The Performance of a Bare Machine Email Server

George H. Ford, Jr., Ramesh K. Karne, Alexander L. Wijesinha, and Patrick Appiah-Kubi
Department of Computer and Information Sciences
Towson University
Towson, Maryland, USA
gford3@students.towson.edu, rkarne@towson.edu, awijesinha@towson.edu, pappia1@students.towson.edu

***Abstract*— Bare machine applications run directly over the hardware without using an operating system or the hard disk. This paper studies the performance of a bare machine email server whose design and implementation is based on several novel architectural features, and targeted at optimizing performance. The results are compared with those for the AxiGen and ShareMailPro email servers, and a lean Java-based email server prototype running on Windows whose application-level operation closely matches that of the bare machine email server. For 80,000 emails in a LAN environment, the bare Machine server processing time is approximately 2 times faster than a Java-based server, and 2.4 times faster than the AxiGen server. For 5,500 emails in a WAN environment, the bare machine server performed at least 1.8 times faster than the Java-based and ShareMailPro servers. The results indicate that the bare machine email server outperforms the conventional email servers in LAN and WAN environments and demonstrate the ability to build high-performance email servers that run on bare machines.**

Keywords- bare Machine Computing (bare), Application Object, Email Server, SMTP/ POP3, Performance.

I. INTRODUCTION

Email servers involve several protocols, whose operation, functionality and evolution are specified in numerous well-known RFCs. The performance of an email server depends on the ability to optimize and exploit architectural design features that facilitate efficient interaction with email clients and other email servers. The principal application-layer protocols involved in these applications (SMTP, POP3 and IMAP) and the underlying TCP/IP network protocol suite also influence the performance of the server. Optimizing, tweaking, and intertwining the necessary protocols for supporting a specific application and simultaneously designing the architectural features of the server enables performance of a server to be dramatically improved, thus providing a blueprint for creating high-performance email servers.

Most of the server designs are based on conventional operating systems, lean kernels, or virtual machines where a centralized operating system controls and manages application resources. The approach presented here is based on eliminating any OS or kernel operating as middleware between applications and hardware. In essence, the application directly communicates to the hardware, thus eliminating the middleware and any associated layers. This provides optimal performance and independence to any operating system environments. The detailed architectural features, design and implementation were presented in [5]. This paper strictly focuses on performance measurements of the bare email server, since inclusion of the design details in this paper would have been too large to present. The bare machine email server provides an alternative to conventional email servers. Bare machine servers are self-contained, self-executing, and application-specific to achieve higher performance, requiring no OS related maintenance or dependences. Their design, based on the bare machine computing paradigm [8, 9] has been previously shown to be successful in optimizing performance on an application-specific basis [6, 7, 11], resulting in an order-of-magnitude performance improvement over conventional systems.

In the current bare approach, an Application Object (AO) [10] executes directly on the most popular Intel IA-32 based PCs or devices. One could also design an AO to execute on RISC based machines. An AO, which consists of the application program, and all the necessary code to boot, load, and execute it, including direct interfaces to the hardware, is a single, monolithic executable code, which does not use any OS related system calls, kernels, mini/nano kernels, or any system libraries. The AO is bootable using a mass-storage medium, such as USB flash memory.

The remainder of this paper presents related work in Section II, a brief overview of the important architectural features and their performance implications in Section III, experimental details and performance results in Section IV, and the conclusion in Section V.

II. RELATED WORK

Eliminating unnecessary operating system abstractions was first proposed in [1] by using an Exokernel. Later, approaches based on micro-kernel [3], TinyOS [15] and lean OS libraries [17] attempted to provide a minimal OS interface to run applications. Attempts to provide an intricate bypass of OS-based system calls was performed by systems such as IO-Lite and Flash [12]. Bare metal Linux [16] applications were used in manufacturing to bring up (boot) hardware. Sandboxing techniques have also been used on x86 systems [4] to extend OS kernel abilities to execute application plug-ins on OS hosts. However, sandboxing and customized kernel approaches still require some OS elements existence to support their applications.

