() : uint32_t +getSPacketCount() : uint32_t +getSOctetCount() : uint32_t +setLastRtpTs(lrts : uint32_t) : void +getLastRtpTs() : uint32_t +setSSRCn(ssrc_n : uint32_t) : void +getSSRCn() : uint32_t +setFractionLoss(floss : uint8_t) : void +getFractionLoss() : uint8_t +getMemObjectType() : std::string +isLossUpdated() : bool +setLossUpdated() : void +resetLossUpdated() : void +getCodecManager() : CodecManager &amp; +setSNtpMSW(ntpMSW : uint32_t) : void +getSNtpMSW() : uint32_t +setSNtpLSW(ntpLSW : uint32_t) : void +getSNtpLSW() : uint32_t </pre>

<b>MediaStreamReceiver</b>
<pre> #codecList : std::list&lt;MRef&lt;Codec*&gt; &gt; #rtPReceiver : MRef&lt;RtpReceiver*&gt; #id : uint32_t #ipProvider : MRef&lt;IpProvider*&gt; #externalPort : uint16_t #ssrcList : std::list&lt;uint32_t&gt; #ssrcListLock : Mutex #running : bool  +MediaStreamReceiver(media : MRef&lt;Media*&gt;, rtPReceiver : MRef&lt;RtpReceiver*&gt;, ipProvider : MRef&lt;IpProvider*&gt;) : ... +getDebugString() : std::string +getMemObjectType() : std::string +start() : void +stop() : void +getPort() : uint16_t +handleRtpPacket(packet : MRef&lt;SRtpPacket*&gt;) : void +getId() : uint32_t +getAvailableCodecs() : std::list&lt;MRef&lt;Codec*&gt; &gt; +getSsrcList() : std::list&lt;uint32_t&gt; +setSStatistic(ss : MRef&lt;StreamStatistics*&gt;) : void +getSStatistic() : MRef&lt;StreamStatistics*&gt; #gotSsrc(ssrc : uint32_t) : void </pre>

<b>MediaStreamSender</b>
<pre> -rtcpReceiver : MRef&lt;RtcpReceiver*&gt; -ssrc : uint32_t -senderSock : MRef&lt;UDPSocket*&gt; -remotePort : uint16_t -seqNo : uint16_t -lastTs : uint32_t -remoteAddress : IP Address * -senderLock : Mutex -payloadType : uint8_t -selectedCodec : MRef&lt;CodecState*&gt; -muted : bool -muteCounter : uint32_t  +MediaStreamSender(media : MRef&lt;Media*&gt;, senderSock : MRef&lt;UDPSocket*&gt;) : ... +getDebugString() : std::string +getMemObjectType() : std::string +getSelectedCodec() : MRef&lt;CodecState*&gt; +start() : void +stop() : void +setPort(port : uint16_t) : void +getPort() : uint16_t +send(data : byte_t *, length : uint32_t, ts : uint32_t *, marker : bool, dtmf : bool) : void +setRemoteAddress(remoteAddress : IP Address *) : void +setMuted(mute : bool) : void +isMuted() : bool +muteKeepAlive(max : uint32_t) : bool +matches(m : MRef&lt;SdpHeaderM*&gt;, formatIndex : uint32_t) : bool +getSsrc() : uint32_t +increaseLastTs() : void +getLastTs() : uint32_t +setSStatistic(ss : MRef&lt;StreamStatistics*&gt;) : void +getSStatistic() : MRef&lt;StreamStatistics*&gt; +setRtcpReceiver(rtcpReceiver : MRef&lt;RtcpReceiver*&gt;) : void </pre>

<b>RtpReceiver</b>
<pre> -socket : MRef&lt;UDP Socket*&gt; -externalPort : uint16_t -kill : bool -mediaStreams : std::list&lt;MRef&lt;MediaStream Receiver*&gt; &gt; -mediaStreamsLock : Mutex -thread : Thread * -rtcpSender : MRef&lt;RtcpSender*&gt; </pre>
<pre> +RtpReceiver(ipProvider : MRef&lt;IpProvider*&gt;) : void +registerMediaStream(mediaStream : MRef&lt;MediaStreamReceiver*&gt;) : void +unregisterMediaStream(mediaStream : MRef&lt;MediaStreamReceiver*&gt;) : ... +run() : void +getPort() : uint16_t +~RtpReceiver() : void +getSocket() : MRef&lt;UDP Socket*&gt; +setRtcpSender(rtcpSender : MRef&lt;RtcpSender*&gt;) : void +getMemObjectType() : std::string </pre>

## B. Unix Script of NIST Net Configuration

```
echo "Start NIST Net";  
sleep 30;
```

```
cnistnet -a b21 a22 --drop 5 -- up;  
ping3 -c 1 a2;  
echo "packet loss 5%, 60 seconds";  
sleep 60;
```

```
cnistnet -a b2 a2 --drop 0 -- up;  
ping -c 1 a2;  
echo "packet loss 0, 60 seconds";  
sleep 60;
```

```
cnistnet -a b2 a2 --drop 6 -- up;  
ping -c 1 a2;  
echo "packet loss 6%, 20 seconds";  
sleep 20;
```

```
cnistnet -a b2 a2 --drop 1 -- up;  
ping -c 1 a2;  
echo "packet loss 1%, 30 seconds";  
sleep 30;
```

```
cnistnet -a b2 a2 --drop 5 -- up;  
ping -c 1 a2;  
echo "packet loss 5%, 30 seconds";  
sleep 30;
```

```
cnistnet -a b2 a2 --drop 0 -- up;  
ping -c 1 a2;  
echo "packet loss 0, 60 seconds";  
sleep 60;
```

```
echo "Stop NIST Net"  
cnistnet -r b2, a2;2
```

---

<sup>1</sup>. b2 is the IP address of Machine A

<sup>2</sup>. a2 is the IP address of Machine B

<sup>3</sup>. ping command is used to inform a change of NIST NET configuration

