TOWSON UNIVERSITY

COLLEGE OF GRADUATE STUDIES AND RESEARCH

VoIP ON A BARE PC

BY

GHOLAM HOSSIEN KHAKSARI

A DISSERTATION PRESENTED TO THE FACULTY OF

TOWSON UNIVERSITY

IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE

DOCTOR OF SCIENCE

IN APPLIED INFORMATION TECHNOLOGY

MAY 2007

TOWSON UNIVERSITY
TOWSON, MARYLAND 21252

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to all those who have supported me to complete this dissertation. I am deeply indebted to my research committee Dr. Alexander Wijesinha (chair), Dr. Ramesh K. Karne, Dr. Yeong-Tae Song, Dr. Yanggon Kim, and Dr. Sungchul Hong for supporting this research. I am especially thankful to Dr. Wijesinha and Dr. Karne for all the support and advice I have received throughout this endeavor. I also would like to thank Dr. Ali Behforooz, executive director of the Center for Applied Information Technology, and Dr. Chao Lu, chair of Department of Computer and Information Sciences at Towson University, for facilitating this work. I am also obliged to my daughter Mitra for helping with final editing of this document.

TOWSON UNIVERSITY

COLLEGE OF GRADUATE STUDIES AND RESEARCH

DISSERTATION APPROVAL PAGE

This is to certify that the dissertation prepared by Gholam Hossien Khaksari, entitled "VoIP on a Bare PC" has been approved by this committee as satisfactory completion of the requirement for the degree of Doctor of Science in Applied Information Technology**.**

Committee Chair, Dr. Alexander Wijesinha	Date
Committee Member, Dr. Ramesh K. Karne	Date
Committee Member, Dr. Yeong-Tae Song	Date
Committee Member, Dr. Yanggon Kim	Date
Committee Member, Dr. Sungchul Hong	Date
Dean, College of Graduate Studies and Research Dr. Jin K. Gong	Date

ABSTRACT

VoIP ON A BARE PC

GHOLAM HOSSIEN KHAKSARI

This dissertation proposes a novel VoIP softphone architecture for a bare Intel-386 (or above) based PC without an operating system. First, we provide an overview of bare PC computing and note the advantages of a bare PC softphone including its inherent simplicity and ability to provide secure, reliable and efficient voice communication. Next, we discuss the design of a bare PC softphone and describe its architecture and implementation. We then present performance measurements from LAN and Internet experiments, which consider delay, jitter, packet loss, and MOS. They indicate that a bare PC softphone has less jitter, less security overhead, and is able to sustain larger voice packet sizes and a heavier load than a WinRTP softphone while maintaining acceptable call quality with or without background traffic. A bare PC softphone also has acceptable call quality when running Voice over Ethernet (voice packets with Ethernet headers only) on a LAN.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1. Introduction

In this chapter, we give an overview of this research, including the problem statement, goals and motivation. We include some general background on VoIP and bare PC computing. We also note the advantages of a bare PC softphone and briefly discuss related work.

1.1. VoIP - Voice over Internet Protocol

Voice over Internet Protocol (VoIP) is a technology that uses the Internet infrastructure and protocols for voice communication. VoIP is sometimes called Internet telephony, IP telephony, or Voice over the Internet. The concept of VoIP originated in the early 1970s when the idea and the technology were developed. Despite this history, VoIP did not establish a commercial niche until mid-1990s. VoIP technology finally became a viable alternative to PSTN due to the commercialization and mass-market use of the Internet, invention of the Web, and massive investments in IP networking infrastructures by business, vendors, and carriers. Advantages of VoIP technology include:

1. Low cost of voice transmission due to utilization of existing IP networks
2. Desire to integrate voice and data communication into a single network resulting in low operational and maintenance cost
3. Inability of legacy PSTN to provide new voice services demanded by today's businesses

Figure 1 shows how operating System (OS) based softphone applications like WinRTP and OS-less bare PC softphone applications could be used in the future by using

SIP (Session Initiation Protocol) [37] to set up calls in a VoIP network. Phones register with a SIP server and use it to locate other VoIP phones. The SIP server implements call connection and termination between phones to emulate some of the call processing features present in the Signaling System 7 (SS7) used by the PSTN. A special PSTN gateway would enable the IP network to interface with the legacy plain old telephone system [1]. We do not consider SIP servers or PSTN gateways further in this dissertation as this research focuses on the design, implementation and performance of an optimized bare PC softphone that directly communicates with a peer bare PC or conventional softphone.

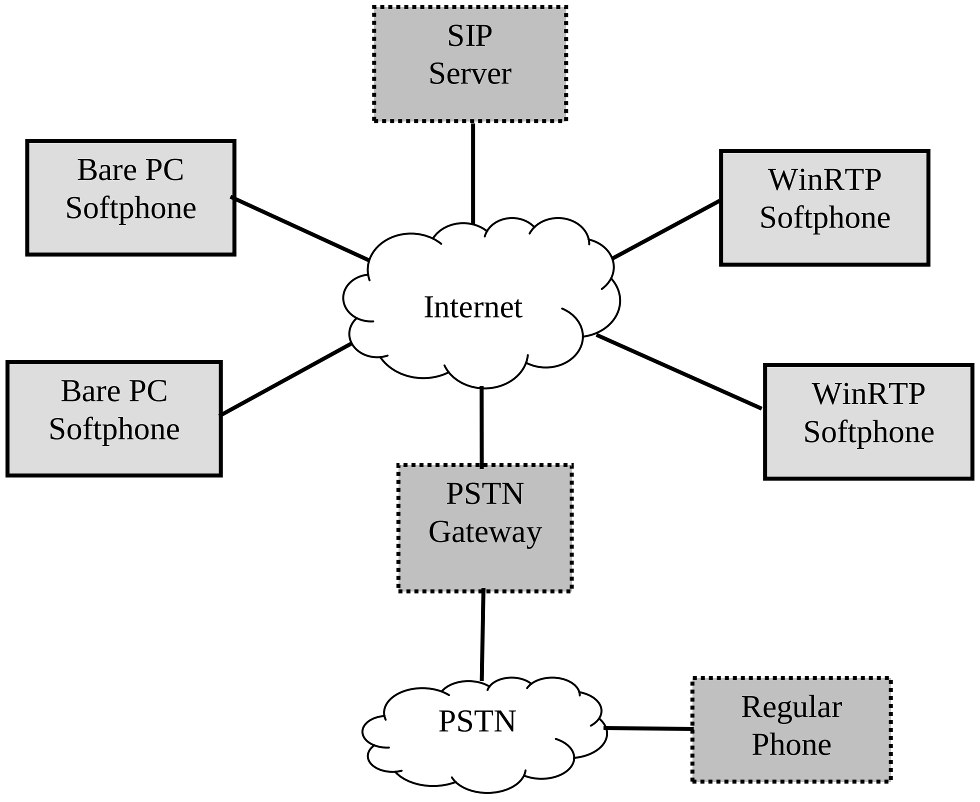

Figure 1: Bare PC Softphone Network Configuration

Softphone applications enable a PC to be used as a VoIP phone. Typically, a softphone inputs digitized voice from a microphone, encodes voice into network packets, and transmits packets over the IP network. The softphone also inputs voice packets from the IP network, decodes each voice packet and outputs the voice payload to speakers. As voice packets travel over the IP network infrastructure from their source to their destination, due to the nature of the IP packet switched network, there is no guarantee that voice packets will reached their destination in a timely manner or that all voice packets will arrive at their destination. Moreover, voice packets may not arrive in order, and may not arrive uniformly spaced out in time. The voice quality for a softphone is usually estimated by measuring the following parameters [2] [9]:

1. **Delay:** time required a talker's voice to reach a listener's ear. It includes time for microphone recording, packet encoding, network transit, packet decoding, and playback on speakers
2. **Jitter:** voice packet delay variation due to the network
3. **Packet loss:** percentage of voice packets lost by the network
4. **MOS:** a measure used for assessment of VoIP call quality that is assigned by a listener or computed automatically

In addition to the challenges of guaranteeing call quality and reliability for VoIP services, security is also a concern. Thus, although the popularity of VoIP continues to grow and major telecommunications companies use IP networks to carry digitized voice as it is convenient and costs less, VoIP has not yet gained widespread acceptance as a replacement for the standard PSTN service.

1.2. Bare PC Computing

An OS manages resources on a computer and interfaces between the users and the hardware. While considerable advancements in improving the versatility of an OS have been made, these benefits have come at the cost of increased size, higher complexity, added overhead, and security flaws that have been exploited by attackers. Critics claim that today's OSs are plagued by buggy code and device drivers making them insecure and unreliable [3]. While new generations of OSs are attempting to address these problems, there is still a need for OS-less application-centric systems that can take full advantage of the underlying hardware resources. Such systems can be more efficient while being inherently less complex due to their reduced size and because they are limited to providing only essential services and functionality. The idea of eliminating OS inefficiencies and abstractions is not new [4].

It is possible to build systems that provide applications with direct interfaces to the bare hardware, thus obviating the need for a conventional operating system. Such applications manage themselves and have complete control of the hardware. An application-centric bare PC system is efficient and easier to secure, as it is less complex. Additionally, a bare PC is convenient for experimenting with novel techniques for improving the performance of applications and protocols since there are no inherent limitations due to an operating system.

In bare PC computing, which is based on the dispersed operating system computing (DOSC) paradigm [5], a computer application contains its own operating environment, thus avoiding operating system middleware. Figure 2 illustrates this approach. Conventional computing layers are mapped into an application object, and the application object runs directly over the hardware with no intervening software or firmware components. The hardware in this case is any bare device such as a PC with an Intel 386 (or above) based architecture.

An application object contains both its application program and its necessary operating environment [6] [7]. Therefore, it is self-contained, self-managed, and self-executed; this enables it to run on any hardware, provided it is compiled for the relevant hardware architecture. Application objects self-manage the CPU, memory, interrupts, and I/O. An application object is self-executed as well, since it manages its loading, execution and termination phases. It may also contain temporal information and security mechanisms. The application object interfaces that enable applications to run on the bare hardware [8] may be part of an application operating environment or implemented in hardware. In this research, we assume that these interfaces are in an application operating environment (i.e., existing hardware is used).

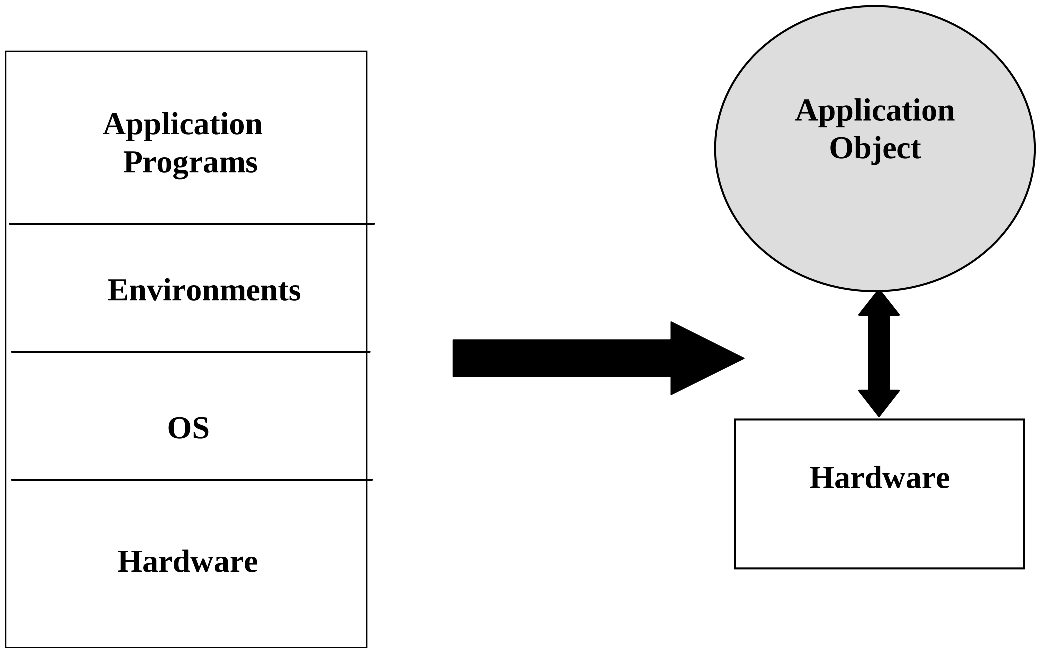

Figure 2: Bare PC Computing

A bare PC is different from an embedded system [10], because a bare PC is capable of running any application and has no OS. The only element of OS functionality in a bare PC is intrinsic to its application and determined by the application (i.e., a bare PC is truly bare outside of its application and an application only includes elements of the operating environment that are essential for its execution). Embedded systems run on some type of operating system and do not provide open interfaces to run applications on the bare hardware. For example, in an embedded system such as a cell phone, no other applications can directly run on the hardware since there are no external interfaces to it.

A bare PC is also different from a system that provides a virtual machine interface enabling software to execute with or without an operating system [11]. A virtual machine interface limits the capabilities of applications running on it and introduces an additional layer that may hinder performance. In contrast, an Intel 386-based bare PC can be used to run any application object, which has direct and full access to the underlying hardware. The bare PC approach eliminates the OS altogether, and it goes further than earlier work on minimal OSs including Tiny OS [21], Exokernel [22], and OSKit [23].

To support bare PC applications, a C++ API allowing applications to run directly on Intel 386 (or above) based PCs have been previously developed [8]. Interfaces to memory, CPU, timer, interrupts, tasks, keyboard, display, floppy drive, audio card and Ethernet card are written in Microsoft assembler (MASM). Several network protocols for a bare PC including ARP, IPv4, UDP, TCP, SMTP, lean FTP, and HTTP have also been implemented. Several applications that run on a bare PC have been developed previously. For example, in [12], a bare PC Web server that runs on any Intel 386-based architecture with no operating system, hard disk, or other supporting software is described, and results comparing its performance to the Windows IIS and Apache Web servers are presented.

1.3. Problem Statement and Research Goal

This research focuses on designing and implementing an optimized VOIP softphone that runs on a bare PC and investigates its ability to provide efficient, reliable and secure communication. Many VoIP systems, including systems for peer-to-peer voice, have been discussed in the literature [13], [14], [15], [16], [17], [18], although none run on a bare PC. Security issues for VoIP systems are highlighted in [19].

Existing VoIP systems are dependent on OS services. For example, both the Skype softphone [13] and the user agents for the peer-to-peer VoIP adaptor [20] require Windows, Linux, or some specialized OS. In view of OS complexity and overhead, it is difficult to fully optimize these VoIP phones to improve performance and call quality. They also inherit security weaknesses that result from OS vulnerabilities and their complexity.

The goal of this research is to design and implement a softphone that runs on any Intel-386 (and above) based bare PC with no OS, and to investigate the effect of several optimizations on call quality. This work is part of an effort to develop bare PC applications including personal Web servers, email clients, and SIP clients and servers together with supporting security protocols. The contributions made by this research provide insight into the construction and optimization of secure softphone applications that can run without an OS.

1.4. Motivation

Bare PC computing [5], which is an alternative approach for general purpose computing, eliminates the OS allowing the programmer direct access to (and complete control over) the underlying hardware. Therefore, it is very convenient for testing and evaluating VoIP optimizations that require changes to low-level system elements such as device drivers, the CPU task scheduler, or the networking subsystem. Furthermore, bare PC computing has many advantages for VoIP including performance benefits due to its low overhead, as well as simplicity, and the likelihood of being more secure due to elimination of the OS and unnecessary services. For example, when a bare PC softphone is connected to the Internet, the only open ports are those required by RTP and they would be the only means for a prospective attacker to send packets that would even be accepted by the softphone.

In bare PC systems, the absence of an OS enables us to investigate novel techniques for the design and construction of optimized softphone applications. The following are some of the advantages of the bare PC softphone application:

- Small code size
- No OS
- No hard disk
- C++ code only
- Runs on any bare Intel-386 (or above) based PC
- Robust and efficient
- Interoperates with an OS-based WinRTP softphone
- Standard PCM codec

- Can be integrated with other bare PC applications such as a personal web server and email client
- Customized security features
- Excellent voice quality
- Can be deployed in any LAN or the Internet
- Can run directly over the Ethernet protocol in pure switched (routerless) LAN environments

1.5. Related Work

To the best of our knowledge, there are no VoIP softphone systems that run on a bare PC. Even if a softphone runs on Exokernel [22] (we are not aware of any), the latter would not give full control to the application as it is still a form of OS. Siscophone [14] uses several optimizations similar to those implemented on the bare PC softphone to improve call quality (such as minimal data copying). However, its design is constrained by the underlying OS. For example, it does not include an optimal task scheduling technique. The peer-to-peer VoIP architecture in [20] has many desirable features. The associated SIP adaptor works with existing SIP phones, is capable of supporting seamless addition of new services such as conferencing and voice mail, and is essentially plug-and-play. However, it may be harder to optimize such a system for performance since the user agents that are required to use the adaptor rely on an OS. Another SIP-based VoIP architecture that also offers mobility support is discussed in [26], and a VoIP architecture based on Java and Web technologies is presented in [27]. Again, such architectures include user agents that run on a conventional OS, and the overall performance of the system are therefore bound by OS limitations. In contrast, a bare PC softphone may be fully optimized since the AO programmer has total control of the system and its hardware resources [25].

There have been many attempts to improve call quality in VoIP systems. In [28], different paths through the network are used in order to improve call quality. Playout buffer algorithms that incorporate jitter and packet loss compensation are given in [29]. In [30], Skype and MSN VoIP systems are compared with respect to throughput, packet inter-arrival statistics, and MOS (mean opinion score). Finally, in [31], MOS ratings are used to evaluate effects of bursty packet loss on call quality, and a method to maximize call quality by optimizing the packet interval is proposed. Since a bare PC has no OS, these and other techniques to improve voice quality can be added to the bare PC softphone with less intrinsic overhead and better performance than a conventional system. In essence, the main difference between existing softphones and a bare PC softphone is that the latter runs directly on the hardware with no OS, and is therefore simpler to optimize and control.

VoIP security issues are discussed in detail in [43]. VoIP applications do not use TLS/SSL [40] for security because it has considerable overhead and is not designed for a real-time application such as voice. Moreover, TLS/SSL would need to be modified to run over UDP instead of TCP. Some security approaches for VoIP attempt to take advantage of the nature of voice conversations. For example, ZRTP [42] relies on an authentication procedure that assumes a user is able to match the peer's voice during the conversation and authentication phases. A possible future standard for secure VoIP will be Secure RTP [44]. However, both ZRTP and SRTP are vulnerable to certain attacks [43]. VoIP security continues to be an ongoing subject of research. The bare PC VoIP security mechanisms we describe are based on public key cryptography and assumes that the participants are in possession of (and trust) each other's public keys.

The rest of this dissertation is organized as follows. In Chapter 2, we discuss bare PC softphone design and in Chapter 3, we describe its architecture and implementation. In Chapter 4, we present performance measurements and in Chapter 5, we give the conclusion.

Chapter 2. Softphone Design

In this chapter, we present an overview of bare PC softphone design. We also discuss softphone requirements and optimizations.

2.1. Softphone Requirements

The general requirement is to design, implement, and test a peer-to-peer VoIP softphone application that can run on a bare Intel-386 (or above) based PC without an operating system, while providing acceptable voice quality. It must be able to record, compress and transmit voice packets over the IP network, as well as receive, decompress, and playback voice packets from the IP network. Specific softphone requirements are:

1. Manage the microphone recording and speaker playback processes
2. Use a PCM codec to compress and decompress voice data
3. Use the lean bare network protocols to send and receive voice packets
4. Use a jitter buffer to adapt to varying Internet traffic conditions
5. Communicate with an OS-based softphone
6. Operate on a LAN or the Internet while maintaining acceptable voice quality

2.2. Microphone Recording and Speaker Playback

The device driver software for the onboard audio codec, as part of the DOSC environment, provides a set of API calls and circular buffers for communicating with the microphone and speaker. The device driver software for the Network Interface Card (NIC), also part of the DOSC environment, provides a set of API calls and circular send and receive buffers for communicating with NIC as well. The microphone and speaker buffers and the NIC send and receive buffers are DMA mapped in the internal memory.

The first task is to manage the microphone recording process and to read the recorded data in real-time with no loss. Two different methods that can be used to solve this problem are interrupts and polling. In the interrupts method, the audio card is configured to interrupt softphone processing upon completion of some microphone data recording, so that the softphone application can process the recorded data. The polling method configures the audio card to continue recording without interrupting the audio softphone application. Instead, softphone is responsible for polling the audio card for completion of recording so that it can process the recorded data. The polling method provides good performance, is convenient to use on a bare PC and has been successfully used by other bare PC applications. We therefore use this method in the bare PC softphone as well.

The recorded microphone data is continuously transferred to the microphone circular buffer located in the main memory at the PCM rate using the onboard codec internal clock. The bare PC softphone uses the PC internal timer to poll for the arrival of recorded data in the microphone circular buffer and for management of the read and write pointers. It is necessary to coordinate data recording and buffer pointer manipulation to maintain call quality.

The second task is to move recorded data from microphone circular buffer directly to the NIC circular send buffer without any intermediate copying of data. Both the microphone circular buffer and the NIC circular send buffers are resident in audio softphone address space; hence, direct copying of data between two circular buffers is feasible. A DOSC application like the softphone is a single monolithic executable that executes in user mode and in a single address space with complete control of system resources. When a voice packet is recorded, it is compressed and directly copied into the NIC circular send buffer, while a pointer to voice data in the NIC circular buffer is used for all processing steps. Zero copy buffering (see below) is thus achieved during recording.

The softphone also processes voice payloads arriving from the IP network and plays them back in real time on the speakers. The softphone has to manage the NIC operations and the NIC circular receive buffer in real-time with no loss of received data. The polling method can be used to manage the NIC receive buffer. When a voice packet arrives from the IP network, it is stored in the NIC circular receive buffer, and the pointer to the packet can be used for all processing steps. The voice payload is then decompressed and copied to the speaker circular buffer directly. This enables zero copy buffering during play back as well.

The softphone manages the speaker playback process and the speaker circular buffer. The onboard codec supplies the speakers with voice data at the PCM rate from its internal queue using its internal clock. The codec driver software must replenish the codec queue with data stored in the speaker buffer. Received voice packets from the IP network are continuously transferred from the NIC receive buffer to the speaker circular buffer located in main memory. The softphone uses the system timer to poll and manage the speaker playback process and the speaker circular buffer. The bare PC softphone uses the PC’s internal timer to poll for arrival of network packets in the NIC circular receive buffer, and for management of the read pointer. The details of microphone buffer and speaker buffer management are covered in the next chapter.

2.3. Voice Compression and Decompression

Network bandwidth is a limited shared resource and VoIP traffic is delay sensitive. Heavy IP network traffic causes excessive network delay and results in packet loss that negatively impacts VoIP call quality. To optimize network bandwidth usage, voice packets can be compressed before transmission and decompressed after they are received from the IP network. The compression and decompression algorithm must not degrade voice quality.

There are many available shareware software voice codecs as well as codec software products. The codec for the PC softphone has to be capable of processing voice data for the onboard audio codec in use. We ported PCM codecs from several free softphones to the bare PC softphone, but found that many of them produce unacceptable voice quality. We ultimately chose the WinRTP G.711 codec for the bare PC softphone since it is easily ported to the bare PC environment and provides good call quality.

