A Peer-to-Peer Bare PC VoIP Application

Gholam H. Khaksari, Alexander L.Wijesinha, Ramesh K. Karne, Long He, Sandeep Girumala
Department of Computer and Information Sciences
Towson University
Towson, MD 21252
{gkhaks1, awijesinha, rkarne, longhe, sgirumala}@towson.edu

***Abstract*—Bare PC computing is a novel approach to computing in which there is no operating system, and applications are provided with direct interfaces to the hardware. Bare PC systems are of particular interest for secure, reliable, and efficient peer-to-peer communication between users in view of their inherent simplicity. In this paper, we first provide a brief overview of bare PC computing and note their advantages for peer-to-peer communication. We then describe a peer-to-peer bare PC VoIP application, and present call quality measurements including packet loss, delay, jitter, and MOS (Mean Opinion Score) from several experiments conducted in our laboratory. The results indicate that call quality achieved with bare PC VoIP systems is better than that of operating system based softphones. They also show that a bare PC is able to sustain larger voice packet sizes with no observable degradation in call quality.**

***Index Terms*— VoIP, Bare PC, Softphone, Peer-to-Peer, Call Quality.**

I. INTRODUCTION

An operating system manages resources and interfaces between the users and the hardware. However, critics claim that today's operating systems are often plagued by buggy code and device drivers making them insecure and unreliable [15]. New generations of operating systems promise to provide enhanced security.

It has been suggested in the past that operating system inefficiencies and abstractions should be eliminated [3]. An alternative approach to building systems is to 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.

We have developed several applications that run on a bare PC. Bare PC systems are particularly suited for peer-to-peer communication. We are presently studying performance, security, reliability, and portability issues involving peer-to-peer bare PC systems. In this paper, we describe the design of a simple peer-to-peer bare PC VoIP application and present preliminary results comparing its performance with that of a VoIP client running on an operating system. We begin with a brief overview of bare PC computing in Section II. In section III, we describe the VoIP application. Performance results are presented in Section IV. Section V contains the conclusion and future directions.

II. BARE PC COMPUTING

In bare PC computing, also known as dispersed operating system computing [7], a computer application contains its own operating environment, thus avoiding operating system middleware. Figure 1 illustrates this approach. Conventional computing layers are mapped into an application object and hardware, 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.

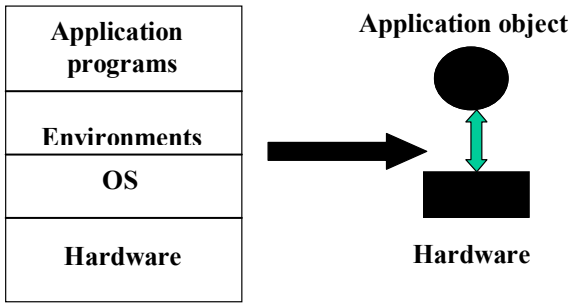

Fig 1. Bare PC Computing

An application object contains both its application program and its necessary application operating environment [8] [9]. It is thus self-contained, self-managed, and self-executed enabling 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 [10] may be part of an application operating environment or implemented in hardware. In this paper, we assume that these interfaces are in an application operating environment (i.e., existing hardware is used).

We note that a bare PC is different from an embedded system [2], and systems that provide a virtual machine interface enabling software to execute with or without an operating system [11]. 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 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 and it is given direct and full access to the underlying hardware.

We have developed a C++ API allowing applications to run directly on Intel 386 (or above) -based PCs. Interfaces to memory, CPU, timer, interrupts, tasks, keyboard, display, floppy drive, audio card and Ethernet card are written in Microsoft assembler (MASM). We have also implemented several network protocols for a bare PC including ARP, IPv4, UDP, TCP, SMTP, RTP, trivial FTP, and HTTP. We described a bare PC Web server that runs on any Intel 386-based architecture with no operating system, hard disk, or other supporting software, and presented results showing its performance is comparable to the Windows IIS and Apache Web servers in [12]. We are presently focusing on peer-to-peer bare PC systems that can support efficient, reliable and secure communication. Many VoIP systems including systems for peer-to-peer voice have been discussed in the literature [1], [4]-[6], [13], [14], [17], although none run on a bare PC. Security issues for VoIP systems are highlighted in [16].

