A VoIP Softphone on a Bare PC

Gholam H. Khaksari, Alexander L. Wijesinha,
Ramesh K. Karne, Qi Yao, and Ketan Parikh
Department of Computer & Information Sciences
Towson University
Towson, Md 21252

Abstract - *We describe the architecture, design, and implementation of a VoIP softphone that runs on a bare Intel-386 (or above) based PC. We compare the performance of bare PC and WinRTP softphones on the Internet by determining call quality and measuring the values of jitter, delay, and packet loss. We also conduct experiments on a LAN to test the limits of the bare PC softphone under heavy load conditions, and investigate the use of voice packets with Ethernet headers only (Voice over Ethernet) to minimize voice bandwidth. The results indicate that a bare PC softphone on the Internet is associated with less jitter than the WinRTP softphone. It is also able to sustain a heavier load on a LAN than the WinRTP softphone while maintaining acceptable call quality. We obtained excellent to good call quality with bare PC softphones running Voice over Ethernet.*

Keywords: VoIP, bare PC, softphone, call quality, network protocols.

1 Introduction

Existing VoIP systems are dependent on operating system (OS) services. For example, both the Skype softphone [1] and the user agents for the peer-to-peer VoIP adaptor [2] 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.

1.1 Bare PC Computing

Bare PC computing [3], 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. It is thus 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. Bare PC computing has many advantages for applications 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. Briefly, the idea in bare PC or dispersed operating system computing (DOSC) is that an operating system is unnecessary as the essential operating environment elements for execution of applications can be dispersed into the application itself. Such an application is referred to as an application object (AO) [4]. An AO is a self-contained and self-managed object that inherits all the benefits of the object oriented programming paradigm.

Embedded systems and bare PCs share some common features. For example, both types of systems do not use a disk drive, minimize use of hardware resources, eliminate unnecessary functionality, and have an optimized design. However, a bare PC is different from common types of embedded systems because a bare PC is capable of running any application and it 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). Also, since the bare PC approach eliminates the OS altogether, it goes further than earlier work on minimal OSs including Tiny OS [5], Exokernel [6], and OSKit [7].

1.2 Bare PC Softphone

In this paper, we describe the design and implementation of a softphone that runs on any Intel-386 (and above) based bare PC with no operating system, and investigate the effect of several optimizations on call quality and both network and system performance. This work is part of an effort to develop bare PC applications

including Web servers, Email clients, and SIP clients and servers together with supporting security protocols. The bare PC VoIP softphone application is designed as an AO. It fits on and is stored on a floppy disk, although in principle, any type of portable storage media such as a CD or USB flash drive can be used. An AO can boot, load, and run on any bare PC. It directly communicates with the hardware and does not use any hard disk. The AO programmer, who is in complete control of the bare PC and manages system resources, can design an application for performance, security, or with other desired features.

The bare PC softphone can run alone, or concurrently with other bare PC applications. Also, the bare PC softphone is capable of functioning like a softphone on an OS-based PC, since a bare PC is able 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 basic communication tool in situations where high-speed Internet connectivity is available, but PCs capable of supporting a modern OS such as Windows XP 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. An added benefit from a security viewpoint is that 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. We do not consider VoIP security issues in this paper.

In [8], we presented a prototype of a bare PC VoIP client and preliminary results from using it in a small test LAN in our laboratory. The bare PC softphone described in this paper has been enhanced for performance with the addition of a simple but efficient task-scheduling scheme. Furthermore, it uses a jitter buffer to improve call quality, and can continuously monitor performance at all levels of the system during a call. We also evaluated its performance by making calls over the Internet. In addition, we investigated the impact on performance due to generating a heavy load on the softphone (similar to one that would result from running several applications concurrently), without and with background traffic on a LAN. We also tested the feasibility of voice over Ethernet and its effect on call quality by using a MAC header only to transfer voice packets within a switched Ethernet LAN with no routers.

The rest of the paper is organized as follows. Section 2 describes the architectural features of the bare PC VoIP softphone, and Section 3 provides design and implementation details. Section 4 presents the experimental results. Section 5 discusses related work, and Section 6 contains the conclusion.