2.4. Optimal Task Scheduling

The idea in optimal task scheduling is as follows: when a task is not doing useful work (no CPU processing), it should suspend itself or return control to the main task. Thus, in the softphone, the receive task will return to the main task after processing a received request. Similarly, the audio task will return to the main task after the audio frame pointers are placed in the jitter buffer. Upon activation, each task is allowed unlimited CPU time. When suspending itself, a task requests to be scheduled at a future time by specifying a suspension time. The main task executes a task if and only if only its suspension time as requested by the task itself has expired.

The bare PC tasking strategy has a direct impact on the intrinsic performance parameters of the softphone application. The softphone must process microphone voice data as soon as it becomes available and send it over the IP network; it must also process the incoming network voice packets in order while minimizing the effects of jitter and delay. The tasking strategy must also be efficient, so as to minimize resource usage.

Initially, standard tasking strategies such as first-come-first-serve (FIFO), priority scheduling and round robin were used. Performance measurements indicated that none of these approaches is optimal for a bare PC softphone. Eventually, we decided to use a task scheduling strategy similar to that used in the bare PC Web server. As the bare PC softphone has complete control of task scheduling and execution, it was easy to test the effect of different task scheduling techniques, including novel strategies especially suited to the bare PC, and determine an optimal strategy.

2.5. Recording and Playback Synchronization

The onboard audio codec chip set uses its own internal timer to manage both microphone recording and speaker playback processes, while the softphone application uses the bare PC system timer for management of its own processing. The challenge is to determine the best time to read and process recorded microphone data and to provide data for speaker playback. Since the polling method is used to communicate to the audio codec, it was convenient and efficient to use the audio codec internal timer to synchronize playback and recording.

The bare PC softphone monitors the recording of the microphone voice data. Whenever a full microphone buffer has been recorded, it is processed and the jitter buffer is checked for received packets to be played back on the speakers. Therefore, every time a full microphone buffer is recorded and processed, a packet is played back on the speakers. This technique of synchronizing recording and playback helps to reduce the intrinsic delay by keeping the gap between the read and write pointers for the speaker playback buffer as short as possible.

2.6. Interoperating with WinRTP

The WinRTP softphone tool [24] enables voice to be recorded at the source, transmitted over the IP network and played back at the receiver. It is a Windows COM component tool, which can be modified, compiled, and executed with minimal effort. To enable voice communication between the bare PC softphone and the WinRTP softphone, it is necessary to match the rates at which they playback voice. Initially, it was observed that the bare PC softphone plays back voice at double the normal speed, while the WinRTP plays voice at half the normal speed.

This problem was traced to the AD1981B audio codec used by the bare PC softphone. The bare PC audio codec records in stereo (i.e. two 16-bit PCM samples are recorded for the left channel and right channel), while the WinRTP machine's audio codec records in mono. This problem is solved by eliminating the data for one of the channels on the bare PC side during recording and copying the data into both channels during playback. Interoperability with the WinRTP softphone is not only useful from a practical viewpoint, but also enables us to determine any effects on performance when a bare PC softphone communicates with a conventional softphone.

2.7. Cross-Layer Protocol Design

Layered protocol design is the dominant approach for implementing network protocols with each layer corresponding to a single protocol and communication that is limited to adjacent layers. Although it simplifies protocol design, the layered approach is limiting and inefficient, especially when processing the packet payload. The bare PC softphone uses cross-layer protocol design since it is intrinsic to and facilitated by the bare PC computing environment. Although cross-layer design refers to a technique used in wireless networks wherein physical layer information is passed to the upper layers [48], we use the term in a more general sense to mean the capability to pass information freely between any two networking layers as done in the bare PC.

In particular, we do not follow strict layering rules. For example, all the protocol processing for sending an outgoing packet is accomplished in a single step and implementation of the protocols RTP, UDP and IP in a bare PC are integrated with Ethernet processing. Communication between any two layers in a bare PC is easy since all layers share a single copy of the data due to zero copy buffering (discussed below). For example, when a microphone buffer is recorded, the bare PC softphone copies the voice data directly from the microphone buffer to the NIC send buffer, thereby eliminating unnecessary layer overhead. The RTP, UDP, IP, and Ethernet headers are then directly added to the payload. This approach to designing network protocols enables the bare PC to communicate lower layer information to any upper layer and vice versa.

2.8. Fixed Size Buffers

The onboard audio codec records microphone voice data in fixed size buffers, and also requires fixed size voice buffers for playback on the speakers, as configured during compile time. The softphone application tasks are designed to process fixed size buffers of voice data per activation, before the tasks suspend themselves. This simplifies the processing steps, makes the softphone design simple and less prone to buffer pointer processing problems, and eliminates unnecessary copying of data into temporary local buffers.

Each recorded microphone buffer provides the payload for a single RTP packet for transmission over the IP network, while the payload from each received RTP packet provides the data for one speaker buffer. Microphone and speaker buffers have the same size. By convention, voice packet size is expressed in milliseconds and mapped into a buffer size in bytes.

2.9. Single Address Space

The softphone application is a single monolithic image executing in user-mode. It contains all the necessary code and hardware interfaces. It does not use any system calls or system libraries. In this system, user space and system space are the same, since the AO programmer controls the memory map for the softphone application.

2.10. Zero Copy Buffering

In a conventional system, the user space is virtual and controlled by an operating system. Together, the single address space feature of the bare PC softphone and modern Ethernet cards features allow the bare PC softphone application to keep network packets in user address space. Similarly, the audio codec chip allows microphone and speaker buffers to be in user space as well.

A novel feature in the bare PC architecture is a zero-copy buffering scheme in which incoming packets are stored in the receive UPD (upload pointer descriptor) buffer, and the same pointer is used until packets are played out. Likewise, zero copy buffering is achieved for outgoing packets by using the same pointer for the audio driver and the send DPD (download pointer descriptor) buffer.

2.11. Minimal Resource Dependence

The bare PC environment supports creation of applications with minimal dependency on system resources. The softphone application does not require complex interfaces. The softphone AO manages the CPU and the memory available in the bare PC. The necessary standard interfaces to the hardware elements, as well as drivers for the network cards and sound cards, were developed by other students during previous bare PC projects. In addition, there are C++ interfaces to tasks, interrupts, and exceptions [8].

2.12. Call Quality

Network delay, jitter and packet loss are important parameters affecting VoIP call quality. In addition, the MOS, which is used to measure call quality during a conversation, can be assigned by a group of listeners or calculated by an automated tool. The bare PC softphone is capable of determining values of call quality parameters including packet loss, maximum packet interarrival gap, mean and maximum jitter, network delay and total (end-to-end) delay. We discuss the calculation of these values in the next chapter.

Chapter 3. Architecture and Implementation

In this chapter, we describe the bare PC softphone architecture and implementation. We include details of tasks and task scheduling, protocol header processing, jitter buffer implementation, network and audio buffer management and security mechanisms. The code for the RSA, AES, and SHA-1 algorithms were obtained from the Web and adapted for the bare PC computing environment by removing OS-related calls [50], [51], [52].

3.1. System Architecture

The bare PC softphone application executes on a bare PC in the DOSC computing environment [38]. This environment provides multi-tasking, audio codec and NIC drivers, and limited screen and keyboard I/O functionality as shown in Figure 3.

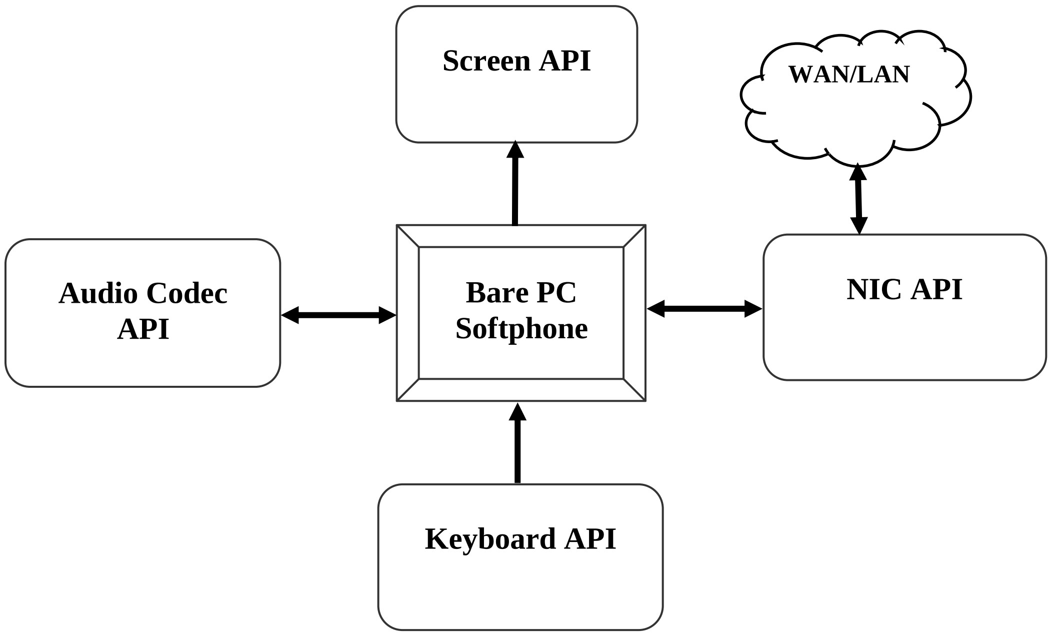

Figure 3: Bare PC Softphone Context Diagram

During recording, the bare PC softphone application inputs digitized microphone voice data from the audio codec, encodes it into packets, and outputs each packet to the NIC for transmission by the network. During playback, the bare PC softphone application inputs incoming network voice packets from the NIC, decodes each voice packet, and outputs the voice payload to the audio codec for playback on the speakers.

The bare PC VoIP softphone inherits many advantages of bare PC (OS-less) computing in general. The basic hardware elements that support the bare PC softphone are CPU, memory, floppy drive, Ethernet NIC (external 3Com 905CX network interface card), onboard audio chip (AD1981B), keyboard, headphones and display. A bare PC provides direct interfaces to the hardware and all other necessary operating elements are built into the application itself. As these interfaces are simple and robust, they are used to support the bare PC VoIP softphone application, as well as other bare PC applications such as an email client and a Web server [12]. These interfaces also enable the softphone to be run concurrently with other applications on the bare PC by adding more tasks.

Figure 4 shows the system elements of the bare PC softphone application. The AD1981B 16-bit PCM codec digitizes analog voice data and the ICH5 DMA controller stores it in the microphone buffer. The bare PC softphone application inputs buffers of digitized voice data from the microphone buffer and compresses each buffer from 16-bit to 8-bit resolution using a G.711 codec, while ignoring data from one of the channels (stereo-to-mono conversion). Next, RTP, UDP, IP, and ETH headers are added to the voice data. Voice packets are stored in the NIC send buffer for transmission on the network.

Conversely, when a voice packet is received from the network, it is stored in the NIC receive buffer. Next, ETH, IP, UDP and RTP headers are checked and removed from each packet, and the RTP[33] header fields plus the pointer to the voice payload are inserted into the jitter buffer. The bare PC softphone application removes each 8-bit PCM compressed voice packet from the receive buffer, passes it through the G.711 codec for decompression to 16-bit PCM resolution, and writes it directly to the speaker buffer left and right channels (mono-to-stereo conversion). The ICH5 [35] DMA controller transfers digitized voice data to the AD1981B codec for playback on the speakers.

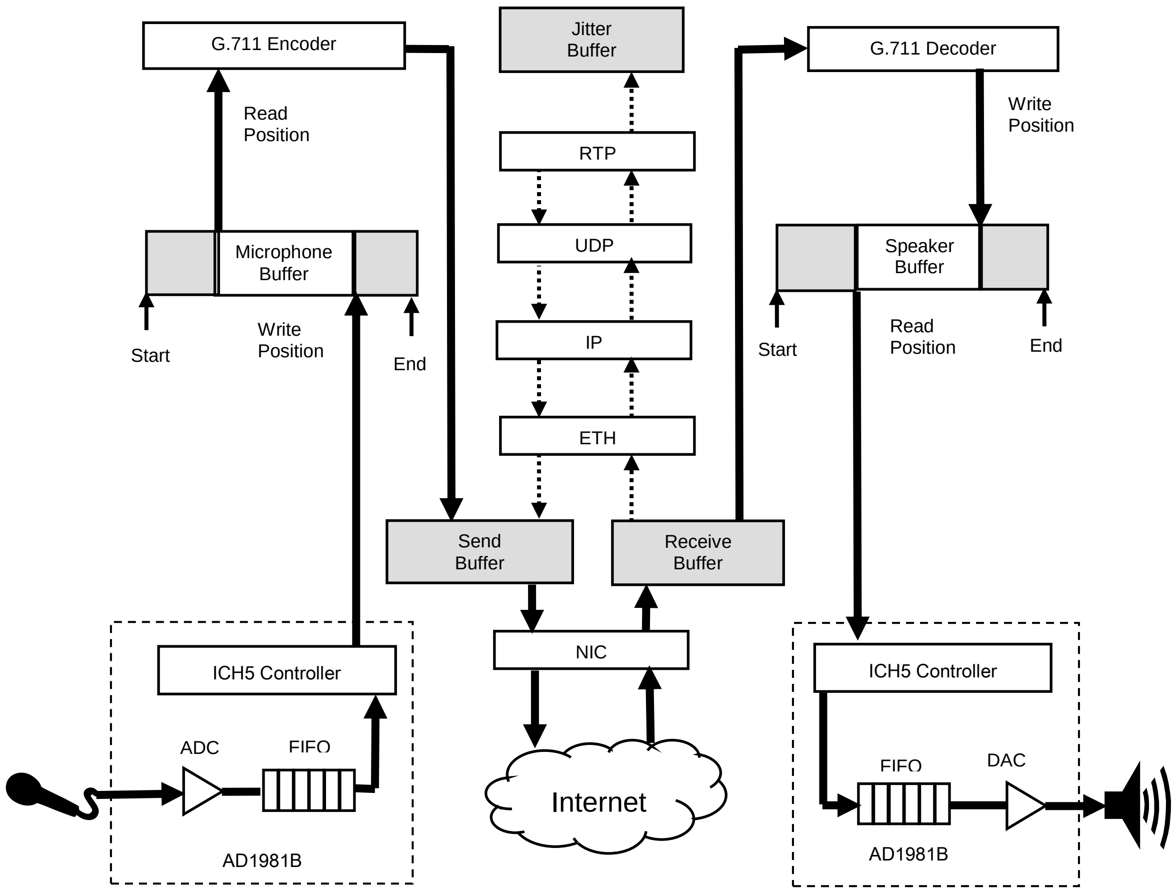

Figure 4: Softphone System Diagram

The above architecture is very simple; it is also customized for the bare PC softphone. There is no functionality implemented in the system that is not relevant to this application. In addition, this architecture enables many novel features (described below) to be incorporated in the softphone. Many of these features are harder to implement in an OS-based system.

3.2. Class Diagram

The softphone application software design is based on object-oriented methodology using a set of interacting objects. This allows for simple reuse of existing bare PC objects, as well as a clean method for porting non-bare PC software to the bare PC environment. Every object maintains its own local state and provides interfaces for communication with the outside world. Object oriented analysis of the softphone application was used to identify the objects; these are briefly described next and shown in Figure 5:

- **G.711:** deals with voice compression and decompression (has been modified and ported to the bare PC environment)
- **Audio Object:** implements the driver for the onboard codec and is concerned with communicating with the onboard codec to manage the microphone recording and speaker playback processes
- **Jitter:** implements the jitter buffer
- **RP:** manages the record and playback functions of the softphone
- **APPTASK:** implements the softphone tasks
- **Hshake:** implements the handshake (over TCP) for exchange of the AES key

- **RTP:** implements the RTP protocol
- **Vsec:** deals with voice security
- **TCP, ARP, UDP, IP, ETH:** implement the respective network protocols
- **RSA:** implements the RSA algorithm (has been ported to the bare PC environment)
- **AES:** implements (AES has been ported to the bare PC environment)

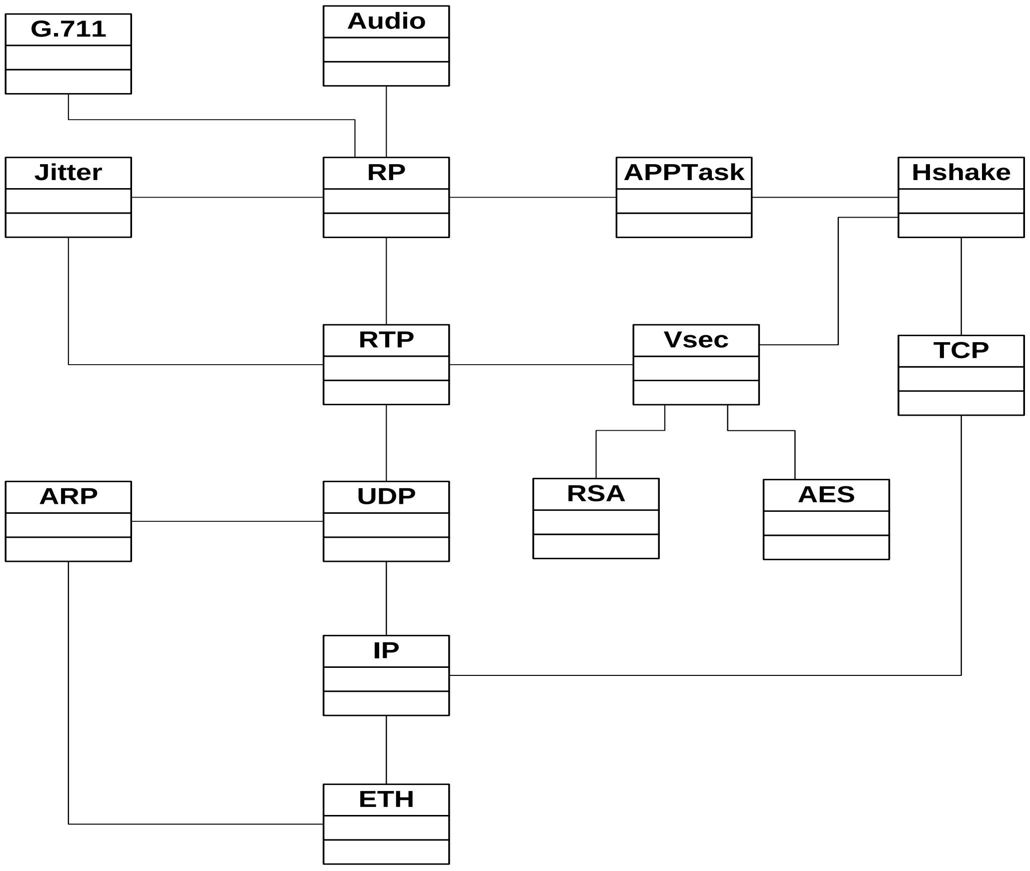

Figure 5: Bare PC Softphone Class Association Diagram

3.3. Task Scheduling

Bare PC softphone task scheduling is designed to be very simple. Yet, it is quite flexible and capable of supporting a powerful multi-tasking environment. The task scheduling strategy is an integral part of the bare PC softphone application, providing the softphone application with complete control of task scheduling and execution. It allows for scheduling of tasks only when processing is required. It also allows any task to suspend itself, and return control back to the main task so that the CPU can be allocated to other useful processing tasks. For example, the receive task will return control back to the main task after it has processed received network packets. Similarly, the audio task will return control back to the main task after it has completed processing the microphone recorded data and the incoming network voice packets.

The bare PC softphone task architecture is shown in Figure 6. First, softphone tasks are created and placed on the idle task list, and then placed on the active task list when processing is required. Upon completion of processing, each task is placed back on the idle task list.

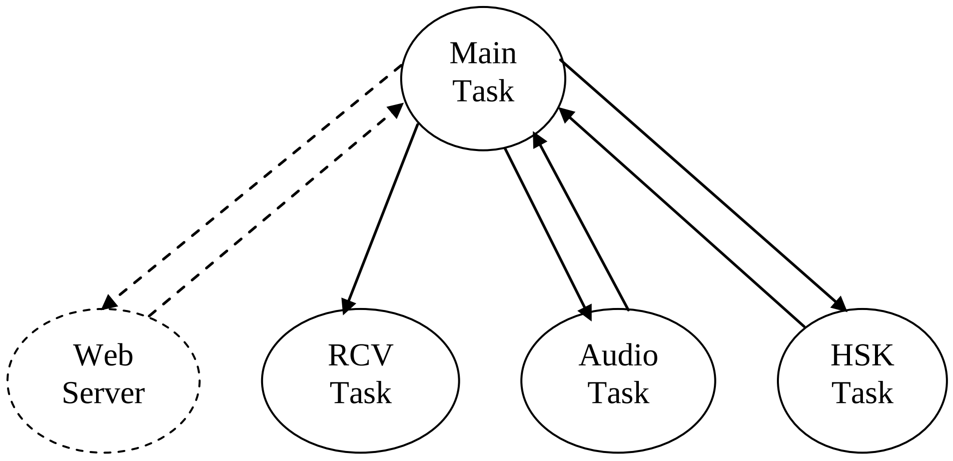

Figure 6: Audio Softphone Task Architecture

As seen in the figure, audio softphone processing is partitioned into the following four separate tasks:

- ***Main Task:*** responsible for management of the bare PC softphone tasks
- ***Receive Task:*** responsible for checking for arrival of new network packets, processing of packet headers, and placement of packet descriptors into the jitter buffer
- ***Audio Task:*** responsible for moving the recorded microphone data from the microphone buffer to the NIC, as well as for moving voice packets from the receive buffer to the speaker buffer
- ***Handshake Task:*** responsible for the establishment of a TCP/IP handshake, exchange of the AES key, and termination of the handshake between two bare PC softphone applications during startup

3.4. Main Task

The main task is responsible for managing the other bare PC softphone tasks (Figure 7). The functions of the main task are activation, scheduling, and execution of other softphone tasks. Once a task is executed, it will complete processing the available data and then suspend itself for a fixed time. This results in a return of control back to the main task.

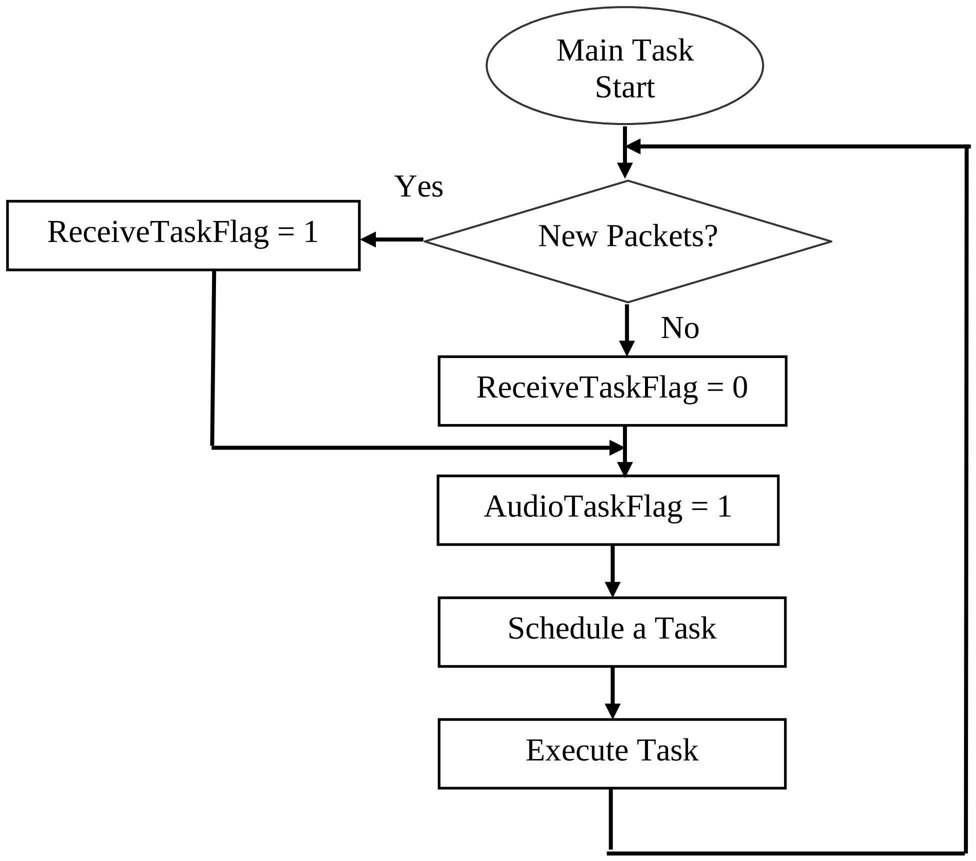

Figure 7: Main Task State Diagram

Figure 8 shows that receive and audio tasks perform the bulk of audio softphone processing.