III. A BARE PC VoIP APPLICATION

We now describe the bare PC VoIP application. Figure 2 shows the elements of the audio client and the tasking structure. Voice is input through the microphone, digitized using 16-bit PCM and placed by the ICH5 controller [18] in the microphone buffer. The record task removes t ms of voice data from the buffer, which results in $8t$ bytes of data after compression using a G.711 encoder (stereo-to-mono conversion is achieved by ignoring the data from one of the channels, and the encoder converts 16-bit PCM data to 8 bits). Next, RTP, UDP, and IP headers (a total of 40 bytes) are added to form a voice packet that is placed in the send buffer for transmission on the network by the Ethernet card driver. Conversely, when a voice packet is received, it is placed in the receive buffer by the Ethernet card driver, and the play task calls the IP handler, which in turn calls the UDP and RTP handlers. The latter then places the G.711-encoded voice data in the jitter buffer and the play task writes it to the speaker buffer for play back by the ICH5 controller after it is decoded (data from the single channel is duplicated to reconvert mono to stereo). Note that the record and play tasks directly communicate with the Ethernet card, thus avoiding inter-process communication.

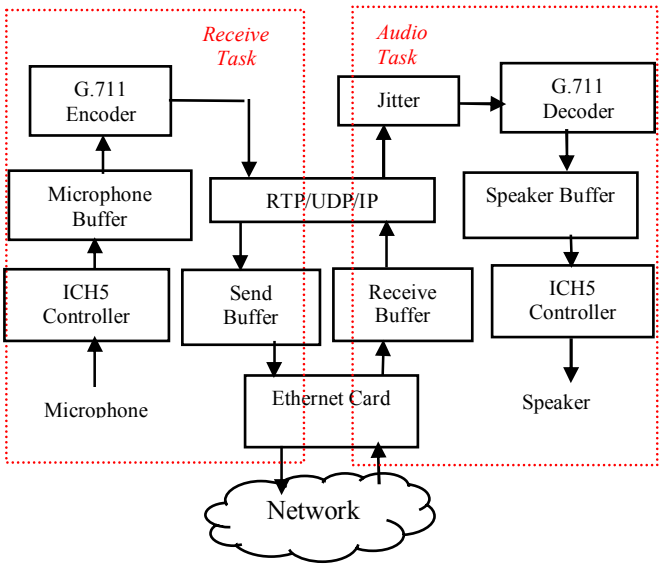

Fig. 2. Bare PC VoIP Components

The record and play tasks are activated by a main task that monitors microphone recording, speaker playback, and the arrival of voice packets. Each task suspends itself and returns control to the main task upon completion. The main task uses a vector to store flags that are used by the scheduler to select tasks for execution. Note that the task management for VoIP in a bare PC is quite simple and very flexible allowing the designer to add enhancements or modify task processing with little effort. Although we have not implemented silence suppression or packet loss concealment yet, such features are easily incorporated and would result in additional performance improvement. Also, since our interest is primarily in peer-to-peer communication, we have not used SIP for connection management.

IV. EXPERIMENTAL RESULTS

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 network to measure call quality. For ease of data collection and measurement, a hub was used (instead of a switch) to connect the bare PCs running the VoIP clients and a sniffer was connected to the hub for passive monitoring of calls. Each bare PC is a 2.4 GHz Dell Optiplex GX260 with 512 MB memory. A commercial tool was also connected to the hub and used to calculate the MOS. The

propagation delay between a pair of VoIP clients is negligible. For reasons of space, and since our interest is in intrinsic bare PC performance, we focus on jitter-related measurements, which will serve to illustrate the potential of bare PC VoIP systems. 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.

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, as expected, 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 each bare PC 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 up to 60 ms.

The maximum interarrival time between successive voice packets in each direction for different frame sizes is shown in Figure 3. For voice data sent by either a bare PC or a Windows client, the maximum gap between packets is close to the voice frame size in ms, so a linear relationship is observed, regardless of whether the receiver is a bare PC or a Windows client. However, if we plot the maximum deviation from the voice packet size considering traffic from both directions as in Figure 4, the deviation for a bare PC is seen to be smaller than that for the Windows client.

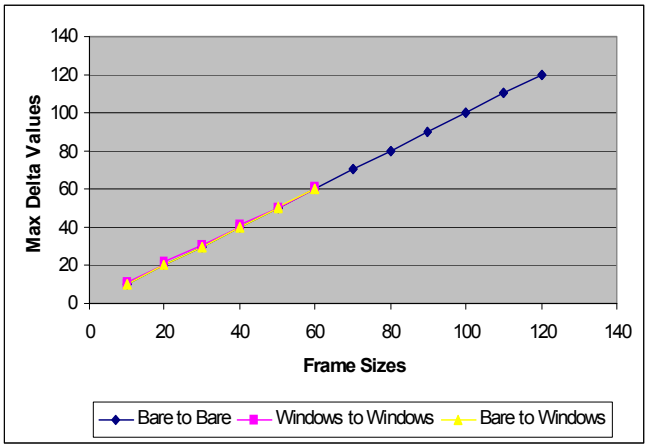

Fig 3. Maximum Interpacket Gap (ms)