2 System Architecture

The bare PC VoIP softphone application system architecture shares many features with bare PC (OS-less) computing in general. A bare PC provides direct interfaces to the hardware and all other necessary operating elements are built-in to the application itself.

2.1 Hardware Interfaces

The basic hardware elements that support the bare PC softphone are CPU, memory, floppy drive, and Ethernet card (external 3Com 905CX network interface card or onboard Network Chip Intel 82540EM), onboard audio chip (AD1981B), keyboard, headphones, and display. The softphone AO manages the CPU and the memory available in the bare PC. We have developed the necessary standard interfaces to the above hardware elements, and drivers for the network cards and sound cards. In addition, we also have C++ interfaces [9] to tasks, interrupts, and exceptions. 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 [3]. These interfaces also enable the softphone to be run concurrently with other applications on the bare PC by adding more tasks.

2.2 Architecture

In a bare PC softphone, voice packets arrive into the Ethernet buffer and wait for the receive task to process them. The device drivers for the Ethernet card/chip receive incoming voice packets and store them in a circular list. Similarly, outgoing voice packets are kept in a circular list by the application. The circular lists are located in user address space. In this system, user space and system space are one and the same as the AO programmer controls the memory map for the softphone application.

Figure 1 shows the architecture for the bare PC VoIP softphone application. The main task (not shown) checks for arrival of network packets in the receive circular list and activates the receive (RCV) task to process these packets. The receive task does the Ethernet, IP, UDP, and RTP processing, and

places the audio frame pointers (to payload and RTP header) in the jitter buffer. The audio task requests a frame from the jitter buffer, does the G.711 decoding, and writes the decoded frame to the circular playout buffer. It also checks for voice frames in the microphone recording buffer, does the G.711 encoding followed by RTP, UDP, IP, and Ethernet processing before writing the outgoing Ethernet frame to the send circular buffer to be transmitted on the network.

Currently, we use only one audio task since there is only a single client on the bare PC. Additional tasks can be easily added for other softphone functions such as call waiting. The task structure is depicted in Figure 2, where softphone tasks are stored in the stack and then placed in run list when needed. Once the call is ended, the task is replaced in the stack. We use the same technique to add other tasks into the run list in order to provide a multi-tasking architecture. The programmer has total control over task management and is able to simplify scheduling by limiting the tasks running in the system.

It can be seen that the above architecture is very simple, and customized for the bare PC softphone; there is no functionality 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 difficult to implement in an OS-based system.

2.3 Optimizations

Implementation of network protocols such as RTP, UDP and IP are integrated with Ethernet processing. In particular, strict layering rules are not followed since the goal is optimal performance. For example, all the protocol processing for sending an outgoing packet is accomplished in a single step. Task scheduling in a bare PC is controlled by the AO, and the AO programmer chooses the scheduling technique.

The idea in optimal scheduling is as follows: when a task is not doing a useful work (no CPU processing), it should be suspended or returned to the main task. Thus, in the softphone, the receive task will return to 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 scheduling mechanism is implemented using a Karnaugh map table, in which the inputs are the tasks, and the output is the task to be scheduled. For example, the receive task and audio task have 00, 01, 10, 11 combinations. Its output combinations may be none, audio, receive, and receive, which indicates that the receive task has more priority than the audio task (a priority scheme is easily changed). This simple scheduling approach is efficient and enables optimal performance in a bare PC softphone.

Another 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. We also avoid receive interrupts for the network card and process transmit interrupts to acknowledge the interrupt controller. This improves performance and simplifies the design. The VoIP softphone described in [10] also avoids data copying and implements other optimizations. However, since it relies on OS support unlike the bare PC, the optimizations are necessarily limited by the underlying OS design.

3 Design and Implementation

The softphone application uses APIs for audio controller, network protocols, screen output, and keyboard input. The record and playback for this softphone are synchronized using the audio card recording and playback timing.

3.1 Audio Task

The audio task monitors the recording of the microphone data. It processes the data whenever the microphone buffer is filled, and then checks if there are packets in the jitter buffer to be processed. So every time a buffer is recorded, a frame (possibly a “silence” frame) is played out as well. This technique helps with minimizing the intrinsic delay by minimizing the distance between the read and write pointers in the playback buffer. It also simplifies the synchronization of the microphone buffer read pointer and the speaker buffer write pointer.