Prior research related to email server architectures have focused on OS-based threading features. Threaded

communications and processes can provide many advantages in terms of security and concurrent operations for users. From a security design approach, existing email servers are typically deficient in providing features for email spam filtering, Denial of Service (DoS) sensing and security awareness [14]. Performance measurements have generally focused on an email server's ability to handle large quantities of emails in a specific time, threading capabilities, and throughput capacities [2, 13].

All of the above legacy applications use some form of kernel or OS libraries to run applications. The bare machine computing approach eliminates the need for any kernel or OS libraries and bundles the necessary operating environment within the application, directly communicating with hardware to manage its resources. The bare email server presented here is based on this approach.

III. BARE MACHINE ARCHITECTURAL PERFORMANCE CHARACTERISTICS

This section identifies the key performance characteristics and novel architectural features of bare machine computing, which can be leveraged when building high-performance email servers. These characteristics are difficult to replicate in a conventional OS-based application. The bare approach is currently focused on a set of selected networking application-specific systems, although the concept can be extended to enterprise applications. Other examples of successful bare systems include VoIP [11], and Web Server [6] applications.

The bare machine computing approach does have some drawbacks in its design approach and portability with existing systems. First, the design of an AO requires indepth knowledge of systems programming and application programming, thus making the AO programmer's job harder than conventional application programming. Howerver, once an AO is designed, it does not depend upon any upcoming OS releases, thus avoiding subsequent porting of an AO to a new platform. This current design also assumes a single, stable underlying architecture (e.g. IA32), thus making the current system specific to this architecture. Secondly, porting current applications to bare machine computing is a daunting effort due to system calls and libraries. In order to port conventional applications to a bare machine computing, it requires that all system calls and libraries must be transformed into direct hardware interfaces that can be invoked in the application, thus making it difficult to port OS-based applications to bare AOs.

Some of the unique architectural characteristics which illustrate performance improvements for bare machine email servers are summarized in the following subsections.

A. Protocols

The bare machine approach to designing and implementing application-specific servers and systems uses intertwining of the necessary protocols [5]. Such intertwining is based on the programmer's ability to include these protocols in an optimal fashion within a single instance of code, avoiding the restrictions due to conventional layered stacks used in OS-based systems, enabling the design of high-performance network applications.

B. Concurrent Requests

Multiple task pools (POP3, SMTP, and Relay Forwarding) are created at initialization to handle concurrent requests. Each of the above task types is handled as a generic task, which saves time by avoiding the dynamic creation of tasks at run-time and improving response time.

C. Concurrency Control

Each email request and its state information are stored in a Transition Control Block (TCB) table, in addition to the parameters used by the reentrant code. In an email server, each request's state information is independent of other requests. The simultaneous sharing of resources is avoided by creating independent resources for each request and maintaining their state information in the TCB. This TCB information is used in task scheduling and may be used in migrating email servers.

D. Task Scheduling

Unlike the OS-driven task scheduling, the bare approach implements a user-driven technique. At development time, an AO programmer designs a task schedule for a given task as a single thread of execution, thus eliminating the need for a centralized task scheduler. When an application needs to wait for an event then the program is suspended. When the relevant event arrives, the task resumes execution. This approach has been demonstrated in building high performance bare Web Servers [6].

E. Client Interactions

OS-based email server designs are focused on application layer request-response interactions with clients. In the bare approach, the request-response interactions are modeled based upon State Transition Diagram behaviors. This alleviates the overhead associated with OS-based servers due to their use of API libraries and layered approaches. The result is faster throughput performance, and an ability to process more emails in a given time.