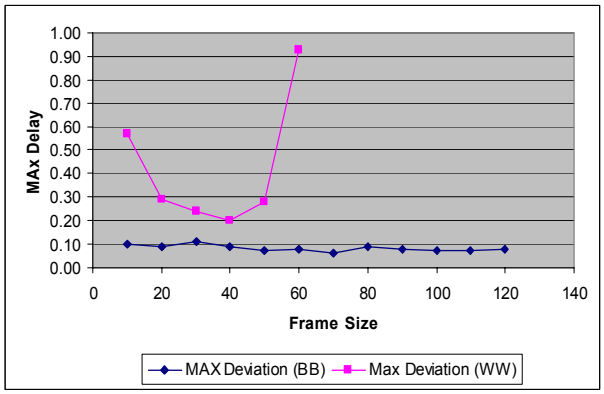

Fig 4. Max Two-Way InterPacket Deviation (ms)

Next, we consider the maximum jitter separately in each direction for a two-way voice stream, which is shown in Figure 5a and Figure 5b respectively. The Windows clients exhibit a marked directional asymmetry when sending voice data to each other, which is likely due to 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 has less asymmetry since 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. We will investigate this observation further in future studies. The corresponding mean one-way jitter (in each direction) for the above experiments shown in Figure 6a and Figure 6b 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 and results depend on the underlying operating system (Windows) on which the sniffer is running. We have not attempted to measure these limits in our studies.

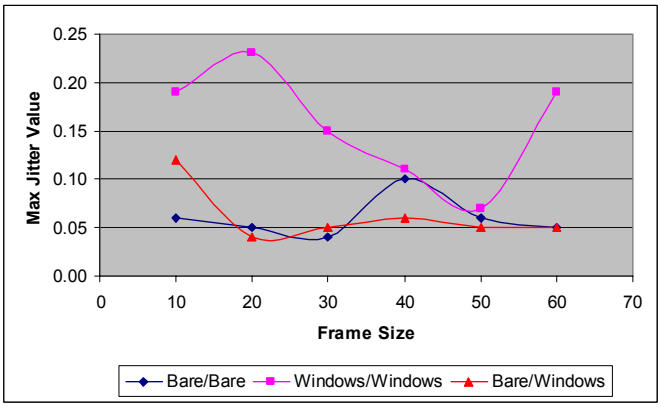

Fig 5a. Maximum One-Way Jitter (ms)

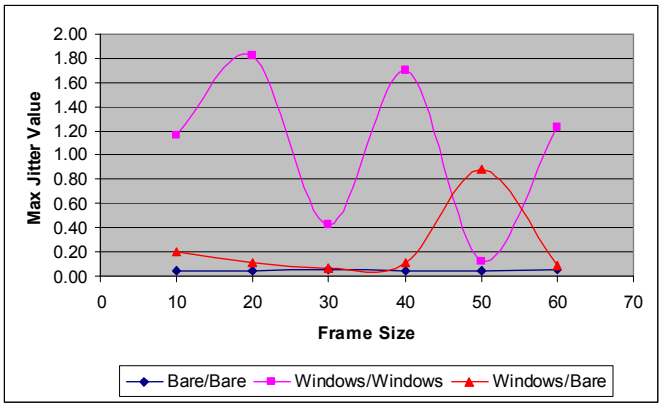

Fig 5b. Maximum One-Way Jitter (ms)

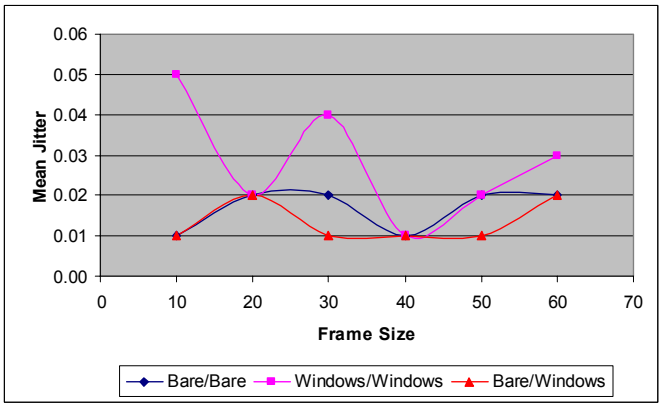

Fig 6a. Mean One-Way Jitter (ms)