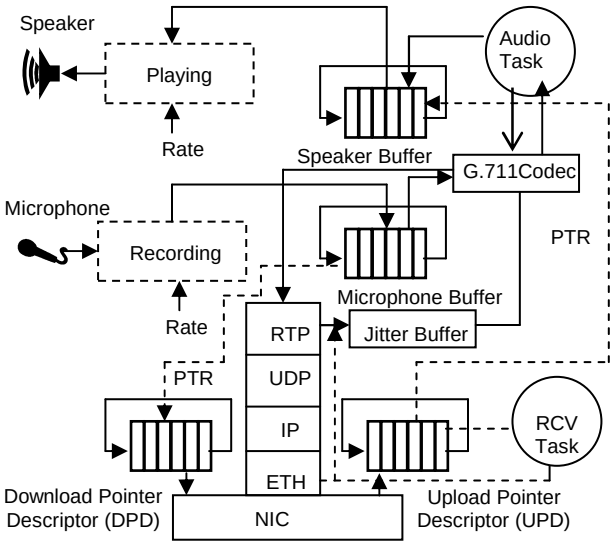

Figure 1: Softphone system architecture

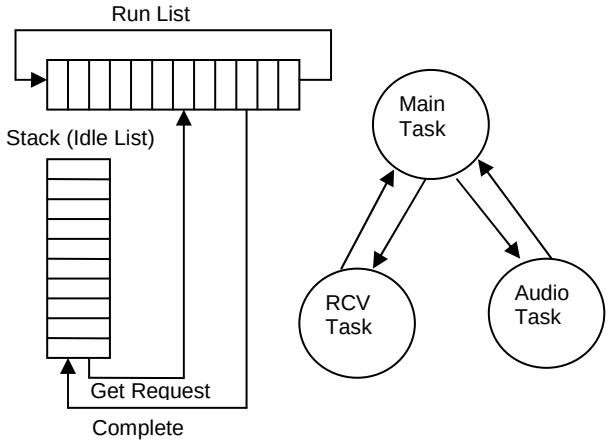

Figure 2. Softphone task structure

While synchronizing the playback and the recording, the audio task also monitors the condition of the jitter buffer. If no packets are arriving due to network conditions or any other failures, it will restore 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 an entire frame of recorded voice data per activation before suspending itself.

The audio card produces two 16-bit (stereo) voice samples every 125µs. The 16-bits for the right channel are ignored resulting in 16 bytes of raw voice data per ms. The G.711 codec performs 2:1 compression leaving 8 bytes of data per ms. Thus, each *t* ms voice sample is equivalent to 8**t* bytes of voice data. During playback, the right channel is restored by copying the left channel data after G.711 decompression. The time stamp field of the RTP header is computed using the system timer and the audio card PCM data rate. Every tick of the system timer is equal to 250 µsec, and the audio card generates 8 bytes of data per ms as seen above, which is equivalent to two bytes of data per timer tick. Therefore, multiplying system timer units by 2 converts them into PCM byte units.

3.2 Jitter Buffer

Voice packets are inserted in a jitter buffer and queued for playback by the audio task. The jitter buffer consists of an array of voice payload descriptors. Each payload descriptor has fields corresponding to RTP header fields. In addition, each descriptor has a field for the size of the received voice packet, a field that contains a pointer to the actual voice packet stored in the DPD memory, and the playout time field for each packet. The values stored in each field except for the playout time field are obtained from the RTP handler. The number of entries in the jitter buffer payload descriptor array is set during initialization of the code that is responsible for managing the jitter buffer. A description of the voice packet is inserted into the jitter buffer at any open slot when a packet arrives, and packets are removed for playback using the playout delay and the sequence number.

We have implemented both fixed delay and adaptive jitter buffer algorithms in the bare PC softphone. Jitter calculations are done as in [11]. The bare PC softphone continually monitors performance and reports the values of network delay, end-to-end (total) delay, jitter, and the maximum inter-arrival gap (max delta) between packets. The end-to-end delay is measured from the time voice data for a packet is extracted from the sender's microphone buffer till it is copied 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 [12]. Note that synchronized sender and receiver clocks are not needed to compute jitter and max delta.