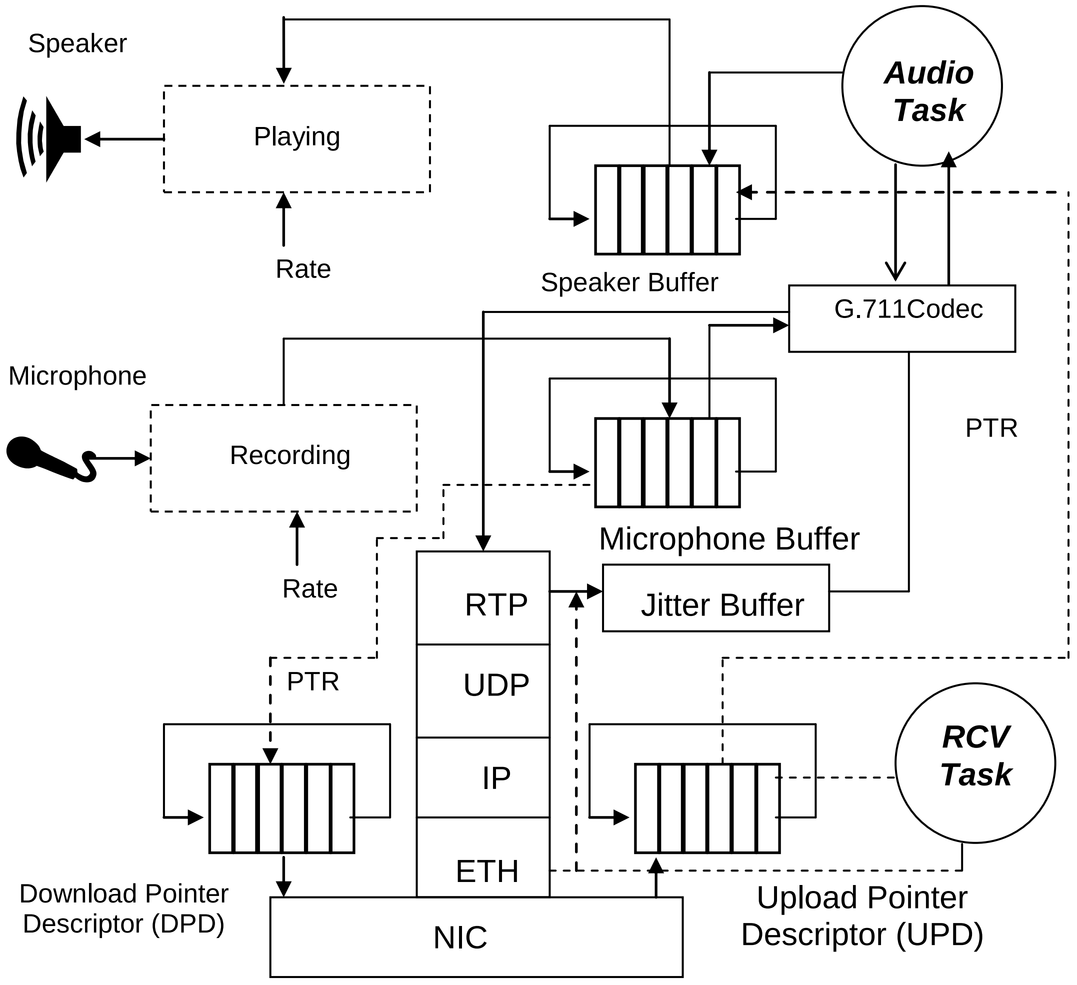

Figure 8: Receive and Audio Tasks Processing

3.5. Receive Task

The NIC card inputs voice packets from the IP network and transfers them to RAM. These are accessible via the UPD list. The Ethernet object validates each packet by examining each field in the ETH header fields and makes packets available for use by the receive task. The receive task checks for arrival of new network packets via the Ethernet object and calls the IP handler. The IP object examines and verifies each IP header field and passes all UDP packets to the UDP handler. The UPD object examines each UDP header field and passes RTP packets to the RTP handler. The RTP handler processes RTP header fields and calls the jitter object to store each voice packet in the jitter buffer. The jitter object receives all RTP header fields associated with each voice packet descriptor and a pointer to the voice payload as stored in the UPD receive buffer (Figure 9). The receive task processing steps will be discussed next.

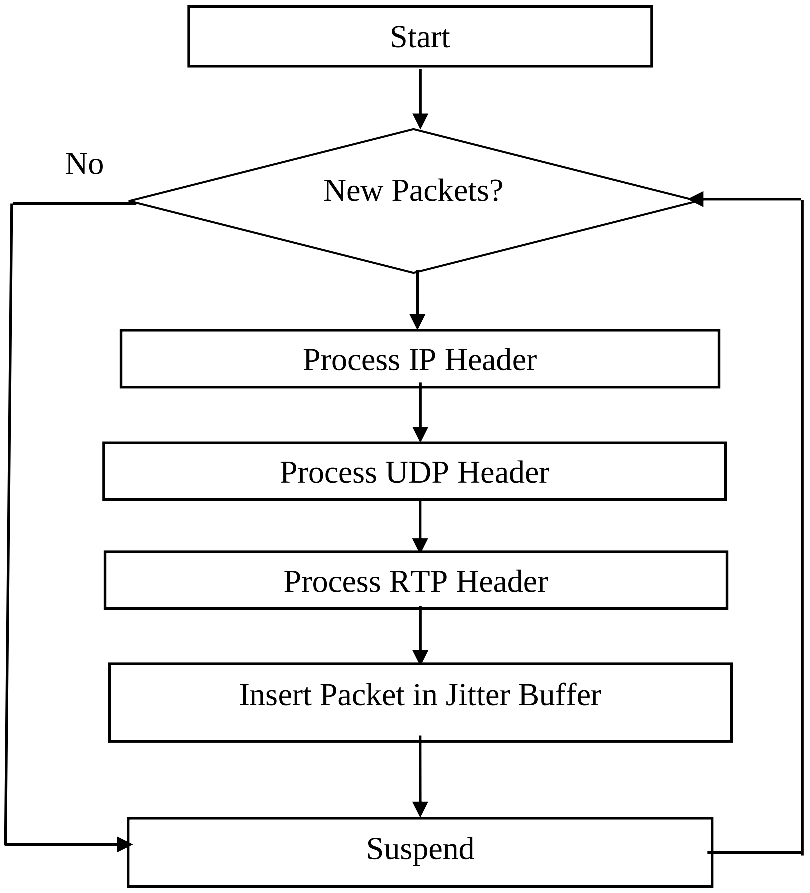

Figure 9: Receive Task State Diagram

3.5.1. UPD Receive Buffer Management

The Ethernet object is responsible for management of the circular UPD receive buffer while working with the NIC driver software. Each element of this buffer corresponds to a single packet of digitized voice data received from the IP network. The sender has prepended each voice packet with header fields for the ETH, IP, UDP, and RTP protocols. The UPD receive buffer located in RAM provides an interface between the NIC DMA controller and the audio softphone application.

3.5.2. ETH Header Processing

The Ethernet object processes each incoming voice packet directly from the UPD receive buffer by examining each ETH header field. The MAC address field of the incoming packet is checked against the MAC of the NIC to ensure that only packets destined for this softphone are accepted. Next, the protocol type field is checked for the IP packet type. All IP packets are accepted and the IP handler receives a pointer to the IP packet stored in the UPD memory.

3.5.3. IP Header Processing

The IP object processes the IP header fields for each IP packet as stored in the UPD receive buffer. If the IP address field matches the IP address for this PC, then the packet is accepted. Next, the UDP handler is called with a pointer to the UDP packet as is stored in the UPD memory.

3.5.4. UDP Header Processing

The UDP object processes and validates UDP header fields for each packet located in the UPD receive buffer. All bytes received are processed in “Little Endian” byte order. UDP header fields are processed as follows:

- **Source Port 16 bits**: This is the port used by the source of UPD packets, i.e. the port that the softphone sending the UDP packet is using for this audio session This field must be checked against the port number of the audio softphone sending the UDP packets, as established at the start of the audio session
- **Destination Port 16 bits**: This is the port by which the softphone receives UDP packets for this audio session. This field must be checked against the port number by which this audio softphone is accepting UDP packets, as established at the start of the audio session.
- **Length 16 bits:** This field contains the size (number of bytes) of the voice payload plus the size of the UDP header field (8 bytes).
- **Checksum 16 bits:** A checksum for the payload is computed and checked against the value stored in the checksum field of the UDP header. If the checksums do not match, the UDP packet is discarded.

3.5.5. RTP Header Processing

RTP is implemented as described in [33]. The RTP object process and validates RTP header fields for each packet located in the UPD receive buffer. Each RTP packet header is processed and checked for validity. All bytes received are processed in “Little Endian” byte order. The RTP header fields are as follows:

- ***Version(V) 2 bits:*** This field must contain a value of 2, which is the current version of the RTP protocol used by the bare PC softphone.
- ***Padding(P) 1 bit:*** This field must contain zero indicating there are no additional padding octets at the end that are not part of the payload.

- ***Extension(X) 1 bit:*** This field must contain zero indicating a fixed RTP header is used.
- ***CRC Count (CC) 4 bits:*** This field must contain a zero indicating no additional data sources is used.
- ***Marker (M) 1 bit:*** This field must contain a zero indicating no tailoring of the RTP header is used.
- ***Payload type (PT) 7 bits:*** This field must contain zero if the source of the RTP packet used PCMU (µ-law) encoding, or 8 if it used PCMA (A-law) encoding.
- ***Sequence Number (SN) 16 bits***: This field is parsed as a 16-bit logical circular number, which designates a sequence number associated with the RTP packet as assigned by the sender. The sequence number starts from zero and is incremented for each subsequent packet (it is reset to zero upon reaching the maximum value).
- ***Time Stamp (TS) 16 bits:*** This field is parsed as a 32-bit logical circular number designating a time stamp that indicates when the packet was generated by the source of the packet.
- ***Synch Source (SSRC) 32 bits:*** This field is parsed as the IP address for the softphone sending the RTP packets. However, this field is not used.

Finally, the RTP object passes the values for RTP header fields and a pointer to the RTP payload (stored in UPD memory) to the jitter object by calling the insert member function from the jitter object.

3.5.6. Jitter Buffer Implementation

As noted above, the last processing step in the receive task is the RTP processing function, which inserts incoming voice packets into the jitter buffer. The jitter object implements the jitter processing functions. It receives voice packets inserted by the RTP object, and queues them for playback on the speakers by the audio task. Audio task processing is discussed later.

The jitter buffer consists of an array of voice payload descriptor elements (Figure 10). Each payload descriptor element has all the fields corresponding to the RTP fields. In addition, each element has a field for the size of the received voice packet, a field that contains a pointer to the actual voice packet (Payload Pointer) as stored in the DPD memory, and a playout time field for each packet. The values stored in each field are received from the RTP handler (except for the playout time field).

The number of entries in the jitter buffer payload descriptor array is set during jitter object initialization. A voice packet description is inserted into the jitter buffer at any open slot as a packet arrives and it is removed for playback using the playback delay time for each packet and the packet sequence number. The playback occurs during audio task execution, which is discussed below.

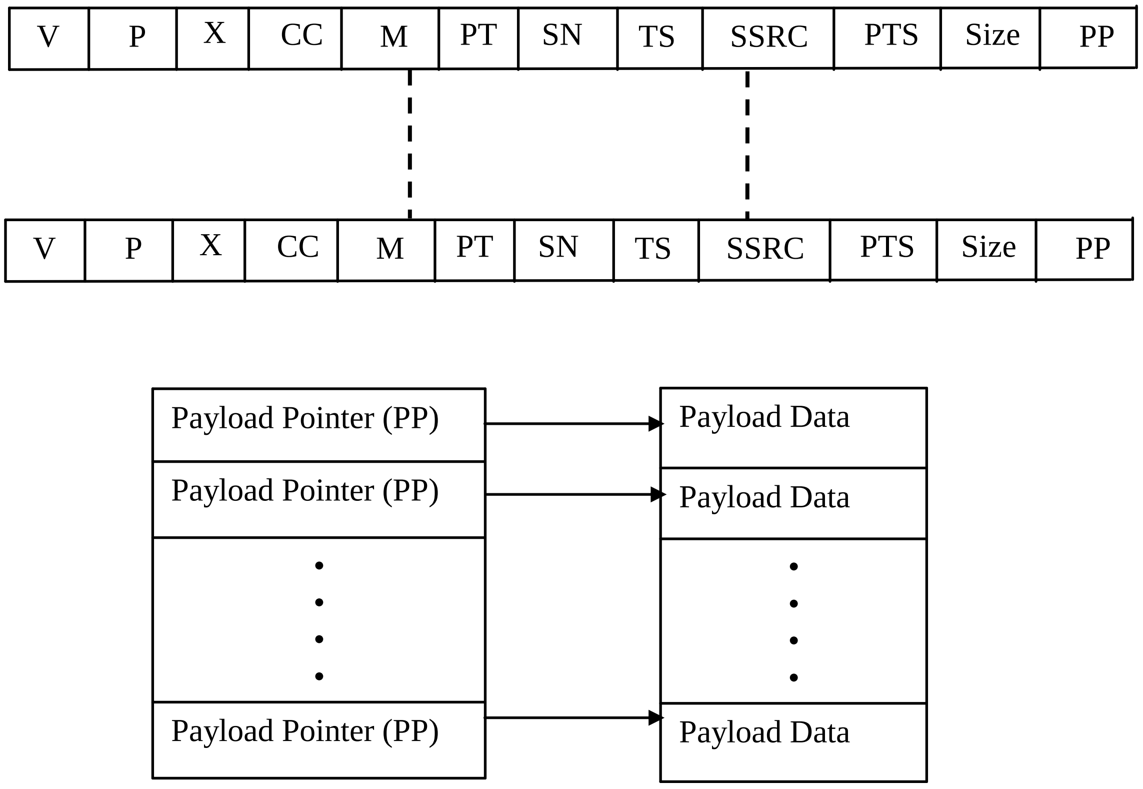

Figure 10: Jitter Buffer Structure

We have implemented both fixed delay and adaptive jitter buffer algorithms [32] in the bare PC softphone. Calculation of jitter follows the specifications in [33]. The bare PC softphone continually monitors performance and reports the values of network delay, end-to-end (total) delay, mean and maximum jitter, and the maximum inter-arrival gap (max delta) between packets. The end-to-end delay is measured as the time that elapses between extracting voice data for a packet from the sender's microphone buffer and copying it to the receiver's speaker buffer. The bare PC also reports the percentage of lost or late packets. We validated jitter and max delta values by comparing them with the values reported by an Ethereal sniffer [34]. Note that synchronized sender and receiver clocks are not needed to compute jitter and max delta. For estimating network and total end-to-end delay, the bare PC computes the approximate delay for a packet by using the value in the timestamp field and the local clock at the receiver. This value of delay is also used to calculate an adaptive playout delay for packets.

3.5.7. Packet Loss

The packet loss calculation uses a 1024 element vector. The vector elements are set to clear initially and again after every 1024 packets that arrive (*SNVector[1024] = {0x00}*). Every time a voice packet arrives, its sequence number is mapped into a position in the vector to set a flag for this position (*SNVector[sn % 1024] = 0x01*). After 1024 packets have arrived, the number of clear positions in the vector is summed to find the packet loss count as follows:

```
$sn\_vector\_count{+}{+};$
$sn\_vector[sn\%1024] = 1;$
$if\,(sn\_vector\_count == 1024)$
{
  $for(i = 0; i < 1024; i{+}{+})$
  {
    $if\,(sn\_vector(i) == 0)\quad packet\_loss\_count{+}{+};$
    $sn\_vector(i) = 0;$
  }
  $sn\_vector\_count = 0;$
}
```

3.6. Audio Task

The audio task is responsible for two primary functions; recording and playback of voice data. Both functions are tightly coupled with the AD1981B onboard codec processing through the microphone buffer and the speaker buffer (Figure 11). This section discusses the interface between the bare PC softphone and the AD1981B codec via the ICH5 DMA controller. The record and playback functions are covered in the subsequent sections of this document.

The record and playback for the softphone are synchronized using the audio codec recording and playback timing. The audio task monitors the recording of the microphone data. Whenever a full buffer of microphone data has been recorded, it is processed. It also processes received voice packets in the jitter buffer. Therefore, every time a buffer is recorded, a packet is played as well. This technique provides the following advantages:

- It simplifies the synchronization of the microphone buffer read/write pointers with the speaker buffer read/write pointers.
- It minimizes the softphone intrinsic delay by keeping the distance between read and write pointers for the microphone and speaker buffers to a minimum.
- It indirectly synchronizes the record and playback timing on both softphone applications.

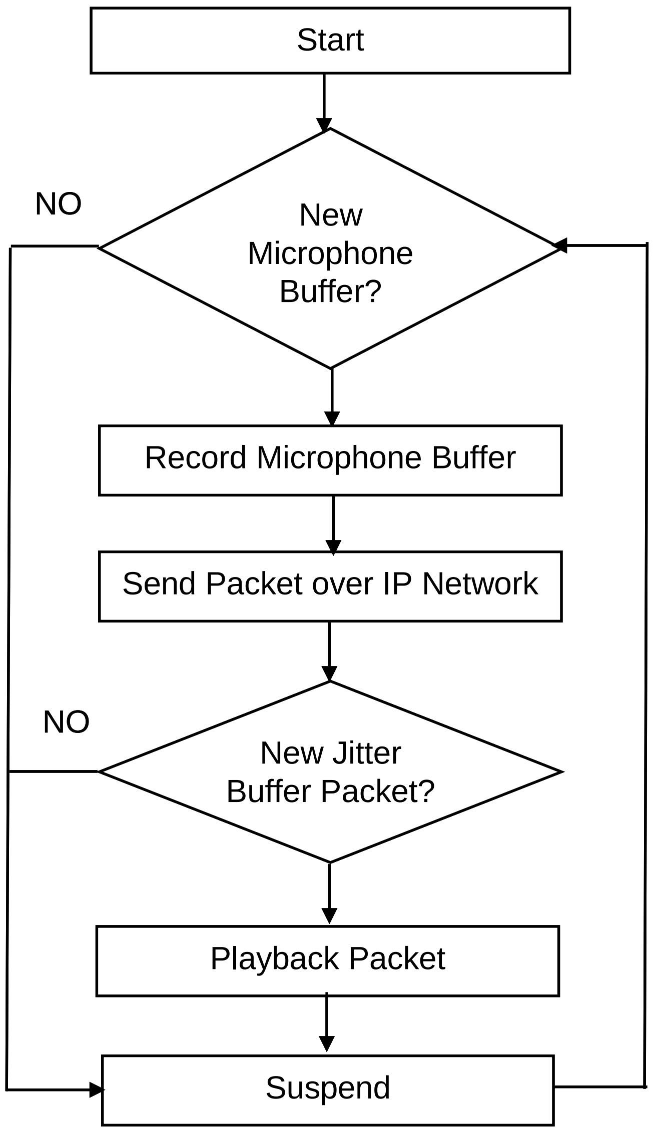

Figure 11: Audio Task State Diagram

While synchronizing playback and recording, the audio task also monitors the condition of the jitter buffer. If no packets are arriving due to network conditions or the any other failures, it will restart the audio task to its start state. The audio task is designed and implemented as a state machine with start, record, play, and suspend states. The audio task processes entire frames of recorded voice data per activation before suspending itself.

3.6.1. The AD1981B Codec

The AD1981B Codec (Coder/Decoder) chip set provides the bare PC softphone application with analog and digital audio capabilities directly from the PC motherboard (Figure 12).

The AD1981B onboard codec continuously samples microphone voice data at the 16-bit PCM rate, converts it from analog to digital format, and stores it internally. The ICH5 DMA controller retrieves recorded digitized voice buffers from the AD1981B codec and stores them in RAM in the microphone circular buffer at the record position. The audio task reads each recorded voice buffer from the microphone buffer starting from the read position. It then does stereo to mono conversion, compresses the data using G.711, adds the RTP, UDP, IP, and ETH headers, and eventually stores each packet in the circular send buffer list, which is accessible via the DPD list. These packets are then placed on the network by the NIC.

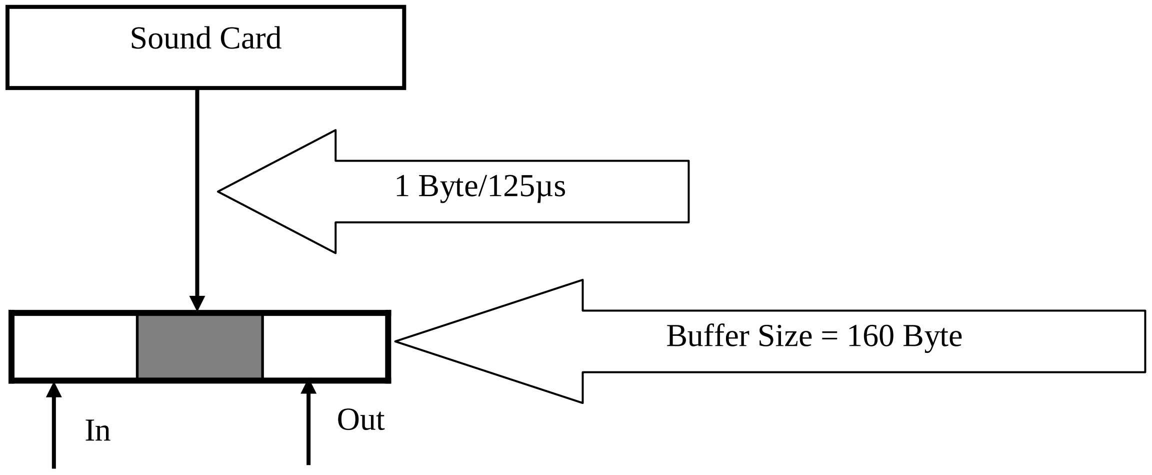

Figure 12: PCM/Codec data rate for 20ms

The audio task is also responsible for playback of voice packets via the speakers. It selects voice packets from the jitter buffer, decompresses the data using G.711, does stereo to mono conversion, and then stores each voice payload into the speaker buffer starting at the write position. The ICH5 DMA controller continuously reads each voice packet from the speaker circular buffer, starting at read position, and transfers them to the AD1981B codec, which converts each voice buffer from digital to analog, and plays it back at the PCM rate. Managing the write and read pointers for the microphone and speaker buffers involves both the audio softphone application as well as the ICH5 controller driver software.