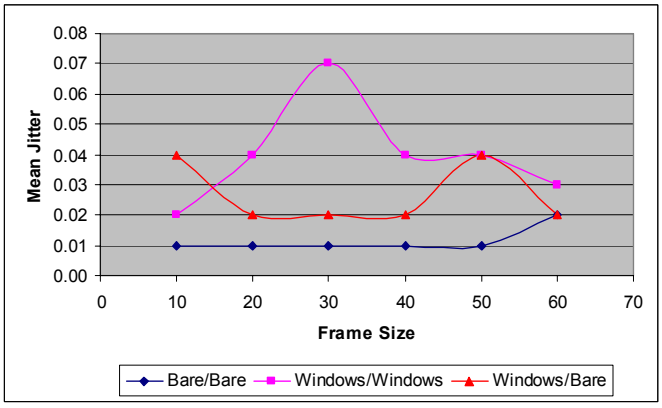

Fig 6b. Mean One-Way Jitter (ms)

In Figure 7, we show the maximum jitter 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 also 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 call quality overall with occasional packet loss and slightly larger values of delay and jitter, but no significant change to the MOS. However, the loss of a larger packet now has an observable effect. We will investigate the effects of heavy background traffic in future studies.

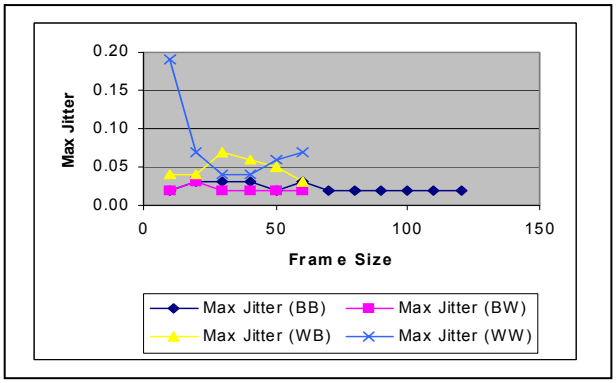

Fig 7. Maximum Two-Way Jitter (ms)

V. CONCLUSION AND FUTURE WORK

We have discussed bare PC computing and its advantages for peer-to-peer applications. We described a peer-to-peer bare PC VoIP application and presented call quality measurements for bare PC systems including the MOS. The results indicate that the performance of a bare PC VoIP system is better than that of an operating system-based VoIP system. In particular, no loss in call quality was observed for increased voice packet sizes.

Bare PC systems are also advantageous due to their convenience, simplicity and efficiency. In particular, they could be used for peer-to-peer computing with no dependence on operating systems. Security and reliability issues in bare PC systems are an important area for future research. We are currently investigating security mechanisms for bare PC VoIP peer-to-peer systems and the use of a bare PC for communication across NAT-based firewalls.

REFERENCES

[1] S. A. Baset and H. Schulzrinne. An analysis of the Skype peer-to-peer internet telephony protocol. In *Proceedings IEEE INFOCOM* 2006.

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

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

[4] O. Hagsand, I. Marsh, and K. Hanson. Sicsophone: A low-delay Internet telephony tool. In *Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)*.

[5] M. Hillenbrand, J. Gotze, and P. Muller. Voice over IP – Considerations for a next generation architecture. In *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'05)*.

[6] 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 *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*.

[7] R. K. Karne, K. V. Jaganathan, and T. Ahmed. DOSC: Dispersed Operating System Computing. OOPSLA '05, 20th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, Onward Track, October 2005, San Diego, CA, pp. 55-61.

[8] R. K. Karne, R. Gattu, R. Dandu, and Z. Zhang. Application-oriented Object Architecture: Concepts and Approach. In *Proceedings of 26th Annual International Computer Software and Applications Conference, IASTED International Conference*, Tsukuba, Japan, NPDPA 2002, October 2002.

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

[10] R. K. Karne, K. V. Jaganathan, and T. Ahmed. How to run C++ Applications on a bare PC. In *Proceedings of SNPD 2005, 6th ACIS International Conference*, May 2005, pp. 50-55.

[11] S. T. King, G. W. Dunlap, and P. M. Chen. Debugging operating systems with time-traveling virtual machines. In *Proceedings 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 *Proceedings of SNPD 2006, 7th ACIS International Conference*, June 2006.

[13] 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.

[14] K. Singh and H. Schulzrinne. Peer-to-peer Internet telephony using SIP. In *NOSDAV'05*.

[15] A. S. Tanenbaum, J. N. Herder, and H. Bos. Can we make operating systems reliable and secure? *IEEE Computer*, May 2006.

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

[17] S. Zeadally and F. Siddiqui. Design and Implementation of a SIP-based VoIP Architecture. In *Proceedings of the International Conference on Advanced Information Networking and Application (AINA'04)*.

[18] Intel 82801 EB (ICH5) I/O Controller Hub: AC '97 Programmers Reference Manual, April 2003, Document Number: 252751-001,