4 Performance Measurements

In order to evaluate the performance impact of bare PC optimizations, we conducted several experiments over the Internet and within a LAN in our laboratory. We also implemented a voice over Ethernet service that minimizes voice bandwidth on a LAN and studied its call quality.

4.1 Internet Measurements

To make calls over the Internet, we tested the bare PC softphone application in typical home and campus/business environments. 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. For all our experiments, the bare PC ran on a 2.4 GHz Dell Optiplex GX260 with 512MB of memory. In order to compare bare PC softphone performance with that of a conventional softphone, we used WinRTP [13] on an identical machine running Windows XP with unnecessary services disabled. 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 this was observed to be worse than WinRTP-to-bare PC performance in earlier studies conducted over a test LAN in our laboratory [8].

Figure 3 shows the maximum packet inter-arrival gap (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 Internet conditions during the experiments.

Figure 4 shows the variation in the maximum packet inter-arrival gap (max delta) and 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 this period.

In Figure 5, 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 between 100 ms to 450 ms.

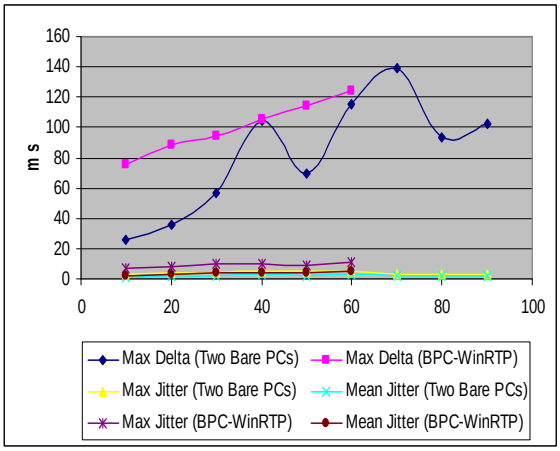

Figure 3. Maximum packet inter-arrival gap (max delta), and maximum and mean jitter with varying voice packet sizes for a bare PC-to-bare PC and a WinRTP-to-bare PC connection.

It is generally accepted that 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, which roughly correspond 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, they did not observe a significant drop in voice quality even with the larger delays and 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.

4.2 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 packets was increased 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 by using the MGEN tool. 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.

4.3 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, or a 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).

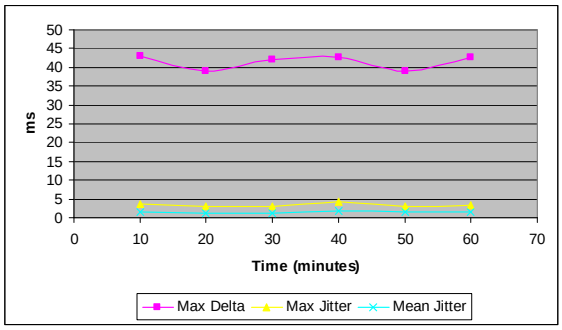

Figure 4: Maximum packet inter-arrival gap (max delta), maximum jitter, and minimum jitter versus time for a bare PC-to-bare PC connection with 20 ms packets and a 100 ms fixed delay jitter buffer.

Of course, this voice over Ethernet service has several drawbacks. For example, packet loss cannot be detected and no ordering of packets is possible, so packets are played in the order of arrival. Moreover, packets cannot be forwarded across IP subnets by routers (or across the Internet) due to 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 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.

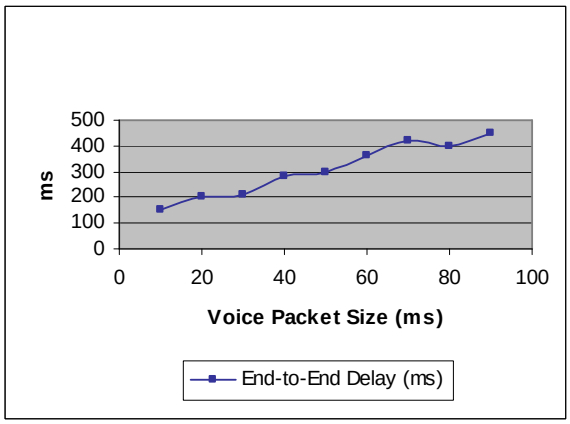

Figure 5. End-to-end delay for a bare PC-to-bare PC connection as packet size is varied with a 100 ms fixed delay jitter buffer.