3.6.2. Microphone Record Buffer

The AD1981B codec samples narrowband speech from the microphone at the PCM rate of 8000 samples per second. Each digitized sample has a 16-bit resolution. This produces a digital signal stream with bit rate of 128Kbits/sec (8000*16) per channel. The recording is performed in stereo format via left and right channels, which results in a total of 256Kbits/sec (8000*16*2). The AD1981B controller performs Analog-to-Digital-Conversion (ADC) during recording and Digital-to-Analog-Conversion (DAC) during playback. It also controls other microphone and speaker operations, such as volume and gain.

The digitized voice buffers are first stored in the AD1981B codec internal FIFO and then mapped into the microphone buffer by the ICH5 controller DMA function as shown in Figure 13.

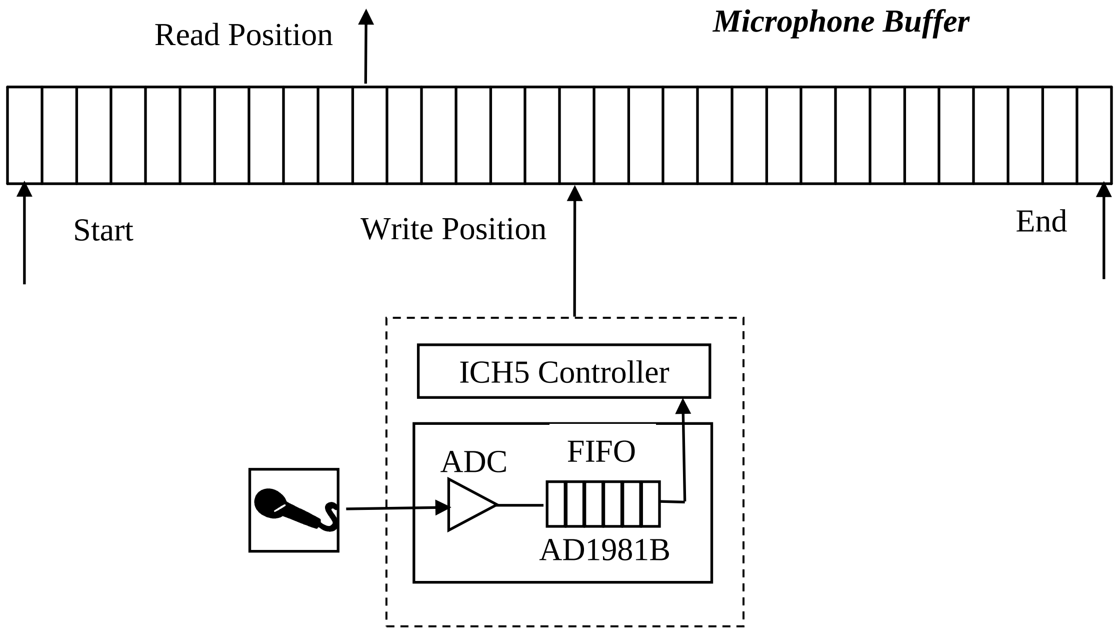

Figure 13: Microphone Record Buffer

3.6.3. Speaker Playback Buffer

In addition to microphone recording and digitization, the AD1981B codec also uses digitized voice buffers stored in its internal FIFO to reconstruct narrowband speech for playback on the speakers. The ICH5 DMA controller driver software is responsible for continuous and sufficient transfer of digitized voice buffers from speaker buffer to the AD1981B codec internal FIFO as shown in Figure 14. The playback requires a bit stream of 128Kbit/sec (8000*16) per channel or total of 256Kbit/sec (8000*16*2) for the left and right channels to reconstruct the original voice.

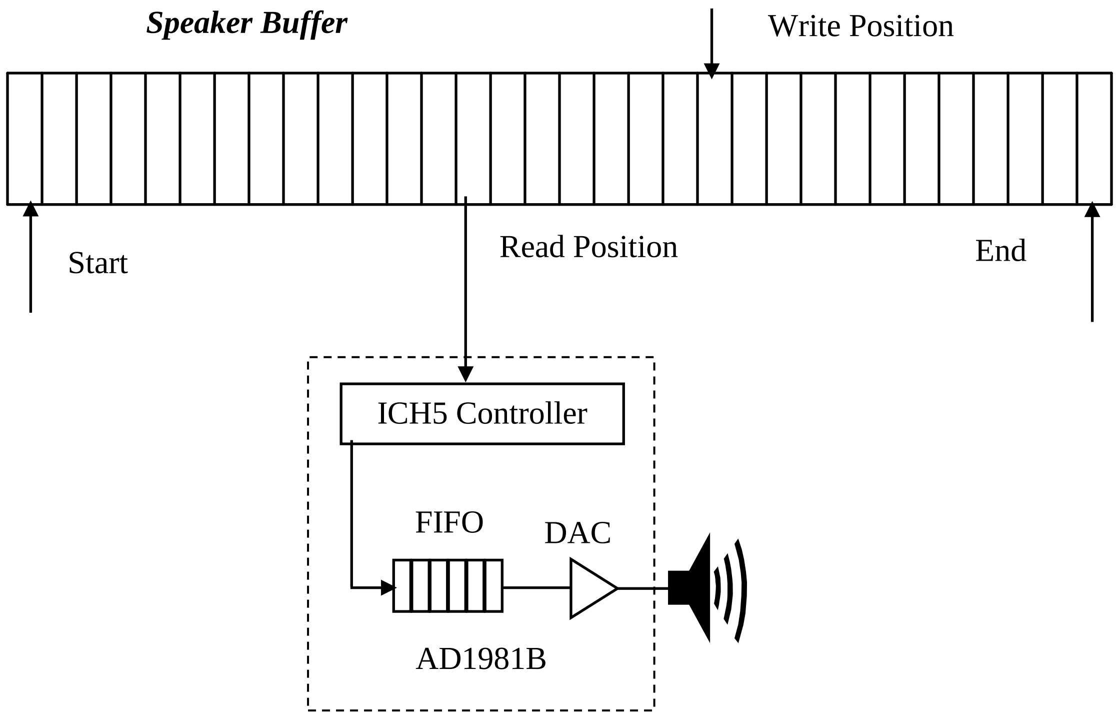

Figure 14: Speaker Playback Buffer

3.6.4. Audio Task Recording

Recording and processing of recorded voice data is one of the two primary functions of the softphone. First, the audio task recording function checks for newly recorded voice data. It then reads each recorded voice packet from the circular microphone buffer, converts it from stereo to mono, compresses it using G.711 and directly writes it to the DPD send buffer. Next, it prefixes the payload with RTP, UDP, IP and ETH headers. The stored voice frames are then transmitted over the network via the NIC. The record task processes an entire frame of recorded voice data per activation before suspending itself.

3.6.5. Microphone Buffer Management

The microphone buffer is a circular list of voice buffers with a write pointer and a read pointer. A voice buffer is written at the write pointer and read at the read pointer. Each element of the microphone buffer holds one packet of raw digitized voice data with a 16-bit resolution and stereo quality. Each element occupies two consecutives bytes for the left channel followed by two bytes for the right channel. The audio task recording function reads buffer elements starting at the read pointer. Proper manipulation of the write and the read pointers is the responsibility of both the ICH5 DMA controller and the audio softphone application.

There are two microphone buffer pointers that require proper and timely management for correct management of the microphone buffer:

- *CIV – Current Index Value (0-31)*
- *LVI – Last Valid Index (0-31)*

The ICH5 DMA controller driver software manages the CIV, and the softphone application manages the LVI for the microphone buffer.

The CIV is an index to the microphone buffer, which always points to the current buffer element where microphone data is being recorded. No processing should be done on this buffer element as it will interfere with data recording process. As part of its initialization steps, using an API call, the audio task sets the microphone CIV to zero so that voice data may be recorded starting in the first element of the microphone buffer. After the current buffer pointed to by the CIV is filled, the ICH5 controller software automatically increments the CIV by one so that microphone data can be recorded in the next available buffer element. The range of valid values for both the CIV and LVI is from zero to 31.

The buffer boundary conditions for the microphone buffer structure must be managed by the audio task recording function using the microphone buffer LVI. The LVI pointer must be set to 31 when CIV has changed to zero, and it must set to zero when CIV has changed to 31, as follows:

```
civ = audio.getMicCIV ()
 If (civ == 31)
      audio.setMicLVI (0)
else if (civ == 0)
     audio.setMicLVI (31)
```

The size of a microphone buffer element must be computed and stored in the BUFFER_SIZE constant at compile time. PCM-16 sampling collects 8 samples per msec, with each sample consisting of four bytes of storage, two bytes for the left channel (LC) followed by two bytes for right (RC). Left and right channel samples are interleaved. The audio softphone application reads and processes packets of multiples of 10 msec (Figure 15), so the frame size is calculated as follows:

*BUFFER_SIZE = Frame Size * PCM Rate * LC Bytes * RC Bytes*

	Frame Size (msec)	**BUFFER_SIZE = (Frame Size) * (PCM Rate) * 2 * 2**
1	10	320 (10 * 8 * 2 * 2)
2	20	640 (20 * 8 * 2 *2)
3	30	960 (30 * 8 * 2 * 2)
4	40	1280 (40 * 8 * 2 * 2)
5	50	1600 (50 * 8 * 2 * 2)

Figure 15: Frame Size and Buffer Size Calculation

The ICH5 controller specification allows for buffer sizes of up to 65535 samples of 16-bit resolution, which is 131,070 (65535*2) bytes of data. In practice, smaller buffers are used as shown in the above table.

The LVI for the microphone buffer must be changed fast enough so that the CIV is changed from 31 to zero and from zero to 31 to ensure no loss of recording. When the CIV reaches 31, the last element of the circular microphone buffer has been filled with voice data. If the LVI is not changed to 0, the recording will halt causing data loss. Similarly, when the CIV reaches 0, the LVI must be changed to 31.

For large buffer sizes, the LVI change is less frequent. It is recommended that the CIV be checked a least a few times during recording of the frame size so that the LVI may change frequently with no loss of data.

3.6.6. Stereo to Mono Conversion

Digitized voice data from the microphone is recorded in stereo format i.e. they are interleaved as two bytes for the left channel followed by two more bytes for the right channel. The conversion from stereo to mono eliminates the two bytes for the right channel from each buffer element (Figure 16). This process reduces the bit rate for the digital stream from 256Kbits/sec to 128Kbits/sec. The stereo-to-mono conversion is performed during G.711 compression by simply skipping the two bytes for the right channel.

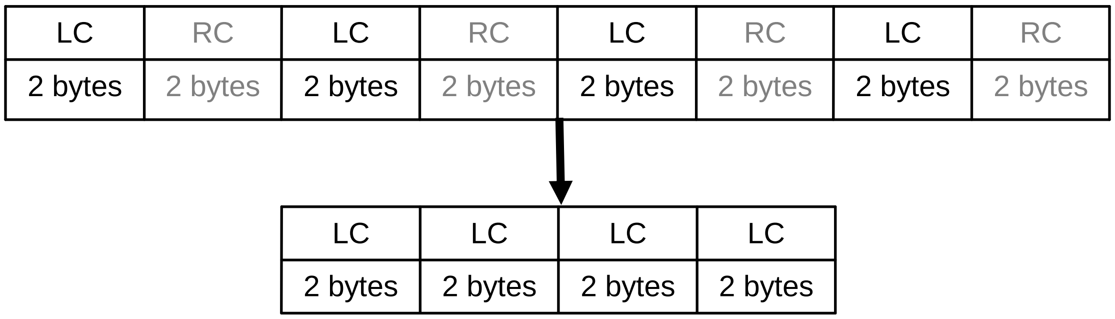

Figure 16: Stereo to Mono Conversion

3.6.7. G.711 Compression

The ITU-PCM (G.711) codec compresses 16-bit PCM samples into 8-bit PCM samples by using a look up table. There are two distinct tables for PCMA and PCMU compression. Each lookup table consists of 16,384 8-bit predetermined values. First, the two most significant bits of 16-bit PCM samples are masked off, while the remaining 14 bits are used as an index to retrieve an 8-bit PCM value from the table. The 8-bit PCM value forms the compressed payload for transmission over the network. This process reduces the bit rate for the digital stream from 128Kbits/sec to 64Kbits/sec. The compression algorithm is valid for either mono or stereo recording formats. For mono, only the 16-bit PCM samples for the left channel are compressed; for stereo, 16-bit PCM samples for both left and right channels are compressed and interleaved. The compression algorithm reads voice data directly from the microphone buffer, eliminates the right channel, compresses the right channel, and directly copies the compressed data to the NIC circular buffer. This eliminates buffer copying of voice data.

3.6.8. Adding RTP Headers

The NIC send buffer now contains the 8-bit PCM compressed voice packets. Next, the voice packet is prefixed with the RTP header (Figure 17). The RTP header fields are processed in “Little Endian” byte order meaning that the LSB bytes are stored before the MSB bytes.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 17: RTP Header Fields

The RTP header fields are populated as follows:

- ***Version(V) 2 bits:*** This field is set to 2, which is the current working version of RTP protocol.
- ***Padding(P) 1 bit:*** This field is set to zero to indicate no additional padding octets exist at the end, which is not part of the payload.

- ***Extension(X) 1 bit:*** This field is set to zero to indicate a fixed RTP header is used.
- ***CRC Count (CC) 4 bits:*** This field is set to zero to indicate no additional data sources are used.
- ***Marker (M) 1 bit:*** This field is set to zero to indicate there is no tailoring of the RTP header.
- ***Payload type (PT) 7 bits:*** This field is set to zero for PCMU and 8 for PCMA compression.
- ***Sequence Number (SN) 16 bits:*** This field is populated with a 16-bit logical circular number starting from zero and incremented by one for each subsequent packet.
- ***Time Stamp (TS) 16 bits:*** This field is populated with a 32-bit logical circular number by using the local timer converted into PCM byte units. This field has the local PCM time at which the packet is created. It is calculated by multiplying the local timer value by two. Each timer unit is equal to 250 μsec, which at the PCM sampling rate constitutes two bytes of data.
- ***Synch Source (SSRC) 32 bits:*** This field has the IP address of the sender.

3.6.9. Adding UDP, IP and ETH Headers

The DPD send buffer now contains the compressed 8-bit PCM packet of voice data prefixed with the RTP header. This is now prefixed with the UDP header fields (Figure 18). The UDP header fields are also in “Little Endian” byte order.

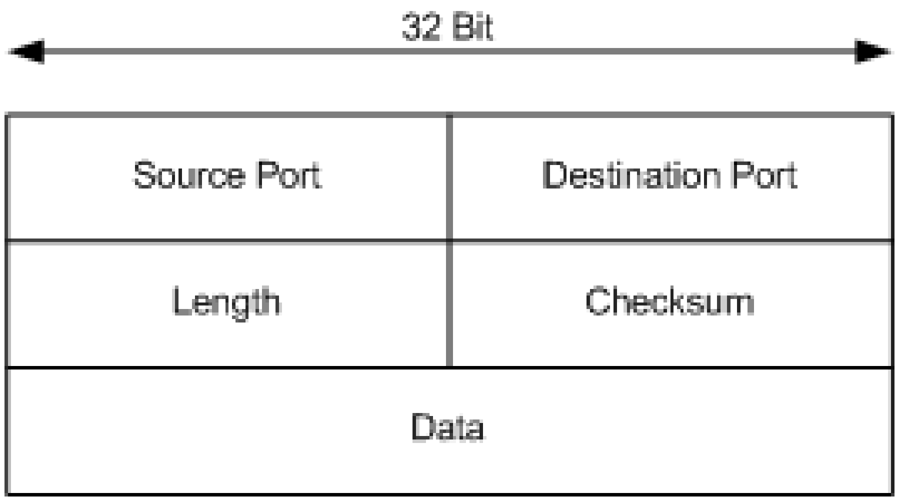

Figure 18: UDP Header Fields

The UDP fields are as follows:

- ***Source Port 16 bits:*** This is the sender's port number.
- ***Destination Port 16 bits:*** This is the receiver's port number.
- ***Length16 bits:*** This is the number of bytes in the UDP datagram.
- ***Checksum: 16 bits***: A checksum is placed in this field.

The IP object prefixes the UDP datagram with IP header fields. The ETH object prefixes the IP packet with the ETH header fields. The last step is to call the NIC API to send the packet on the network. The UDP, IP and ETH code were developed by others during previous projects.

3.6.10. DPD Send Buffer Management

The DPD send buffer now contains digitized voice data prefixed with the RTP, UDP, IP and ETH headers. The DPD send buffer is located in RAM, which provides an interface between the NIC DMA controller and the audio softphone application. The audio softphone application and the NIC DMA controller software must coordinate to manage the send buffer.

The audio task record function must perform the necessary actions for managing the send buffer as follows:

- Acquire a valid pointer to write to the send buffer
- Copy the data into the send buffer
- Add the RTP, UDP, IP and ETH header fields
- Send the voice packet on the network

3.7. Audio Task Playback

Processing and playback of the incoming network voice packets is the other primary function of audio softphone. The audio task will playback a packet if and only if the audio task recording function has detected and recorded a new buffer of microphone data via an API call to the audio card driver. The playback function of the audio task fetches voice packets from the jitter buffer, decompresses the voice data using G.711, and copies the data directly from the NIC buffer to the speaker buffer. The decoded data is mono, and therefore must be converted to stereo format. This is accomplished by copying the decompressed voice data into left channel as well as the right channel.

3.7.1. Jitter Buffer Playout

The jitter object uses a priority queue to implement the jitter buffer functions. The receive task inserts voice packets into the jitter buffer queue as they arrive by using the jitter buffer insert function. The audio task playback function fetches packets from the jitter buffer one at a time using the jitter buffer remove function. The jitter buffer remove function uses the packet sequence number as well as the packet playout time for all packets in the queue and returns the packet with smallest sequence number whose playout time has expired. The playout time for each packet is stored with each packet during the jitter buffer insert as discussed previously. The stored playout time is compared with current system time to determine whether the play out time for a packet has elapsed. If no voice packet is selected by the jitter remove function, then packet loss concealment will take place, via playback of a background noise packet.

3.7.2. G.711 Decompression

The ITU-PCM G.711 codec decompresses 8-bit PCM voice samples into 16-bit PCM samples by using a look up table for PCMA or PCMU decompression based on the payload type field in the RTP header for the incoming voice packets. Each lookup table consists of 256 16-bit predetermined values. The 8-bit PCM sample is used to retrieve a 16-bit PCM value from the table, which forms the decompressed payload for playback on the speakers. This process increases the bit rate for the digital stream from 64Kbits/sec back to 128Kbits/sec per channel. The decompression algorithm is valid for either mono or stereo formats (8-bit PCM samples for the left channel or 8-bit PCM samples for the left channel and right channel are decompressed).

3.7.3. Mono to Stereo Conversion

The onboard AD1981B codec requires that digitized incoming voice buffers for playback on the speaker be stored in the speaker buffer in stereo format i.e. they must be interleaved as two bytes for the left channel followed by two bytes for the right channel. The conversion from mono to stereo copies each 16-bit PCM sample into the left channel as well as the right channel (Figure 19). The mono to stereo conversion is implemented as part of the G.711 decompression algorithm to eliminate any intermediate copying of voice data.

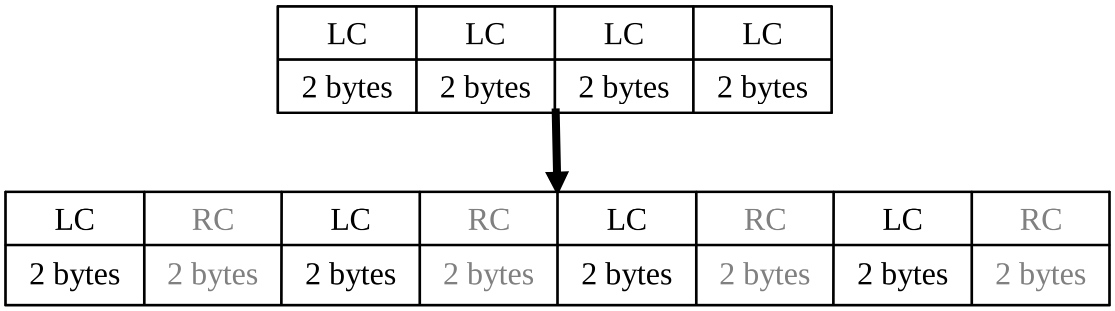

Figure 19: Mono to Stereo Conversion

3.7.4. Speaker Buffer Management

Each element of the speaker buffer holds exactly one packet of digitized voice data that has arrived from the network. Each packet must have the actual raw voice data for playback by the speaker at a 16-bit PCM rate and in stereo format. Each 16-bit PCM sample will occupy two consecutives bytes for the left channel and two consecutive bytes for the right channel. The circular speaker buffer is located in RAM and provides an interface between the ICH5 DMA controller and bare PC softphone application during the playback. The speaker buffer is a 32-element circular buffer with each element containing a single frame of digitized voice. The ICH5 DMA controller and bare PC softphone application must coordinate to ensure proper manipulation of the record and read pointers.

The bare PC softphone application writes the incoming voice frames into the speaker buffer, and the ICH5 controller driver software moves them into the AB1985B codec FIFO for playback on the speakers. There are two buffer pointers associated with the circular speaker buffer:

- *CIV – Current Index Value (0-31)*
- *LVI – Last Valid Index (0-31)*

The ICH5 controller driver software manages the CIV and the bare PC softphone application manages the LVI for the speaker buffer. The CIV is an index for the speaker buffer, which points to the buffer element where speaker data is being read for playback on the speakers. No processing should be done on this buffer element as it will interfere with playback. The CIV is set to a zero during the initialization of the ICH5 controller driver software, so that voice data may be read starting from the first buffer element. After the current buffer is read, the ICH5 controller software automatically increments CIV by one if it is less than or equal to 31. When the CIV reaches 31, the CIV can be changed from 31 to zero but only if the LVI has been set to zero by the softphone application.

The speaker buffer boundary conditions are managed by the audio softphone application using the speaker buffer LVI. The LVI pointer must be set to 31 when the CIV changes to zero, and to zero when the CIV changes to 31, as follows:

```
civ = audio.getSpkerCIV ()
 If(civ == 31)
        audio.setSpkrLVI (0)
else if(civ==0)
        audio.setSpkrLVI (31)
```

The size of a speaker buffer element is the same as the size of a microphone buffer element, which is defined by BUFFER_SIZE constant. The speaker playback also requires 16-bit voice data samples at the PCM rate. Each sample requires two bytes for the left channel and two bytes for the right channel. Samples for the left and right channels are interleaved in the buffer. We omit details concerning the adjustment of the LVI and CIV for the speaker buffer as it is similar to the case of the microphone buffer discussed previously.

3.8. Security

The bare PC softphone implements a lightweight security scheme designed for peer-to-peer VoIP communication. The scheme relies on several standard security mechanisms. For example, we use RSA encryption and signatures for secure symmetric key exchange and peer authentication. Similarly, AES encryption and a SHA-1 hash ensure the privacy and integrity of voice data. The scheme provides limited replay protection. We will add stronger replay protection and AES counter mode [45] in the future. It is possible to disable security by using a voice security flag in case the users do not wish to protect their conversation, or if performance takes precedence over security. Users can also choose either encryption or authentication protection if both are not needed.

The bare PC softphone security protocol consists of a simple two-way handshake over TCP to exchange the AES key (or keys), which is followed by transfer of encrypted authenticated voice data over RTP/UDP. In keeping with the goal of simplicity, we do not support negotiation of a cipher suite or the use of X.509 [41] certificates. However, it would be easy to add these capabilities if needed. Each bare PC softphone generates an RSA key pair and an AES key. At present, we do not address the important issue of key management, but assume that an out-of-band method (such as manual exchange of public keys between peers or use of a trusted key server) enables the peers to obtain and verify each other's RSA public keys. We also assume that the bare PCs have public IP addresses or that it is possible to configure firewalls in the presence of NAT (network address translation) to enable the peers to establish a TCP connection for the handshake and transfer voice data over UDP. We are currently investigating alternate approaches for establishing call connections through NAT/firewalls, including STUN [46] and ICE [47].

3.8.1. Handshake

When making a call, the caller is the client and the callee is the server for the purposes of establishing a TCP connection to exchange AES keys. The caller generates an AES key and its SHA-1 hash. We encrypt the latter with the caller's private RSA key to produce a signature. The AES key and its signature are then encrypted with the callee's public RSA key and a message consisting of these encrypted items is sent to the callee. The callee decrypts the message using its private RSA key and recovers the AES key and its signature. It then decrypts the signature using the public RSA key of the caller and verifies the signature by computing the SHA-1 hash of the AES key and comparing it with the decrypted signature. If there is a match, the callee responds by generating a new AES key for use in the opposite direction and repeating each of the above steps taken by the caller. If desired, the same AES key can be used in both directions. The caller processes the received message in the same manner as the callee and the TCP connection is closed. The messages involved in the handshake are shown in Figure 20 and the details of key exchange processing are shown in Figure 21.

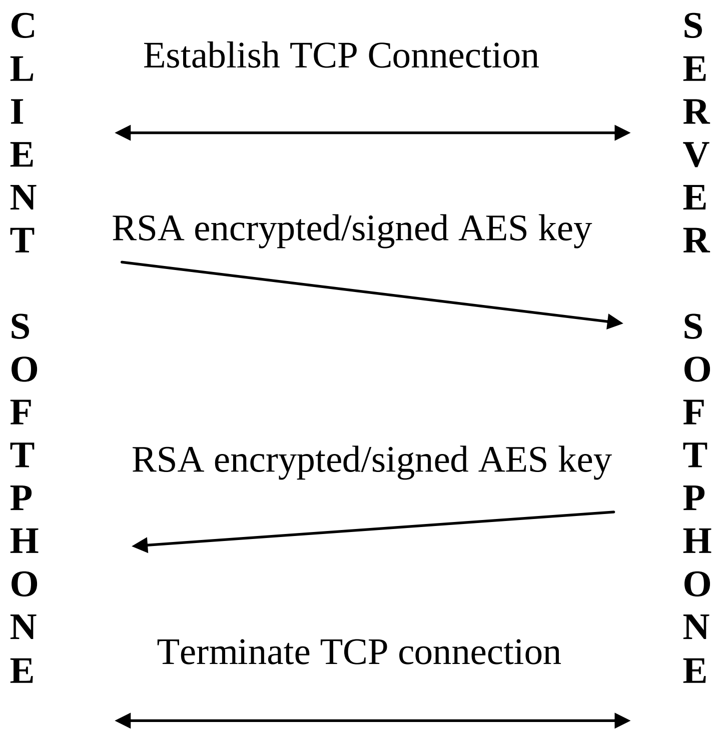

Figure 20: Handshake Messages for AES Key Exchange

If either party is unable to decrypt the AES key or verify the signature, the call connection attempt fails, and a message appears on the screen to notify the user. Otherwise, the call proceeds to the next stage, which is the exchange of voice data. The above key exchange is secure, since only the caller and callee can decrypt the messages with their respective private keys and recover the AES keys and signatures. Moreover, only the caller and callee could have produced the signatures with their respective private keys. If it were necessary to ensure liveliness of the key exchange and protection against replay, each side would also need to generate a nonce, include it in the signature and send it as part of the encrypted key exchange message. In this case, the nonce needs to be stored by each side and also returned in the response, resulting in one additional handshake message sent by the caller.

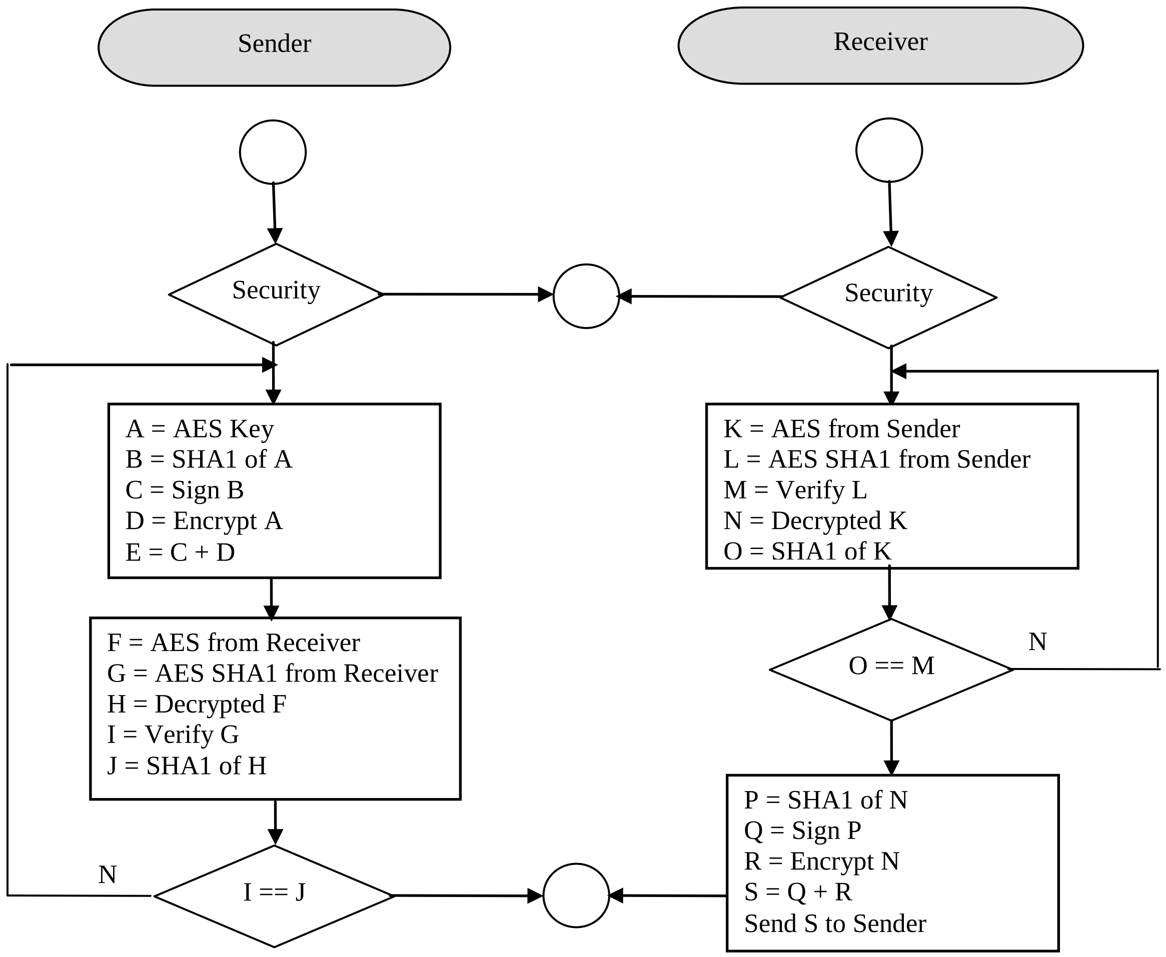

Figure 21: AES Key Exchange Processing

The security of the handshake relies on the ability of the bare PC softphones to protect the private RSA keys. We use an adaptation of a standard technique [49] to protect the private keys, which consists of the following steps. When the RSA keys are generated, the user supplies a password. A SHA-1 hash of the password is created, and an AES key is derived from the hash. This key is used to encrypt the private RSA key. Only the encrypted private RSA key is stored. To recover the private RSA key, the user enters the password and the hash is recomputed. The AES key is again derived from the hash and used to decrypt the private RSA key.

3.8.2. RSA Key Generation and Exchange

Each bare PC softphone generates private and public RSA keys on behalf of its user. For two users to communicate, the bare PC softphone of a user must know the public key of the other user. For now, these public keys are placed on a floppy disk or flash drive and read into each softphone.

3.8.3. Voice Payload Security

If the handshake is successful, each side has the AES key generated by the other side and is able to send and receive encrypted and authenticated voice messages. First, the recorded voice data is compressed as described earlier. Then the sender computes a SHA-1 hash of the compressed voice data and the RTP header, and encrypts the voice data and the hash using its AES key. The inclusion of an encrypted hash serves to authenticate the message and verify the identity of the sender. By including the RTP header in the hash, we can provide limited protection against replay, since there is a timestamp field in the header. The receiver decrypts the compressed voice data and the hash using the AES key transferred during the handshake. It then computes and verifies the hash in the usual manner. Since the AES key was exchanged securely, only the legitimate parties can decrypt the voice message, and its integrity and the sender’s identity are guaranteed. After decryption, the voice data can be played back after it is decompressed. Figure 22 shows the steps involved in the secure transmission of voice data.

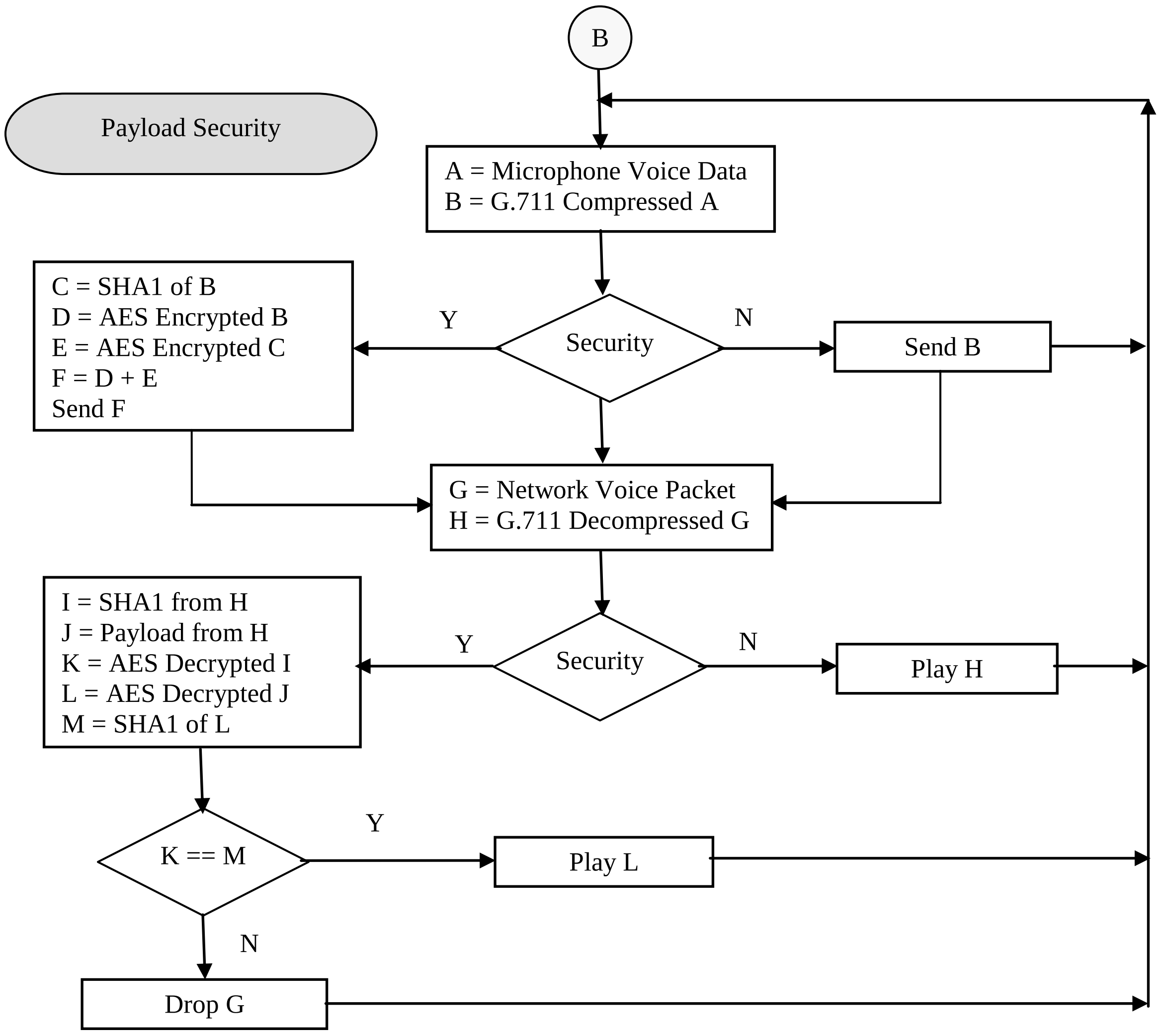

Figure 22: Secure Transmission of Voice Data

Chapter 4. Performance

In this chapter, we present the results of experiments conducted with bare PC softphones on a LAN and on the Internet. We compare the performance of bare PC and WinRTP softphones, study the impact of heavy load conditions, demonstrate the feasibility of Voice over Ethernet, and examine the overhead due to VoIP security mechanisms.

4.1. Performance on a LAN

To study performance of the bare PC VoIP application, we conducted several experiments in our laboratory using a simple, isolated (i.e., there is no connection to external networks or the Internet) test LAN to measure call quality. For ease of data collection and measurement, a hub is used (instead of a switch) to connect the bare PCs running the VoIP clients, and an Ethereal sniffer is connected to the hub for passive monitoring of calls. Each bare PC is a 2.4 GHz Dell Optiplex GX260 with 512 MB memory, an external 3Com 905CX network interface card, and onboard audio chip AD1981B. The WinRTP softphone runs on an identical machine (the OS is Windows XP and we disabled unnecessary services). A commercial tool to calculate the MOS is also connected to the hub (Figure 23). The propagation delay between a pair of VoIP clients is negligible. In what follows, note that one-way measurements report separately the results for each direction in two-way paired (simultaneous) voice streams, whereas two-way measurements report the results considering the combined data from both directions.

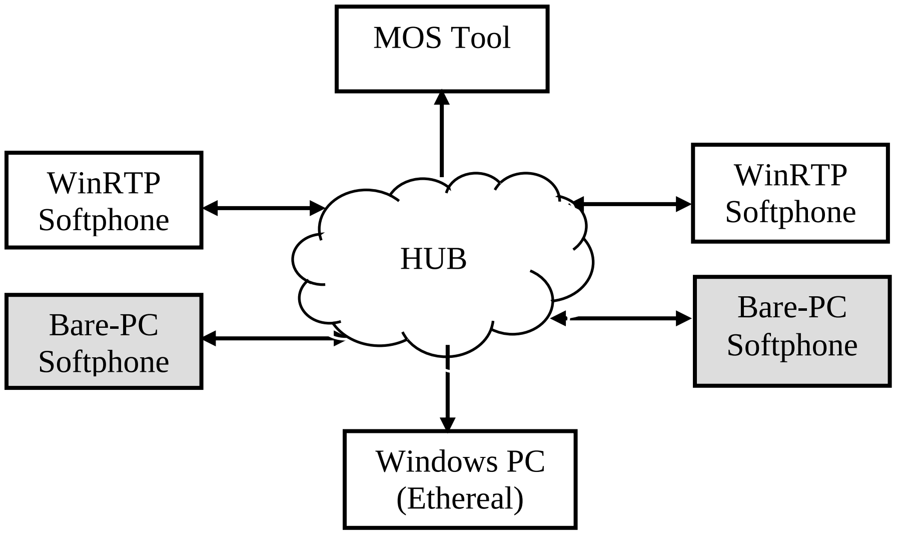

Figure 23: LAN Measurement Setup

First, we transferred voice data between two bare PCs running the VoIP application and determined packet loss, delay, and jitter for voice packets sizes ranging from 10 ms to 120 ms. As there was no other traffic on the network, no packet loss occurred. Also, values of delay and jitter were well within the prescribed ranges for acceptable voice quality, and the MOS remained at 4.43 throughout. The experiments were repeated after replacing first one bare PC and then both bare PCs with a VoIP client running on Windows. The bare PC is capable of supporting voice packet sizes ranging from 10 ms up to 120 ms, but the Windows client only allowed a maximum packet size of up to 60 ms.

The maximum packet interarrival time (delta) in each direction for different frame sizes is shown in Figure 24. For voice data sent by either a bare PC or a Windows client, the maximum interval between packet arrivals is close to the voice frame size in ms. Therefore, a linear relationship is observed, regardless of whether the receiver is a bare

PC or a Windows client. However, as shown in Figure 25, if we plot the maximum deviation from the voice packet size considering traffic from both directions, the deviation for a bare PC is seen to be smaller than that for the Windows client.

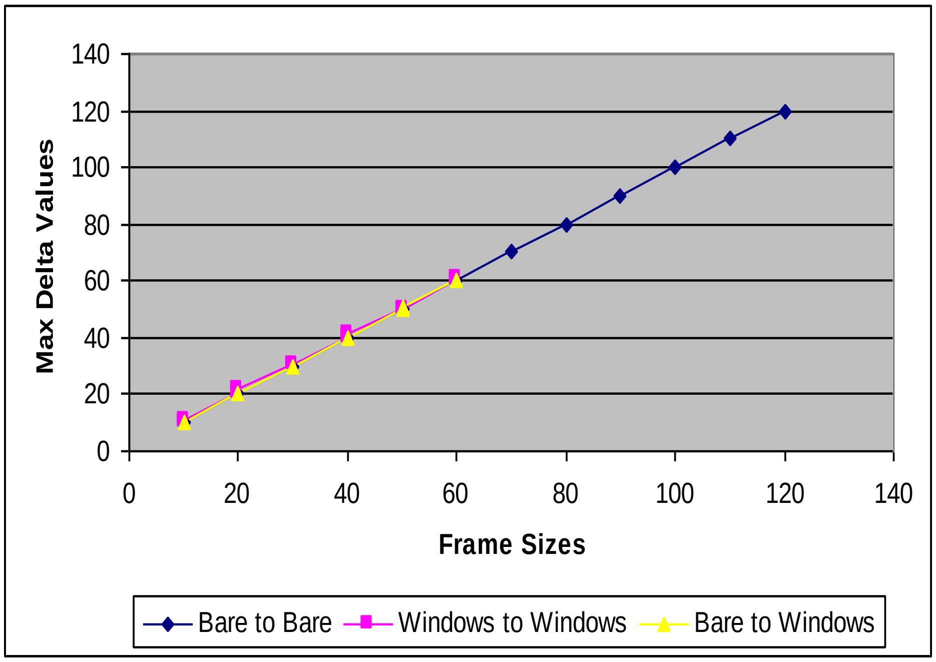

Figure 24: Maximum Packet Interarival Time (ms)

Next, we consider the maximum jitter separately in each direction for a two-way voice stream, which is shown in Figure 26 and Figure 27 respectively. The Windows clients exhibit a marked directional asymmetry when sending voice data to each other, which is likely caused by unmatched systems and operating system behavior. In contrast, the maximum jitter for voice data sent between a pair of bare PC clients is significantly lower than the values for the Windows clients and also has less asymmetry. This occurs because a bare PC has less overhead, and its behavior is more uniform and predictable. It is interesting to note that the performance of a Windows client sending to a bare PC is better than when it is sending to another Windows client, whereas the bare PC performance is the same whether it is sending to a Windows client or a bare PC. The corresponding mean one-way jitter (in each direction) for the above experiments, shown in Figures 28 and 29 respectively, confirms the lower jitter values for the bare PC. Note that jitter measurements for both Windows and bare PC clients are well within the recommended jitter limit of 50 ms. Also, note that the accuracy and resolution of the measurements depend on the underlying operating system (Windows) on which the sniffer is running. We have not attempted to measure these limits in our studies.

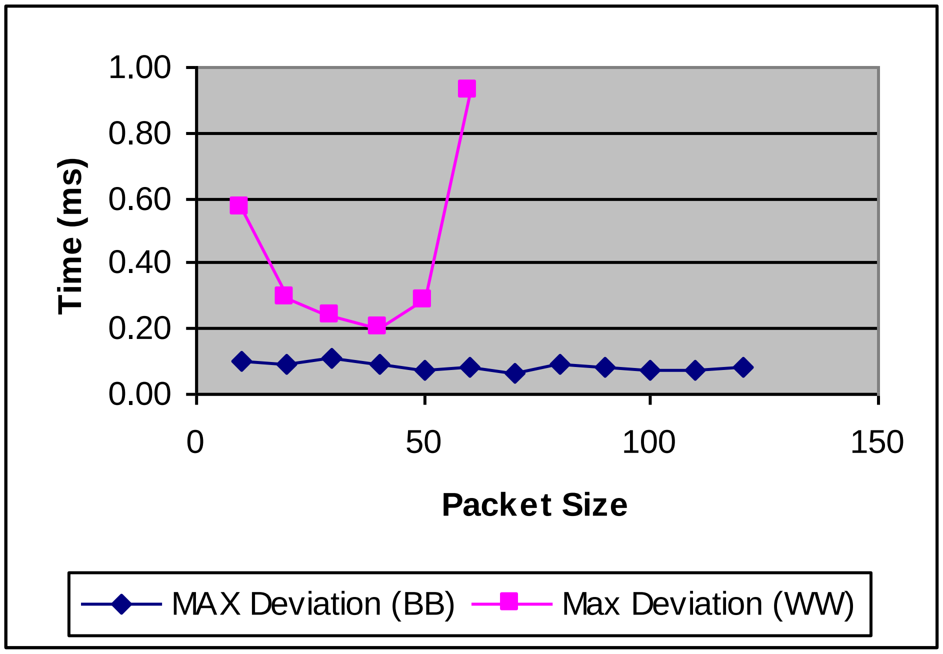

Figure 25: Maximum Two-Way InterPacket Deviation (ms)

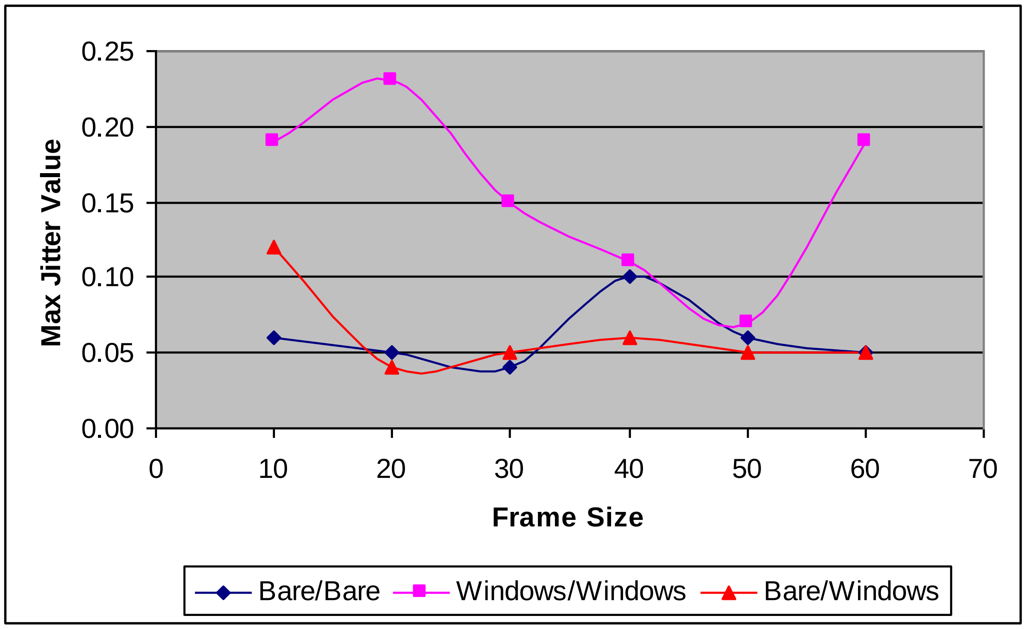

Figure 26: Maximum One-Way Jitter (ms)