5 Related Work

To the best of our knowledge, there are no VoIP systems that run on a bare PC. The tool described in [10], uses several optimizations similar to ours to improve call quality. However, unlike our softphone, it is built on top of an OS, and does not include an optimal task scheduling technique. The peer-to-peer VoIP architecture in [2] has many desirable features. The associated SIP (Session Initiation Protocol) adaptor works with existing SIP phones, is capable of supporting seamless addition of new services such conferencing and voice mail, and is essentially plug-and-play. Yet, it may be harder to optimize such a system for performance since the user agents that are required in order to use the adaptor rely on an OS. Another SIP-based VoIP architecture that also offers mobility support is discussed in [15]. Again, this architecture is implemented on a conventional OS and the overall performance of the system is 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.

Many attempts to improve call quality in VoIP systems have also been made. In [16], 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 [17]. In [18], Skype and MSN VoIP systems are compared with respect to throughput, packet inter-arrival statistics, and MOS (mean opinion score). Finally, in [19], 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. Addition of such optimizations to a bare PC may further improve performance since it has less intrinsic overhead 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.

6 Conclusion

In this paper, we described the architecture design and implementation of a bare PC (OS-less) VoIP softphone. We discussed several architectural and design features unique to a bare PC softphone including zero copy buffering and optimized task scheduling. On the Internet, a bare PC-to-bare PC connection is associated with smaller values of jitter than a WinRTP to bare PC connection even for larger voice packet sizes. A bare PC softphone also provides better call quality than a WinRTP softphone under heavy system load conditions on a LAN. Finally, 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.

7 References

[1] S. A. Baset and H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *Proc. of the INFOCOM '06*, Barcelona, Spain, April 2006.

[2] K. Singh and H. Schulzrinne. Peer-to-Peer Internet Telephony using SIP. In *Proc. NOSDAV'05*, Stevenson, WA, June 2005.

[3] R. K. Karne, K. Venkatasamy and T. Ahmed. Dispersed Operating System Computing (DOSC). In *Proc. OOPSLA 2005 (Onward Track)*, San Diego, CA, October 2005.

[4] R. K. Karne. Application-oriented Object Architecture: A Revolutionary Approach. In *Proc 6th International Conference, HPC Asia 2002*, December 2002.

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

[6] D. R. Engler. The Exokernel Operating System Architecuture. *Ph.D. thesis, MIT*, October 1998.

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

[8] 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 2007*, Jan 2007.

[9] R. K. Karne, K. Venkatasamyand T. Ahmed, T. How to run C++ applications on a Bare PC. In *Proc. 6th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel / Distributed Computing*, May 2005.

[10] O. Hagsand, I. Marsh and K. Hanson. Sicsophone: A low-delay Internet telephony tool. In *Proc. of the 29th EUROMICRO Conf. (EUROMICRO'04)*, 2003.

[11] 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

[12] Ethereal Network Analyzer. http://www.ethereal.com

[13] WinRTP. http://www.vovida.org

[14] Multi-Generator (MGEN). http://cs.itd.nrl.navy.mil/work/mgen/index.php

[15] S. Zeadally and F. Siddiqui. Design and Implementation of a SIP-based VoIP Architecture. In *Proc. 18th Intnl. Conf. on Advanced Information Networking and Application (AINA'04)*, 2004.

[16] M. Ghanassi and P. Kabal. Optimizing Voice-over-IP Speech Quality using Path Diversity. In *2006 International Workshop on Multimedia Signal Processing*, October 2006.

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

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

[19] W. Jiang and H. Schuzrinne. Comparison and Optimization of Packet Loss Repair Methods on VoIP Perceived Quality under Bursty Loss. In *Proc. NOSSDAV'02*, 2002.