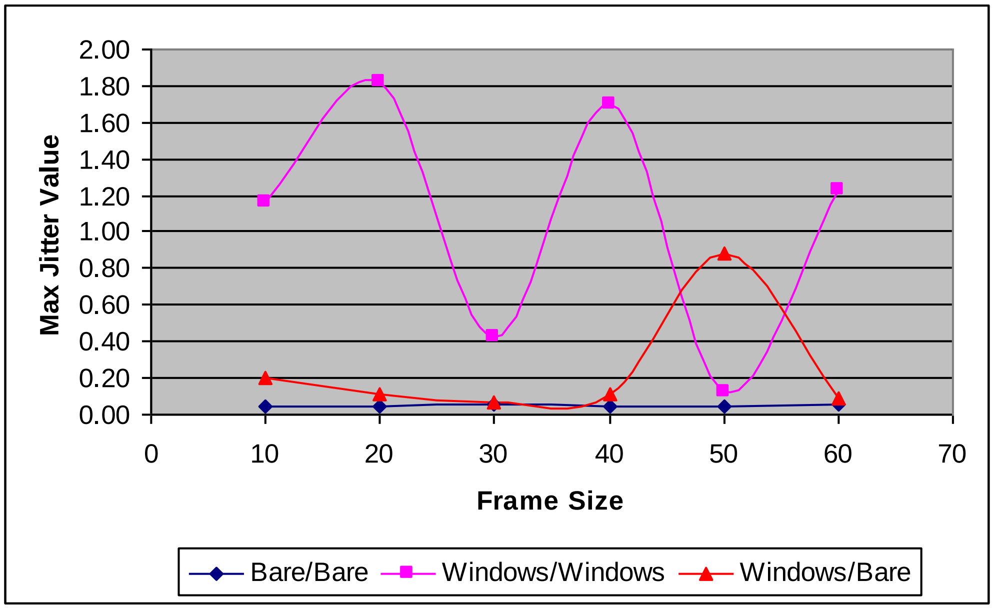

Figure 27: Maximum One-Way Jitter (ms)

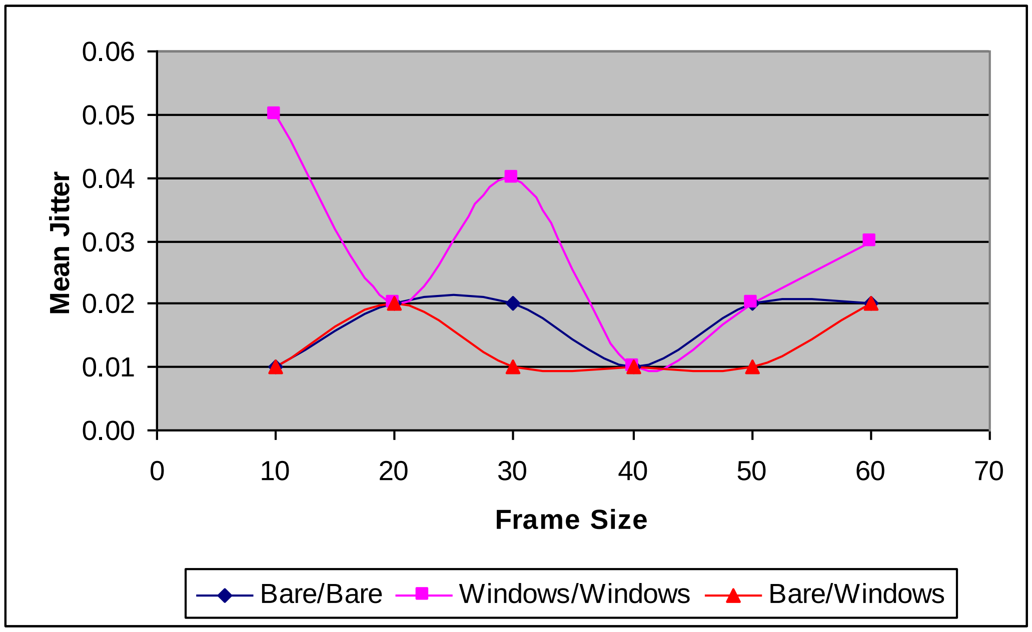

Figure 28: Mean One-Way Jitter (ms)

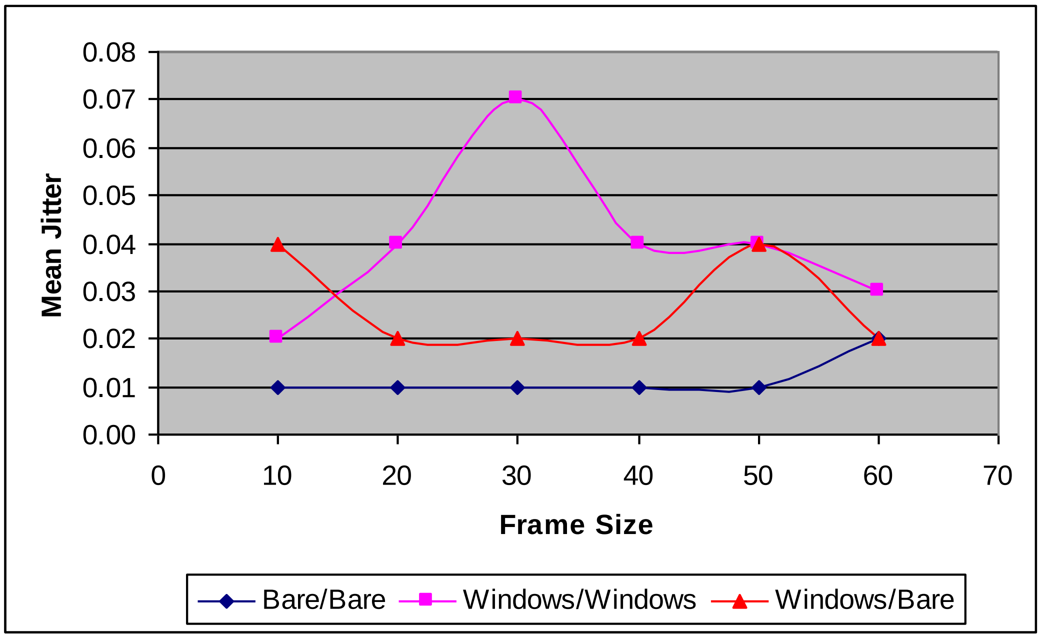

Figure 29: Mean One-Way Jitter (ms)

In Figure 30, we show the maximum jitter when considering two-way traffic. The graph clearly shows that the maximum jitter when a bare PC client is sending is significantly lower than when a Windows client is sending. Again, we observe that the performance of the Windows PC improves when it is sending voice to a bare PC.

In all cases considered above, since there is no packet loss, larger voice packet sizes do not have an observable impact on call quality. We introduced moderate levels of background traffic on the network and repeated the above experiments. We observed similar performance gains for the bare PC client over the Windows client. However, for both bare PC and Windows systems, there was a minimal impact on overall call quality with occasional packet loss and slightly larger values of delay and jitter (no significant change to the MOS was seen). However, the loss of a larger packet now has an observable effect.

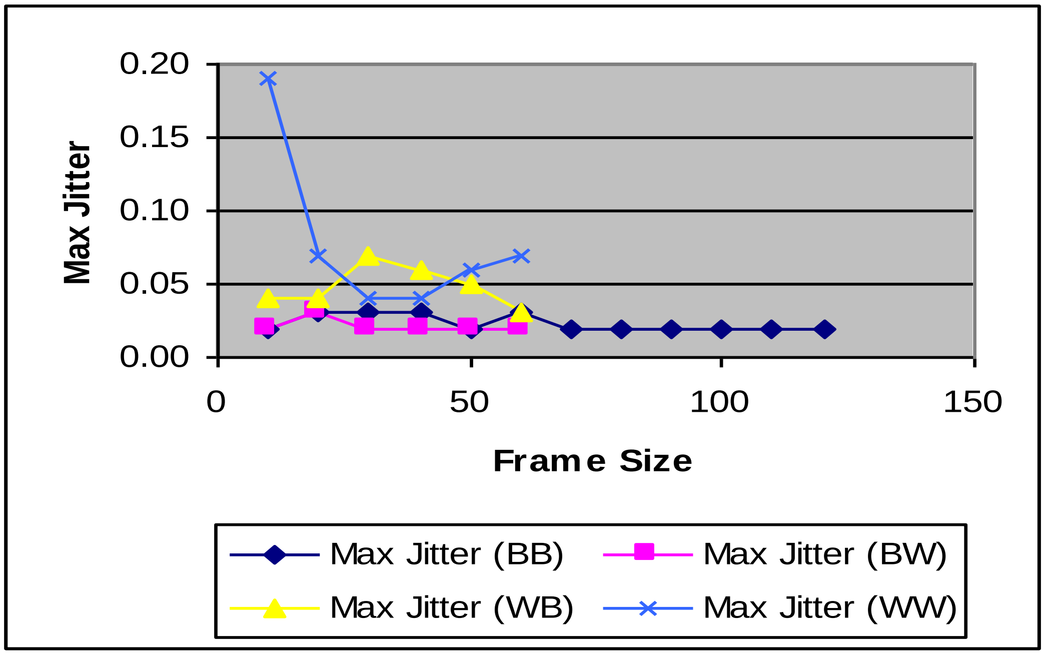

Figure 30: Maximum Two-Way Jitter (ms)

4.1.1. Performance under Heavy Load

Next, we conducted experiments on a LAN to test the call quality of the bare PC softphone when it is performing other tasks. For example, these experiments can simulate a situation when other applications are running on the bare PC concurrently with the softphone. The experiments consisted of interleaving 20 ms voice packets and dummy packets of 1038 bytes containing an Ethernet header only. The number of dummy packets was increased gradually from 1 to 30. We found that CPU utilization was very low and the call quality ranged from good to acceptable for up to 20 dummy packets. When 30 dummy packets were sent, the call quality was poor. We repeated the experiment while flooding the network with background traffic from another source by using the MGEN tool [36]. In this case, call quality became poor with only 20 interleaved dummy packets. Although we could not interleave dummy packets in this manner on the Windows machine, we found that the call quality of the WinRTP softphone was unacceptable with an increased load on the system when the CPU utilization reached 30%. These results indicate that a bare PC can sustain a heavier load while running a softphone with or without background traffic.

4.1.2. Performance of Voice over Ethernet

Finally, we studied the performance of a bare PC softphone in an Ethernet LAN with no routers. In this case, we used bare PC softphones to investigate the feasibility of using voice packets that only had an Ethernet header (i.e., we eliminated the RTP, UDP and IP headers). We believe that it is much easier to incorporate such a Voice over Ethernet service using a bare PC rather than an embedded system, Exokernel, custom

Linux kernel, Linux, or Windows OS. We are not aware of any published studies that have used voice over Ethernet. In this case, packets are delivered by using the MAC address (a packet carries no IP address, sequence number, timestamp, or port numbers).

Of course, this voice over Ethernet service has several drawbacks. For example, packet loss cannot be detected and no ordering of packets is possible, causing packets to be played in the order of arrival. Moreover, packets cannot be forwarded across IP subnets by routers (or across the Internet) due to a lack of IP addresses. However, in a pure switched Ethernet LAN environment, there is virtually no packet loss or out of order packets. The tradeoff is that these packets reduce VoIP bandwidth consumption in a LAN environment, and thus enabling increased call capacity (or more room for other traffic on the LAN). In our experiments, we found that call quality ranged from excellent to good. Packet size is reduced from 213 bytes to 174 bytes, and the savings in bandwidth is about 19%. Voice over Ethernet may be feasible in a small organization or an in-building LAN. More studies are needed to determine the applicability of this approach and the ability to integrate it with IPv6 link local addresses.

4.2. Performance on the Internet

In order to evaluate the performance impact of bare PC optimizations, we conducted several experiments over the Internet, wherein the bare PC softphone application was tested in typical home and campus/business environments (Figure 31). The distance between end points on the Internet was between 16-22 hops. In a home, the bare PC softphone was tested using both DSL and cable modem connections to an ISP. On a campus network, the bare PC was directly connected to the campus LAN through a

100 Mbps Ethernet switch or hub. The PC hardware, NIC and onboard audio chip specifications are the same as for the LAN experiments in Section 4.1.

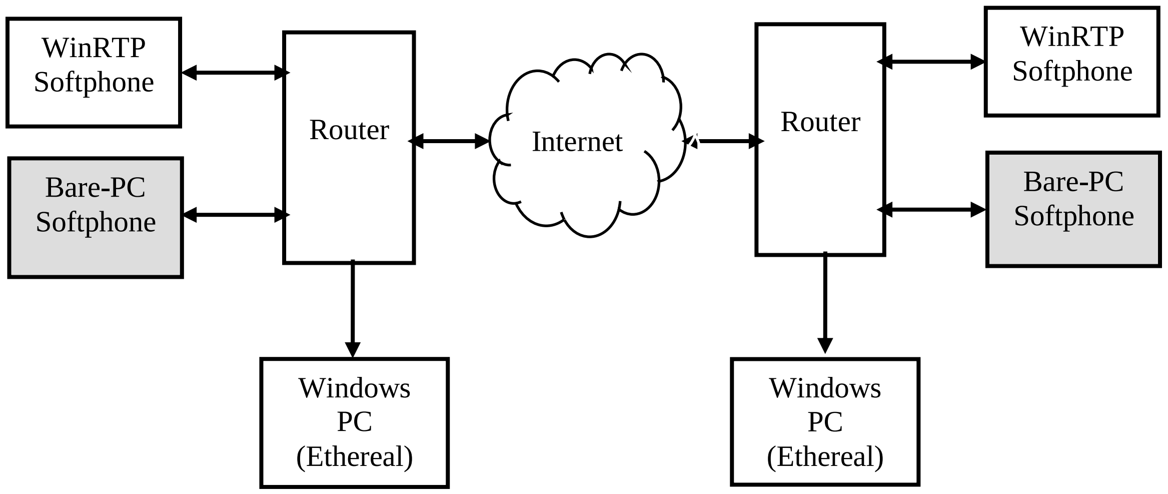

Figure 31: Internet Measurement Setup

A fixed delay jitter buffer was used in our experiments. We do not show packet loss as we did not observe significant packet loss (except in a LAN under conditions of heavy system load when testing the limits of a bare PC; in this case, the reported values of packet loss were unreliable since the systems were unstable). We also did not measure WinRTP-to-WinRTP performance as the preceding LAN experiments showed that this was worse than WinRTP-to-bare PC performance.

Figures 32, 33 and 34 show respectively the maximum packet inter-arrival time (max delta), the maximum jitter, and mean jitter for a bare PC-to-bare PC connection and a bare PC-to-WinRTP connection as the packet size is varied. The jitter values for the bare PC-to-bare PC connection are always smaller than those for the WinRTP-to-bare PC connection. This is due to the efficient task scheduling and low processing overhead on the bare PC softphone. Note that the larger differences in the values of max delta (60 ms for 10 ms packet for example), maximum jitter (6 ms for 30 ms packets for example), and mean jitter (2.5 ms for 50 ms packets for example) reflect the variation in the Internet conditions during the experiments.

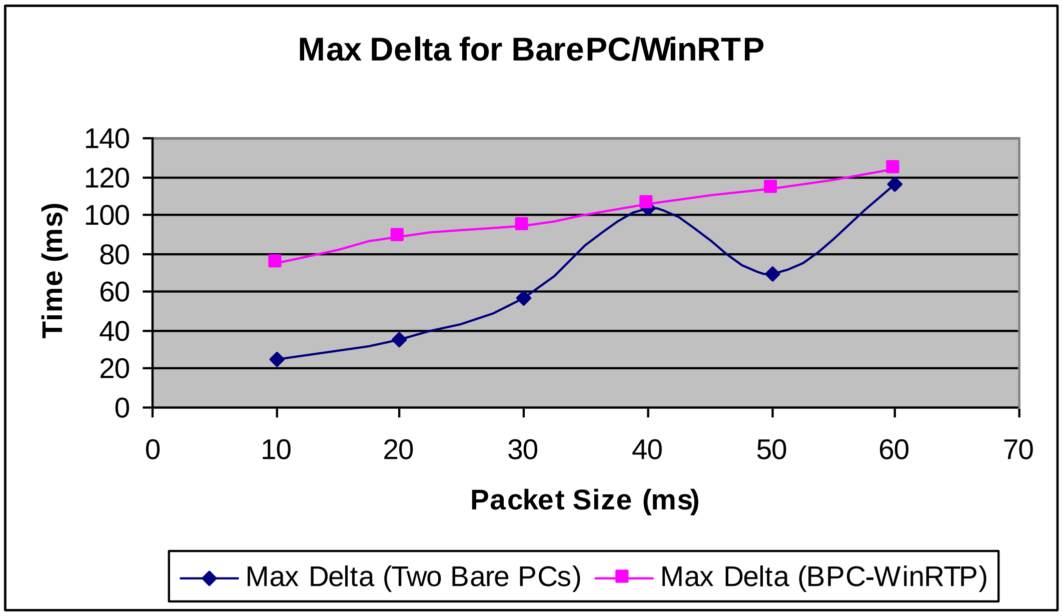

Figure 32: Maximum packet inter-arrival time (max delta)

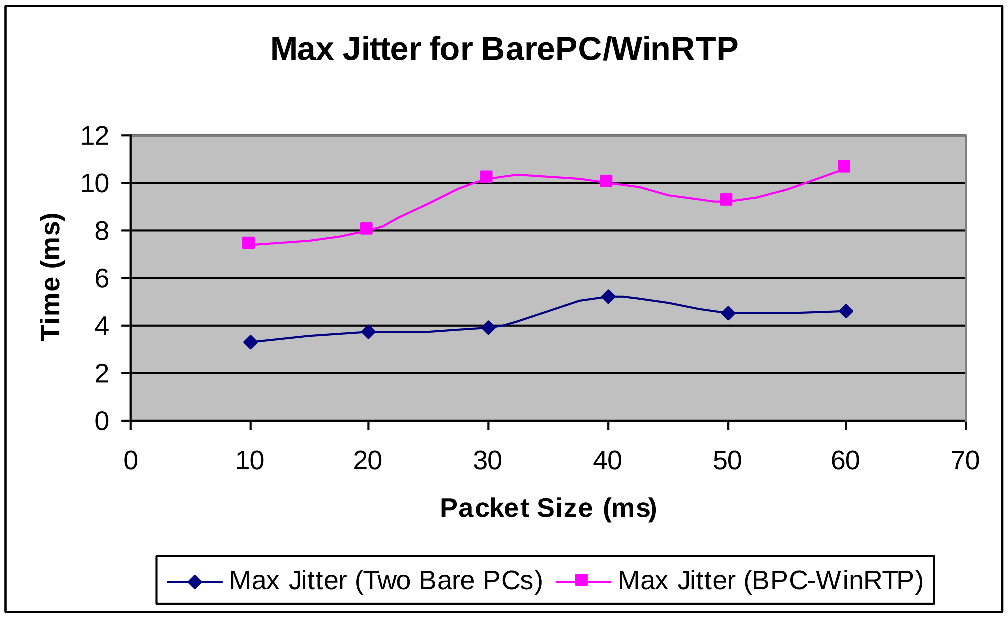

Figure 33: Maximum jitter

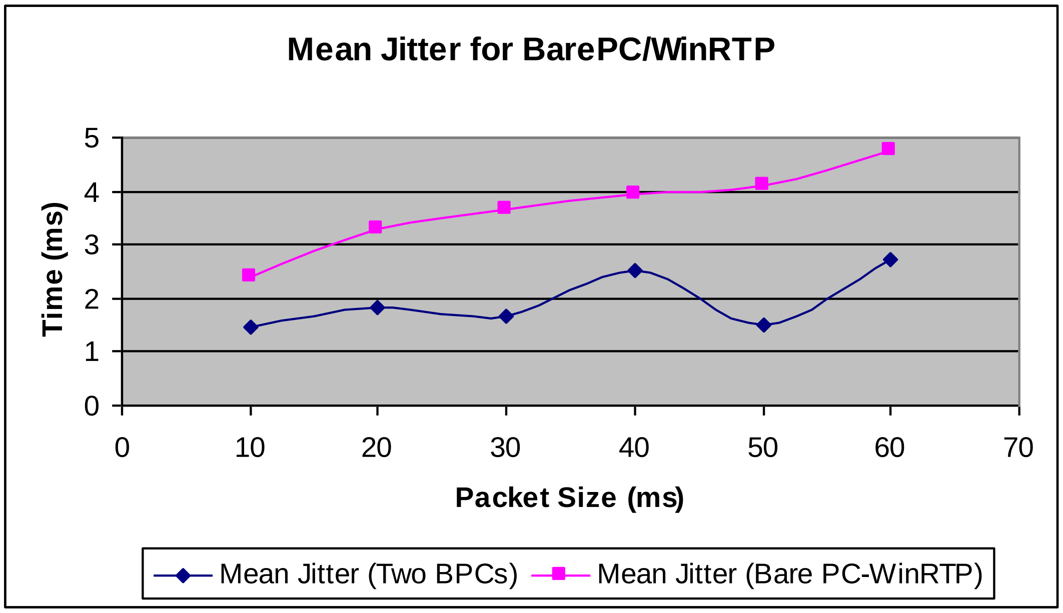

Figure 34: Mean jitter

Figure 35 shows the variation in the maximum packet inter-arrival gap (max delta), maximum and mean jitter values over a period of 1 hour for a bare PC-to-bare PC connection with a fixed packet size of 20 ms and fixed delay jitter buffer size of 100 ms. Notice that the network conditions remained relatively stable during the period of measurement.

In Figure 36, we show the end-to-end delays over the Internet for various voice packet sizes with a fixed delay jitter buffer size of 100 ms. The end-to-end delays vary from 100 ms to 450 ms. Delays over 400 ms are unacceptable, while those under 150 ms are not noticeable. During these experiments, we had the participants rate the quality of the calls as poor, acceptable, good or excellent; this roughly corresponds to MOS (mean opinion score) ratings of less than 2, 2-2.5, 3-3.5, 4 or greater, respectively (this scale assumes implicit rounding of MOS values). However, participants did not observe a significant drop in voice quality even with the larger delays, and they typically assigned ratings ranging from good to acceptable. In this case, we were unable to compare the performance of the WinRTP softphone under the same conditions. This experiment suggests that voice quality achieved by a bare PC softphone under marginal to poor network conditions is adequate, although more studies are needed to reach a definite conclusion.

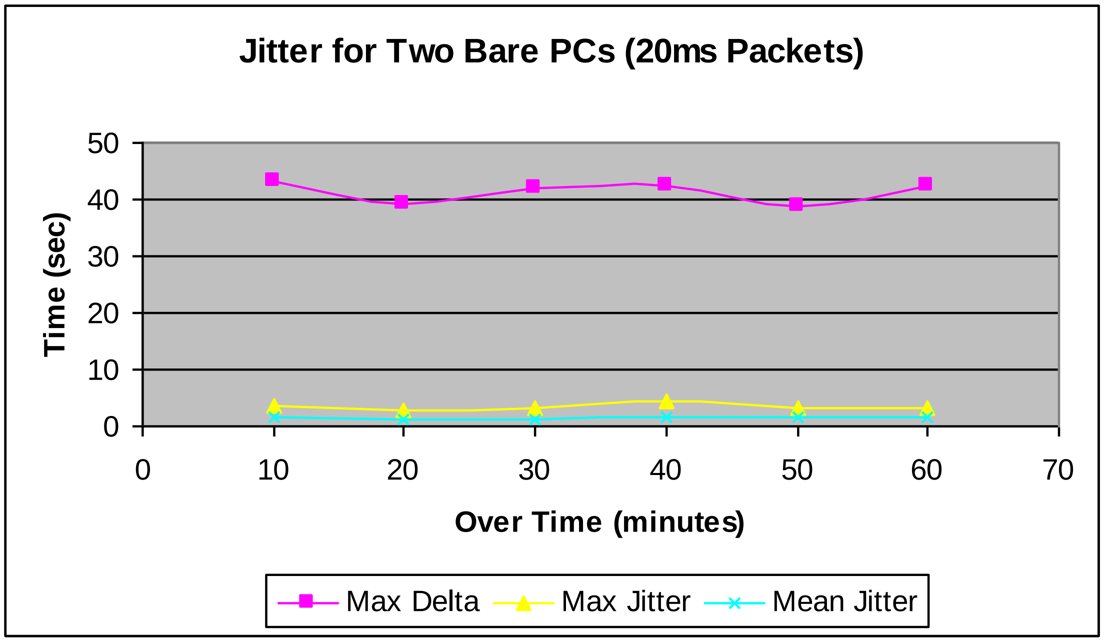

Figure 35: Max packet inter-arrival gap (max delta), max jitter, and min jitter

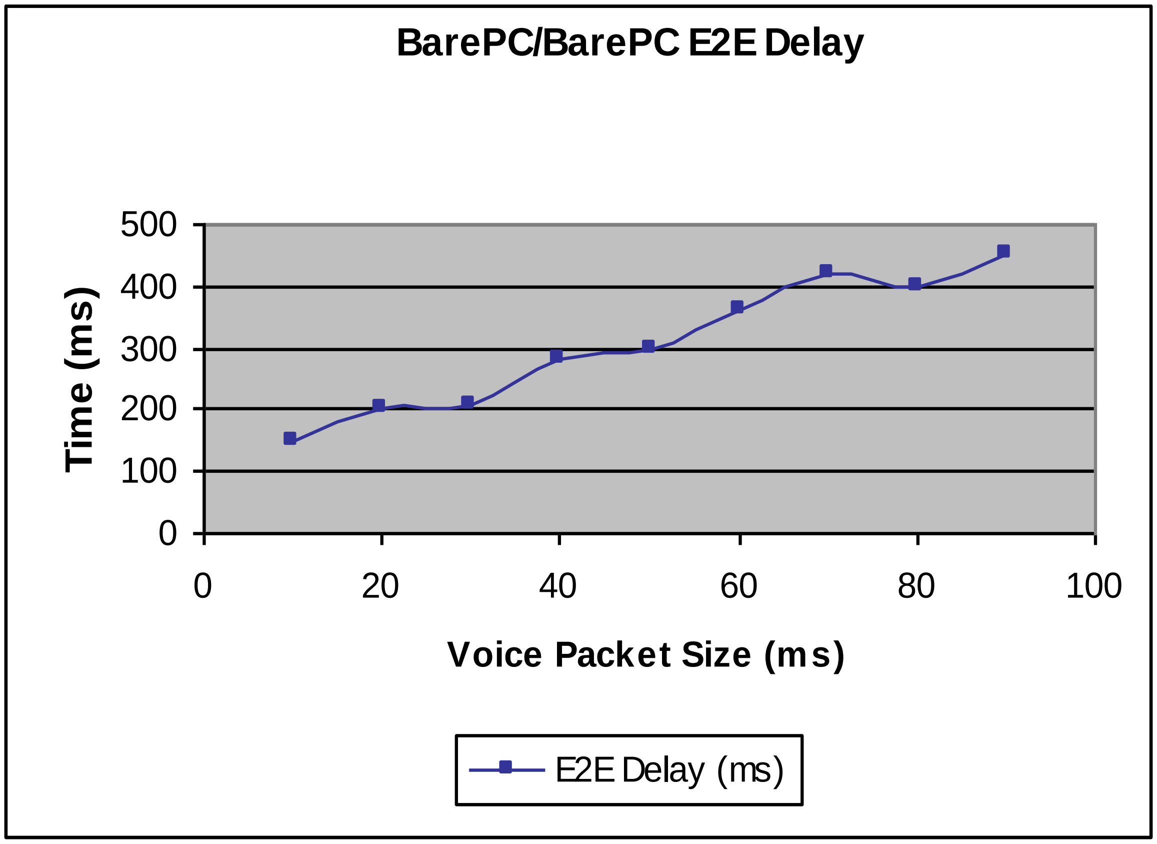

Figure 36: End-to-end delay

4.3. Performance with VoIP Security

We conducted several experiments to evaluate the performance impact of adding security mechanisms for VoIP to a bare PC softphone. The experiments used the same test LAN environment and hardware as in Section 4.1. Since there is no other traffic on the LAN, access delays due to collisions are negligible, and also, there is no significant network delay. Therefore, we assume that the values of jitter and max delta reflect the intrinsic overhead due to processing the voice packets and the addition of security mechanisms.

Calls were made between two bare PC softphones and between two WinRTP softphones. Data was collected for approximately 20 seconds and each experiment was repeated six times to ensure that the results were consistent. For each run, we first averaged the results for the voice streams in both direction, and then computed the average of these results over the six runs. All voice packets consisted of 20 ms of data and the length of the SHA-1 hash was 20 bytes. For convenience, we used the same AES key in both directions. These experiments did not consider replay protection as the hash did not include the RTP header.

Figures 37 and 38 show the maximum packet interarrival time (max delta), maximum (max) jitter and mean jitter for barePC to barePC calls using 20-ms voice packets without security and with security. In the latter case, the AES key sizes are 128, 192 and 256 bits, and the SHA-1 hash is 20 bytes. As expected due to additional processing with a larger key size, there is a constant insignificant increase (about 20 microseconds) in max delta for each 64-bit increment in key size as seen in Figure 37. Figure 38 shows that max jitter and mean jitter are not significantly different; mean jitter remains the same for all key sizes (about 100 microseconds), while max jitter with a 192 and 256-bit key is the same but negligibly (about 10 microseconds) less than that for 128-bit keys.

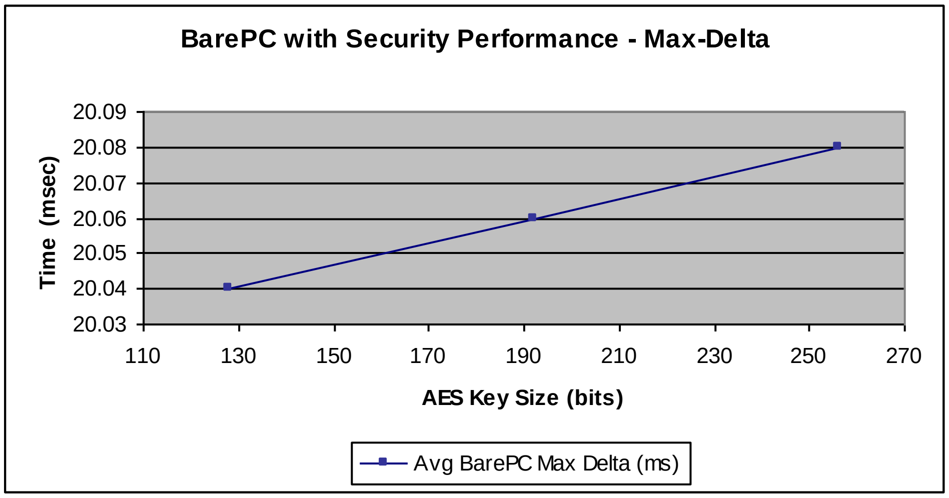

Figure 37: Max delta

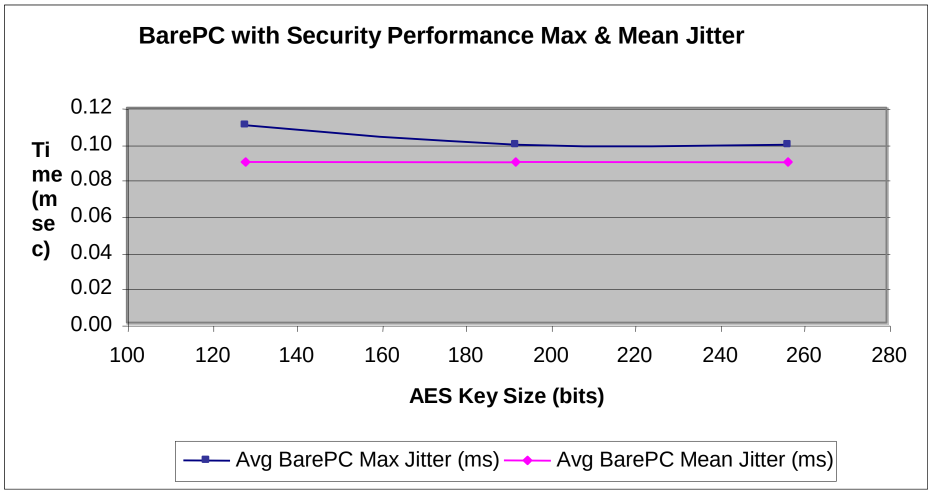

Figure 38: Max and mean jitter

Thus on the bare PC softphone, there is no significant difference in max delta, max jitter, and mean jitter due to encrypting voice or computing the hash even when increasing the AES key size. This performance benefit is a result of optimized processing and task scheduling on the bare PC softphone. The implication is that the cost to achieve higher levels of security with the bare PC softphone by increasing the AES key size is negligible.

We were unable to repeat the preceding experiment by adding a hash to the voice data on the WinRTP softphones as it was not possible to easily access the WinRTP code that needed to be modified. Instead, we conducted separate experiments to compare bare PC and WinRTP softphone performance. Specifically, we determined max delta, max jitter, and mean jitter for bare PC to bare PC and WinRTP to WinRTP calls when encryption of the 20-ms voice data (payload) only was performed for AES key sizes of 128, 192 and 256 bits without adding a SHA-1 hash. The results are shown in Figures 39, 40, and 41. Figure 39 shows that max delta for the bare PC softphone is constant and about 1 ms less than that for WinRTP softphone for all AES key sizes. The smaller gap for the 192 bit key size is due to a reduction in max delta for WinRTP with this key size. Figure 40 shows that max jitter on a bare PC softphone is about 100 microseconds less than that for a WinRTP softphone for all key sizes although a slight increase in the gap is seen for 256 bit keys. Figure 41 shows that the difference between mean jitter values for the two softphones is very small, with a slight increase occurring in the 256 bit keys. The results indicate that a bare PC softphone performs better, and also shows less variability in max delta and jitter values than a WinRTP softphone for all AES keys sizes.

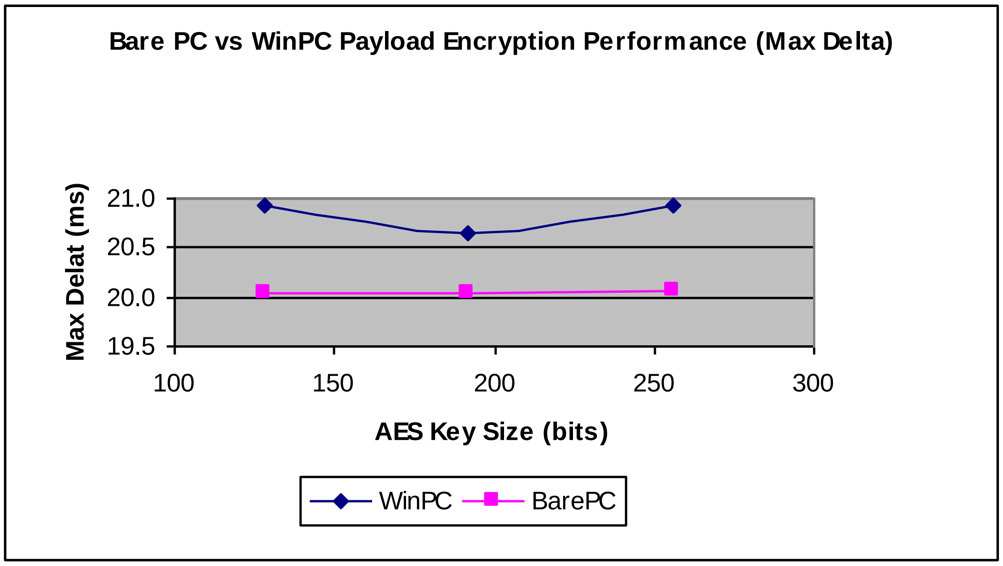

Figure 39 Max delta for bare PC to bare PC and WinRTP to WinRTP

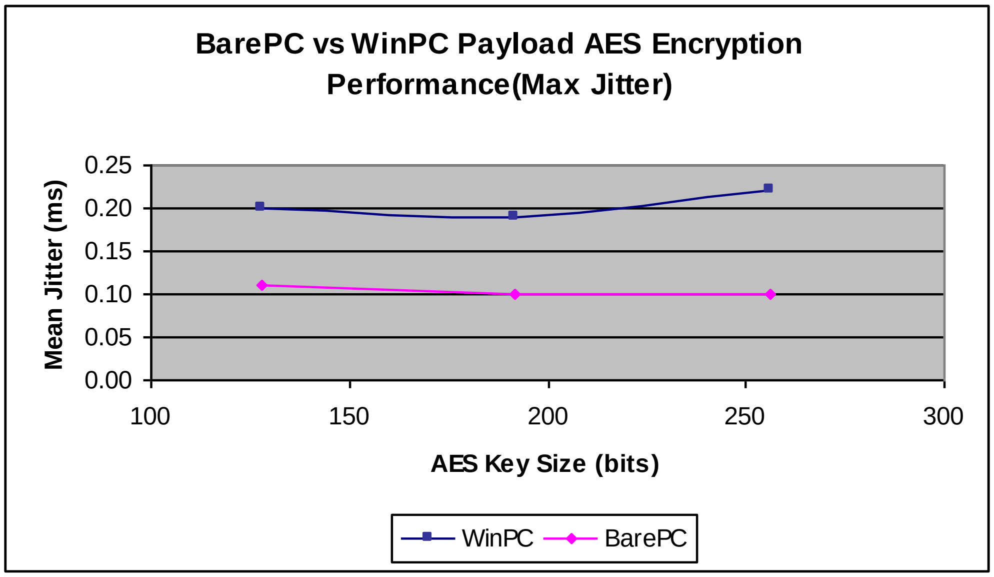

Figure 40: Max jitter for bare PC to bare PC and WinRTP to WinRTP

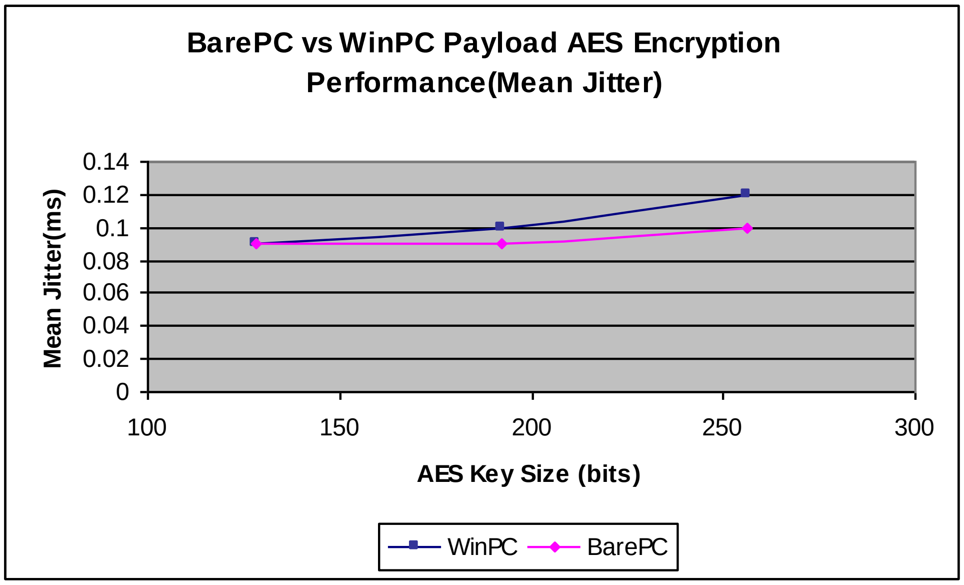

Figure 41: Mean jitter for bare PC to bare PC and WinRTP to WinRTP

In Figure 42, we compare handshake performance between bare PC softphones and between WinRTP softphones when exchanging 128, 192 and 256 bit AES keys. Although the total time increases by about 20 ms for each increase in the AES key size, the time for the bare PC softphone is about 200 ms less than that for the WinRTP softphone for all key sizes. This constant reduction in overhead reflects the benefits of optimal processing done on the bare PC softphone.

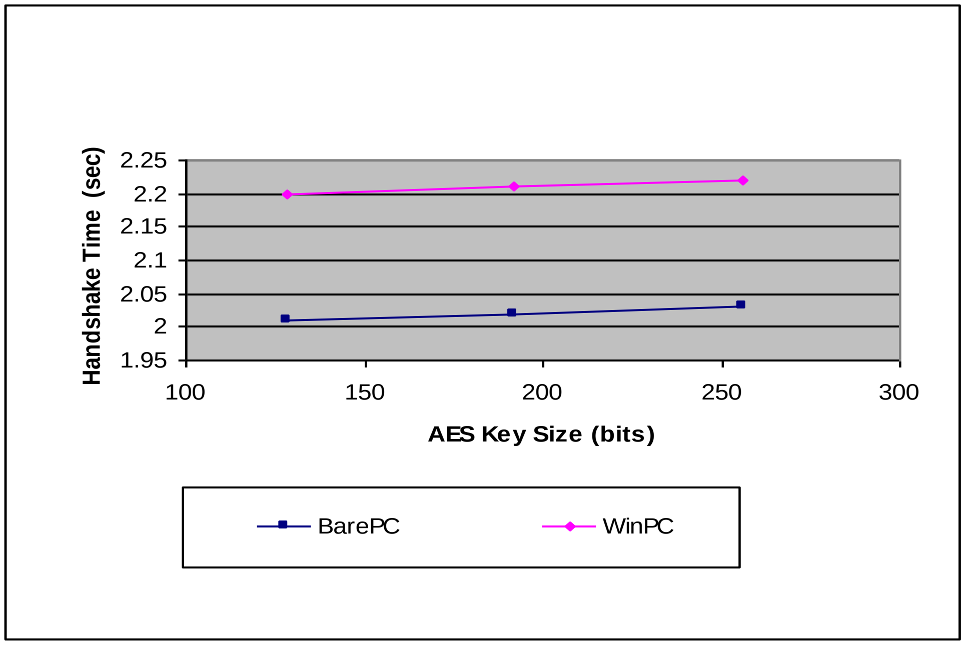

Figure 42: Total time for the handshake to exchange an AES key

Figure 43 shows the time for the various components of the handshake between bare PC softphones and between WinRTP softphones at the sender and the receiver using a 256-bit RSA key, a 256-bit AES key and a 20 byte SHA-1. While AES key generation and computing the SHA-1 take minimal time, the bulk of processing time involves RSA encryption and decryption. The total time for the bare PC softphone at the sender and receiver is about 0.5 seconds less than that for the WinRTP softphone. This difference illustrates the advantage of using optimized bare PC softphones for secure P2P VoIP systems.

Handshake Step	Handshake Sub Step	B2B AVG (ms)	W2W AVG (ms)
At Sender			
Generate AES		0	0
Process AES			
	Compute SHA-1	0	0
	Encrypt SHA-1	736	782
	Encrypt AES	554	588
Recover AES			
	Decrypt AES	203	232
	Recompute SHA1	0	0
	Decrypt SHA1	556	591
	Compare SHA1	0	0
At Receiver			
Recover AES			
	Decrypt AES	195	254
	Recompute HA1	0	0
	Decrypt SHA1	589	688
	Compare SHA1	0	0
Process AES			
	Compute SHA1	0	0
	Encrypt SHA1	697	818
	Encrypt AES K	584	685

Figure 43: Time for various components of the handshake

In Figure 44, we compare the time to generate 256, 512 and 1024 bit RSA keys on bare PC and WinRTP softphones. The benefit of using a barePC softphone is greatest when generating 1024 bit keys. However, these results indicate that even with optimized processing on the bare PC softphone, RSA key generation overhead with larger key sizes is significant and may not be acceptable unless a high level of security is desired.

Figure 45 compares the total time for generating between 10 to 100 keys on bare PC and WinRTP softphones. The time to generate 100 keys is about 3 minutes less on a bare PC softphone than on a WinRTP softphone. Generation of multiple keys could be done in the background and the faster completion time of the bare PC softphone suggests that key generation will have a lesser effect on voice calls and other applications running simultaneously.

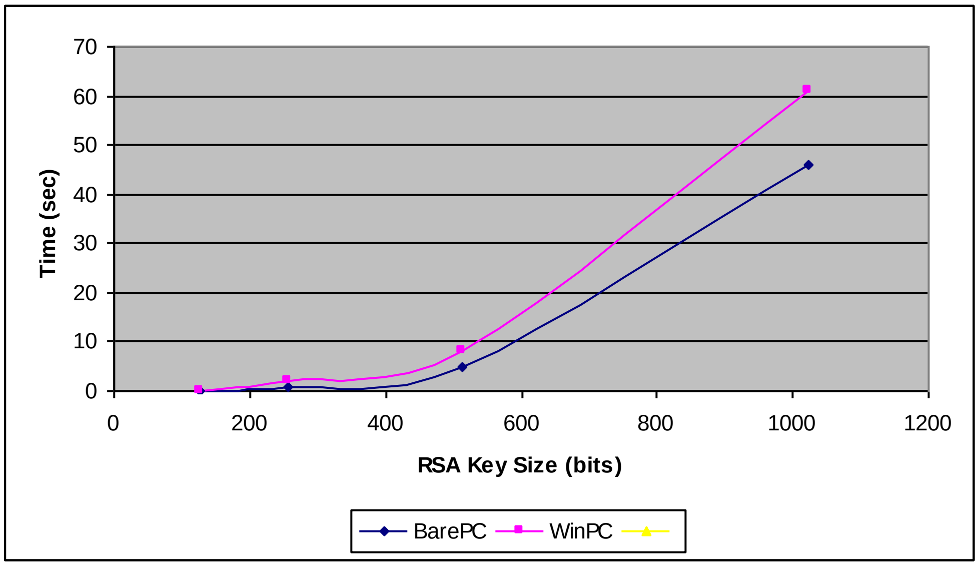

Figure 44: RSA key generation time for various RSA key sizes

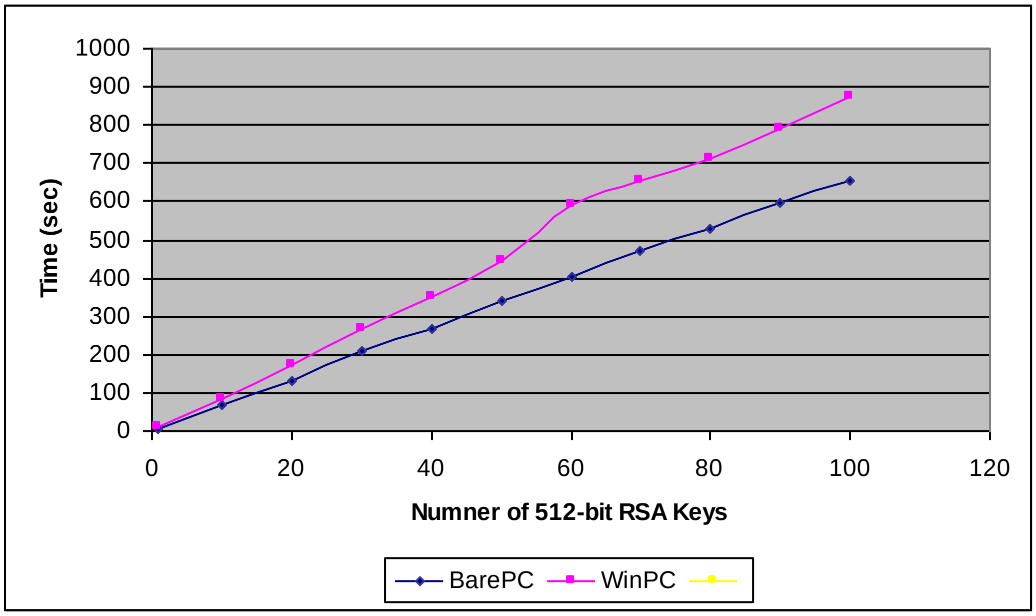

Figure 45: Total time for generating 10-100 RSA keys on barePC and WinRTP

Chapter 5. Conclusion

This dissertation has investigated using the bare PC (OS-less) computing environment for peer-to-peer VoIP. We described the design, architecture and implementation of a bare PC softphone and conducted experiments to measure its performance in LAN and on the Internet. We also compared the performance of a bare PC softphone and a Windows-based WinRTP softphone, and examined the impact on performance due to adding security mechanisms for VoIP.

We discussed several design and architectural features unique to a bare PC softphone including optimized task scheduling, zero copy buffering and cross-layer design. Optimized task scheduling results in better CPU utilization and minimizes intrinsic delays in processing packets. Zero copy buffering minimizes overhead and facilitates communication between network layers. The use of polling instead of interrupts also contributes to better performance. In addition, the record and playback functions for the bare PC softphone are synchronized, which helps to improve its efficiency.

The LAN and Internet experiments indicate that the performance of a bare PC softphone is better than that of the OS-based WinRTP softphone. We compared call quality by measuring delay, jitter, packet loss and the MOS. The bare PC softphone has lower jitter than the WinRTP softphone even when packet sizes are larger. The bare PC softphone was also found to have adequate performance under marginal to poor network conditions.

We conducted additional experiments suggesting that the bare PC softphone provides acceptable call quality under heavy system load conditions without or with background traffic on a LAN. In contrast, a WinRTP softphone degraded under a lower system load. Finally, we found that it is possible to obtain excellent to good call quality on a switched Ethernet LAN with no routers by using packets containing only an Ethernet header. The resulting savings in bandwidth could be used to support more calls or run other applications on the LAN.

Bare PC softphones support standard security mechanisms. A secure two-way handshake based on RSA is used to exchange the AES key. Thereafter, the voice data is encrypted using AES, and authenticity and integrity of voice data is guaranteed by use of an encrypted SHA-1 hash. Comparison of the performance of bare PC and WinRTP softphones after adding security mechanisms indicates that the bare PC is more efficient and able to provide better voice quality as it has less variation in maximum packet interarrival time and less jitter than a WinRTP softphone. Bare PC handshake performance is also better than a WinRTP softphone since it takes less time to complete. Bare PC softphones can also be used to achieve a higher level of security than WinRTP softphones since they have less overhead for larger AES key sizes. Generation of multiple RSA keys can also be done faster on bare PC softphones compared to WinRTP softphones.

The bare PC softphone can run alone, or concurrently with other bare PC applications, and it can be used to communicate with the OS-based WinRTP softphone. Also, the bare PC softphone is capable of functioning like a conventional OS-based softphone, since it can be used to connect seamlessly to an existing home, business or campus network, and thus to the Internet. Furthermore, since the bare PC softphone can run on older Intel-386 based PCs, it could serve as a communication tool in situations where high-speed Internet connectivity is available but PCs capable of supporting a modern OS required to run today’s multi-featured softphones are scarce. It should also be noted that, in principle, the AO for a bare PC softphone can be run on any device with an Intel 386 (or above) based architecture. Bare PC softphones inherit the advantages of bare PC computing, which include simplicity, efficiency and inherent security. This research suggests that bare PC softphones are an attractive option for direct efficient communication between peers while providing customizable security based on personal preference.

APPENDICES

Appendix A: Bare PC Softphone Guide

1. Compilation Environment

The bare PC softphone is written entirely in C++. The softphone also uses existing bare PC C++ API calls to interface with the hardware. A bare PC C++ API call invokes a C call, and that in turn invokes an assembly call. The compiling environment uses batch files to compile and link the softphone application with the necessary bare PC modules [8]. Visual Studio C++ compiler (batch mode), MASM 6.11 assembler, and Turbo assembler compilers are used to create executable modules. We have written batch files to do compilation and linking for boot and loader programs and the VoIP softphone application. All command files are executed in the root directory for the softphone as shown in Figure 46.

\VoIPSec	Root directory of the softphone
\aes\	AES code
\arp\	Address Resolution Protocol code
\audio\	Audio card drivers code
\bin\	Compiler and linker executables
\dosclib\	DOSC object files
\ethernet\	Ethernet protocol code
\G711Codec\	G.711 codec code
\hshake\	Handshake code

\interfaces\	DOSC interface files
\ip\	Internet Protocol code
\jitter\	Jitter buffer code
\MASS\	Assembler executable
\memorymap\	DOSC memory map files
\rp\	Record and playback code
\rsa\	RSA key generation code
\rtp\	Real-Time Transport Protocol code
\sha1\	Sha1 code
\tcp\	TCP code
\udp\	UDP code
\vsec\	VoIP security code
\webserver\	C++ main and tasking code

Figure 46: Softphone Directory Structure

Figure 47 shows lines of code information for the bare PC softphone application.

File Name	**File Type**	**Uncommented Lines of Code**
VoIPSec		257
Aes	H	44
ARP	H	60
Audio	H	105
Ethernet	H	119
G711Codec	H	4186
Hshake	H	82
IP	H	45
Jitter	H	192
Rp	H	26
Rtp	H	45
sha1	H	26
Tcp	H	277
Udp	H	29
VSEC	H	36
webserver	H	610
Aes	CPP	1200
ARP	CPP	586
Audio	CPP	439
Ethernet	CPP	848
G711Codec	CPP	84
Hshake	CPP	597
interfaces	CPP	805
IP	CPP	307
Jitter	CPP	609
Rp	CPP	271
Rtp	CPP	199
sha1	CPP	157
Tcp	CPP	2696
Udp	CPP	179
VSEC	CPP	386
webserver	CPP	5132
Total		**20634**

Figure 47: Lines of Code

2. How to Boot, Load and Execute the Softphone Application

In order to load and execute the bare PC softphone application, one needs to place a boot program, a loader program, and the softphone application on a bootable device such as a floppy diskette. The boot program enables bare PC applications such as the softphone to be executed after booting is completed. You need to follow standard PC boot procedures and power up the PC with the boot device in the boot drive. During the boot process, the loader program will load an AOA interface menu into internal memory for execution. Using this interface menu, a user can load the softphone application from the same boot device and then execute it.

References

[1] Upkar Varshney, Andy Snow, Matt McGivern and Christi Howard. "Voice Over IP." *Communication of ACM,* January 2002/Vol 45, No. 1.

[2] Cole, R. G. and Rosenbluth, J. H. "Voice Over IP Performance Monitoring. " *ACM SIGCOMM, Computer Communication Review,* 2001.

[3] A. S. Tanenbaum, J. N. Herder and H. Bos. "Can we make Operating Systems Reliable and Secure?" *IEEE Computer,* May 2006.

[4] D. R. Engler and M. F. Kaashoek. "Exterminate all operating system abstractions." *5th Workshop on Hot Topics in Operating Systems,* 1995.

[5] R. K. Karne, K. V. Jaganathan and T. Ahmed. "DOSC: Dispersed Operating System Computing." *OOPSLA,* October 2005, San Diego, CA, pp. 55-61.

[6] R. K. Karne, R. Gattu, R. Dandu and Z. Zhang. "Application-oriented Object Architecture: Concepts and Approach." *In Proc. IASTED,* October 2002.

[7] R. K. Karne. "Application-oriented Object Architecture: A Revolutionary Approach." *HPC Asia*, December 2002.

[8] R. K. Karne, K. V. Jaganathan and T. Ahmed. "How to run C++ Applications on a bare PC." *In Proc. SNPD*, 2005, pp. 50-55.

[9] M. Hillenbrand, J. Gotze and P. Muller. "Voice over IP – Considerations for a next Generation Architecture." *In Proc. EUROMICRO,* 2005.

[10] G. Borriello and R. Want. "Embedded Computation Meets the World Wide Web." *CACM, Vol. 43, No. 5*, May 2000.

[11] S. T. King, G. W. Dunlap, and P. M. Chen. “Debugging operating systems with time-traveling virtual machines.” *In Proc. USENIX*, 2005.

[12] H. Long, R. K. Karne, S. Girumala, A. L. Wijesinha, and G. H. Khaksari. “Design Issues for a Bare PC Web Server.” *In Proc. SNPD,* 2006.

[13] S. A. Baset and H. Schulzrinne. “An analysis of the Skype peer-to-peer internet telephony protocol.” *In Proc. IEEE INFOCOM*, 2006.

[14] O. Hagsand, I. Marsh, and K. Hanson. “Sicsophone: A low-delay Internet telephony tool.” *In Proc. EUROMICRO*, 2003.

[15] R. C. Hsu, C-T Liu, W-P Huang, and J-J Yang. “An embedded software approach for the development of SIP-based VoIP Server.” *In Proc. APSEC,* 2004.

[16] M. Manousos, S. Apostolacos, I. Grammatikakis, D. Mexis, D. Kagklis, and E. Sykas. “Voice quality monitoring and control for VoIP.” *IEEE Internet Computing*, July/August 2005.

[17] K. Singh and H. Schulzrinne. “Peer-to-peer Internet telephony using SIP.” In *Proc. NOSDAV*, 2005.

[18] S. Zeadally and F. Siddiqui. “Design and Implementation of a SIP-based VoIP Architecture.” *In Proc. AINA*, 2004.

[19] T. J. Walsh and R. Kuhn. “Challenges in securing Voice over IP.” *IEEE Security and Privacy*, 2005.

[20] K. Singh and H. Schulzrinne. “Peer-to-Peer Internet Telephony using SIP.” *In Proc. NOSDAV,* Stevenson, WA, June 2005.

[21] TinyOS Community Forum. *http://www.tinyos.net*

[22] D. R. Engler. “The Exokernel Operating System Architecture.” *Ph.D. thesis, MIT,* October 1998.

[23] The OS Kit Project. *http://www.cs.utah.edu/flux/oskit*

[24] Vovida. “WinRTP*.”* *http://www.vovida.org*

[25] G. H. Khaksari, A. L. Wijesinha, R. K. Karne, L. He and S. Girumala. “A Peer-toPeer Bare PC VoIP Application.” *In Proc. IEEE CCNC*, Jan 2007.

[26] S. Zeadally and F. Siddiqui. “Design and Implementation of a SIP-based VoIP Architecture.” *In Proc. AINA, 2004.*

[27] M. Hillenbrand and G. Zhang. “A Web Services Based Framework for Voice over IP.” *In Proc. EUROMICRO,* 2004.

[28] M. Ghanassi and P. Kabal. “Optimizing Voice-over-IP Speech Quality using Path Diversity.” *International Workshop on Multimedia Signal Processing,* October 3-6, 2006, Victoria, BC, Canada.

[29] J. Rosenberg, L. Qiu, and H. Schulzrinne. “Integrating Packet FEC into Adaptive Voice Playout Algorithms on the Internet.” *In Proc. INFOCOM,* 2000.

[30] W. H. Chiang, W. C. Xiao and C. F. Chou. “A Performance Study of VoIP Applications: MSN vs. Skype.” *In Proc. MULTICOMM,* 2006.

[31] W. Jiang and H. Schuzrinne. “Comparison and Optimization of Packet Loss Repair Methods on VoIP Perceived Quality under Bursty Loss.” *In Proc. NOSSDAV,* 2002.

[32] R. Ramjee, J. Kurose, D. Towsley and H. Schulzrinne. “Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks.” *National Science Foundation.*

[33] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. “RTP: A Transport Protocol for Real-Time Applications.” *RFC 3550. http://www.ietf.org/rfc/rfc3550.txt*

[34] Ethereal. “Ethereal Network Analyzer.” *http://www.ethereal.com*

[35] IntelliCorp. “Intel 82801 EB (ICH5) I/O Controller Hub: AC ’97 Programmers Reference Manual.” April 2003, Document Number: 252751-001

[36] Naval Research Laboratory. “MGEN: Multi-Generator.” *http://cs.itd.nrl.navy.mil/work/mgen*

[37] J. Rosenberg, Dynamicsoft, H. Schulzrinne, Columbia U., G. Camarillo, Ericsson, A. Johnston, WorldCom, J. Peterson, Neustar, R. Sparks, M. Handley, ICIR, E. Schooler, AT&T. “ SIP RFC 3261.” *http://www.faqs.org/rfcs/rfc3261.html*

[38] Gholam. H. Khaksari, A. L. Wijesinha, R. K. Karne, Q. Yao, and K. Parikh. “A VoIP Softphone on a Bare PC.” *In Proc. WORLDCOMP ESA*, 2007.

[39] Hansen, Markus; Hansen, Marit; Moeller, Jan; Rohwer, Thomas; Tolkmit, Carsten, Waack, Henning. “Developing a Legally Compliant Reachability Management System as a Countermeasure against SPIT.” *Third Annual VoIP Security Workshop, Berlin,* June 2006.

[40] T. Dierks, C. Allen. “The TLS Protocol 1.0.” *http://www.apps.ietf.org/rfc/rfc2246.html*

[41] R. Housley, W. Ford, W. Polk, D. Solo. “Internet X.509 Public Key Infrastructure: Certificate and CRL Profile.” *IETF RFC 2459*. January 1999.

[42] P. Zimmerman, A. Johnston, J. Callas. “ZRTP: Extensions to RTP for Diffie-Hellman Key Agreement for SRTP. AVT WG Internet-Draft.” 2006 March 5. *http://www.ietf.org/internet-drafts/draft-zimmermann-avt-zrtp-01.txt*

[43] D. R. Kuhn, T. J. Walsh, S. Fries. “Security Considerations for Voice Over IP Systems.” Recommendations of the National Institutes of Standards and Technology.” January 2005. *http://csrc.nist.gov/publications/nistpubs/800-58/SP800-58-final.pdf*

[44] M. Baugher, D. McGrew, Cisco Systems, Inc., M. Naslund, E. Carrara, K. Norrman, Ericsson Research. “SRTP: Secure Real-time Transport Protocol.” *http://www.rfc-archive.org/getrfc.php?rfc=3711*

[45] H. Lipmaa, P. Rogaway, and D. Wagner. “CTR-mode encryption.” *In 1st NIST Workshop on Modes of Operation,* 2000.

[46] J. Rosenberg, J. Weinberger, dynamicsoft, C. Huitema, Microsoft, R. Mahy, Cisco. “STUN: Simple Traversal of UDP through Network Address Translators.” RFC 3489, March 2003. *http://www.ietf.org/rfc/rfc3489.txt*

[47] E. Rescorla, Network Resonance. “Interactive Connectivity Establishment (ICE).” *http://tools.ietf.org/id/draft-rescorla-mmusic-ice-lite-00.txt*

[48] Shakkottai, S., Rappaport, T.S., Karlsson, P.C. “Cross-layer design for wireless networks.” *Communications Magazine, IEEE, Volume 41, Issue 10,* Oct 2003 Page(s): 74 – 80

[49] J. Callas, L. Donnerhacke, H. Finney and R. Thayer. “OpenPGP Message Format.” *RFC 2440. http://www.ietf.org/rfc/rfc2440.txt*

[50] D. Reichl. “CSHA1 - A C++ class implementation of the SHA-1 hash algorithm.” 2002. *http://www.codeproject.com/cpp/csha1.asp*

[51] G. Anescu. “A C++ Implementation of the Rijndael Encryption/Decryption method.” 2001. *http://www.codeproject.com/cpp/aes.asp*

[52] RSA Public-Key Cryptography. *http://www.efgh.com/software/rsa.txt, http://www.efgh.com/software/rsa.htm*

Curriculum Vitae

Gholam H. Khaksari Curriculum Vitae

Department of Computer Science
Morgan State University
1700 East Cold Spring Lane
Baltimore, MD 21251
USA

voice: (443) 885 1395
email: khaksari@jewel.morgan.edu

Education

2007	*D.Sc. Applied Information Technology,* Towson University, Towson, MD
1992	*M.Sc. Computer Science,* John's Hopkins University, Baltimore, MD
1984	*B.Sc. Computer Science,* Youngstown State University, Youngstown, OH
1981	*B.Sc. Civil Engineering,* Youngstown State University, Youngstown, OH

Research Interests

- Voice over IP (VoIP)
- Bare PC Computing
- IT Security
- Artificial Intelligence (AI) & Decision Support Systems (DSS)
- Database Management Systems & Data Mining

Publications

[1] Gholam H. Khaksari, Alexander L. Wijesinha, Ramesh K. Karne, and Qi Yao, "A VoIP Softphone on a Bare PC" to appear *in proceeding of 2007 Worldcomp ESA, Las Vegas, Nevada, USA.*

[2] Gholam. H. Khaksari, A. L. Wijesinha, R. K. Karne, L. He and S. Girumala. "A Peer-to-Peer Bare PC VoIP Application" *in proceeding of 2007 IEEE CCNC, Las Vegas, Nevada, USA.*

[3] Long He, Ramesh K. Karne, Alexander L. Wijesinha, Sandeep Girumala, and Gholam H. Khaksari, "Design Issues for a Bare PC Web Server" *In proceeding of 2006 Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*

[4] Gholam H. Khaksari, et al. "Expert Drug Trafficking Detection System" *In proceeding of 1990 Westinghouse R&D Symposium, Baltimore, MD.*

[5] Gholam H. Khaksari, "Westinghouse Fault Isolation System" *in proceeding of 1989 Westinghouse R&D Symposium, Pittsburgh, PA.*

[6] Gholam H. Khaksari, "Expert Diagnostic Systems" *in proceeding of 1988 International Conference on Industrial and Engineering Applications of Artificial Intelligence & Expert Systems, Knoxville, Tennessee, USA.*

[7] Gholam H. Khaksari, "Generic Fault Isolation System" *in proceeding of 1987 Westinghouse R&D Symposium, Baltimore, MD.*

Research Experience

2004-present

Doctoral Research: VoIP on a Bare PC
Department of Computer Science
Towson University, Towson, MD
Advisors: Alexander L. Wijesinha Ph.D., Ramesh K. Karne Ph.D.

This dissertation proposes a novel VoIP softphone architecture for a bare Intel-386 (or above) based PC without an operating system. First, we provide an overview of bare PC computing and note the advantages of a bare PC softphone including its inherent simplicity and ability to provide secure, reliable and efficient voice communication. Next, we discuss the design of a bare PC softphone and describe its architecture and implementation. We then present performance measurements from LAN and Internet experiments, which consider delay, jitter, packet loss, and MOS. They indicate that a bare PC softphone has less jitter, less security overhead, and is able to sustain larger voice packet sizes and a heavier load than a WinRTP softphone while maintaining acceptable call quality with or without background traffic. A bare PC softphone also has acceptable call quality when running Voice over Ethernet (voice packets with Ethernet headers only) on a LAN.

1984-1990

Research and Development: Hardware Diagnostic using AI
Aerospace Division
Westinghouse Electric Corporation
Baltimore, MD

The rapid increase in the complexity, use and dependency of our society on state-of-the-art electronic systems has produced the need for better methods of maintaining the operational readiness of these complex systems. Diagnostics is the process of detecting, localizing, isolating and fixing the failures in such sophisticated systems

The Expert Diagnostic System (EDS) is an innovative application of Artificial Intelligence (AI) techniques to hardware failure diagnosis that provides less experienced technicians with diagnostic knowledge and understanding attributed to experts

Experts Shallow Knowledge is captured and represented in the form of rules, and Hardware Deep Knowledge is captured and represented in a schematic data model describing the signal flow and connectivity of the system components and the related testability information. The EDS also captures and maintains system failure history and uses this knowledge along with shallow and deep knowledge to adaptively diagnose hardware failures

Academic Activities

- Course development
- Student advising
- NSF REU program
- Weekly technology seminars presentation
- Technology committee member
- Curriculum assessment & development committee member

Teaching Experience

2005-present Lecturer
Department of Computer Science
Morgan State University
Baltimore, MD

Courses developed & taught:
Internetworking using TCP/IP (COSC 357)
Object Oriented Programming (COSC 230)
Operating Systems (COSC 354)
Computer Architecture (COSC 243)
Network Security Fundamentals (COSC 358)
Introduction to Computer Science (CCCOSC 111)

2001-2005 Lecturer

Department of Information Sciences
Morgan State University
Baltimore, MD

Courses developed & taught:
Decision Support Systems (INSS693)
System Analysis and Design (INSS370)
Database Management Systems (INSS380)
E-Commerce (INSS430)
Introduction to Information Systems (INSS360)
Computer Based Information Systems (INSS141)

2004-present Adjunct
Department of Computer Science
Towson University,
Towson, MD

Courses taught:
Fundamentals of Data Structures and Algorithm Analysis (COSC 501)
Project Management (CIS 479)
Information & Technology for Business (CIS 111)
Introduction to Computer Science (COSC 236)

Industry Experience

1997-2000 Customer Services Director
Powerize.com

As the director of customer services, I acted in partnership with product manager, director of engineering, director of sales as well as customer service staff in definition, design, development, testing, documentation, installation, training and support of software products

Responsibilities included: Development of software product installation procedures and technical documentation, development and implementation of technical training courses, setup of call center and workflow tools and procedures, hiring and managing of customer support representatives to support software products and web services, consultation with customers to identify new and enhanced product features

1991-1997 Senior Software Engineer
Convera Inc.

As a Senior Software Engineer, I worked with a top-notch software engineering team responsible for design, development, and testing of RetrievalWare product, a pioneering distributed client server search engine

	and document management system *Responsibilities included:* Analysis, design, coding, testing and integration of software components such as a document parser, document summarizer, ranking algorithms, profiler, and query by Example (QBE), as well as system integration and localization of RetrievalWare involving API, TCP/IP, HTML, CGIs, Web servers, RDBMS, NT, Windows, and UNIX, and C
1984-1991	Senior Software Engineer Westinghouse Electric Corp.
	As a Senior Software Engineer, I worked with system engineering, software engineering, and test station equipment development and design teams responsible for development, testing and integration of diagnostic software used to support DOD Aerospace products *Responsibilities included:* Application of Artificial Intelligence (AI) techniques to Fault Isolation and Testing (FIT), development, coding, testing and integration of diagnostic Ada software for ALQ-131 and AWACS using MIL-STD 2167A

Programming Experience

Languages: C, C++, Lisp, Pascal, Fortran, Prolog, ADA, Atlas, Assembly
Operating Systems: Unix, Linux, VMS, VM, Windows
Databases: Oracle, MySQL, SQL Server

Honors, Awards and Patents

1990	Signature award of excellence, Westinghouse
1989	Software patent disclosure award, Westinghouse
1987	First place IR&D symposium award, Westinghouse