F. System Operation

Contemporary applications require OS-based kernel utilities and programs to start the system, load the application, and manage resources. The bare approach provides the ability to run a self-contained, monolithic executable code capable of bootstrap loading and providing resource management directly within the application. This allows the bare application to avoid any need for an OS when a user boots and executes the application from USB flash memory or similar storage media.

G. Volatile Memory and Persistant Storage

Legacy email servers rely upon OS-based features for memory usage which require caching and paging, and virtual memory management with disk I/O. Since there is no OS in the bare approach, it frees up the use of physical memory for specific applications. Persistent storage in the

bare is provided by USB flash memory, which can serve as large mass storage. Thus, USB flash memory can provide a portable storage medium for both the bare application and its data.

H. Input / Output

When OS-based applications use I/O, they require use of system calls or an API. In the bare application design, interrupt usage is limited to keyboard, hardware timer, and the NIC device. Limited BIOS interrupts are used during the boot process and in acquiring device addresses. Some popular device drivers have been developed for the bare applications, where they can invoke in-line calls to the device driver which results in an increased I/O performance.

I. Security

Bare applications are simple, small in code size, and immune to attacks that exploit known OS vulnerabilities. A high-performance bare machine email server would be easier to secure than a complex OS-based server especially due to its streamlined architectural design.

IV. Performance Measurements and Analysis

The performance measurements and analysis, presented below, demonstrates and validates the bare machine computing architectural design and implementation approach as described in [5]. These results also indicate that similar performance enhancements can be achieved in other types of servers using this bare email server approach.

A. Experimental Setup and Components

This paper focuses on the performance of a bare email server running on an ordinary desktop PC. Experiments were conducted using Dell OptiPlex PCs with 2.4 GHz Pentium 4 processors and 3COM 905CX NICs. For comparison with the bare email server, email servers running on Windows XP including a Java-based email server prototype with equivalent functionality and optimized with the just-in-time compiler, as well as ShareMailPro and AxiGen were used. All unnecessary services executing on the MS-Windows-based systems were disabled. Additionally, file I/O was disabled on the Java-based server prototype and it was limited to displaying only a simple text status string for monitoring purposes (it had no GUI). It was observed during tests that the conventional email servers also cached messages into memory.

The bare email server, like its Java-based counterpart, supports SMTP, POP3, authentication, MIME encoding, and relay forwarding. In addition to conducting extensive tests to verify correct operation of the bare server, its ability to interoperate with many popular email clients and servers was verified. Clients used in performance testing included SMTP Stress Test Checker v1.0, Mozilla Thunderbird, Pegasus, MS-Outlook Express, Alpine, eM-Client, Dream Mail, Eudora 7, IncrediMail, and Mulberry; a Java-based client, and a bare email client were used for testing specific aspects of bare email server operation. The SMTP Stress Test Checker v1.0 generates and sends bulk emails to the server for an email size (up to 1KB). This tool also allows up to 100 concurrent threaded connections, each with a tester specified quantity of emails. This test tool maintains statistics on the email transactions. In addition, email transactions for all threaded connections are captured with the Ethereal / Wireshark network tool, which allows handshaking, packet and timing analysis to be determined.

The application object for the email server consists of C/C++ code in addition to some hardware interfaces coded in assembly language. The bare server contains 31,743 lines of code including 10,000 lines of comments, and 14,000 executable statements. The resulting single monolithic executable (AO) consists of 512 sectors of code (256 KB).

B. Performance Results and Analysis

A variety of performance measurements were conducted on bare, Java, ShareMailPro, and Axigen email servers to investigate the following:

- Determining LAN based throughput processing times
- Determining WAN based throughput processing times
- Measurement of internal message protocol timing
- Effects on different topologies used for implementation of the scheduling queue mechanism.

1) LAN Measurements

For LAN measurements, the Stress Checker Tool and the Wireshark network capturing tool were running on MS-Windows with no other applications running on the machine (A). The email servers were running on another dedicated machine (B). Machines A and B were connected through an isolated LAN in the research lab.

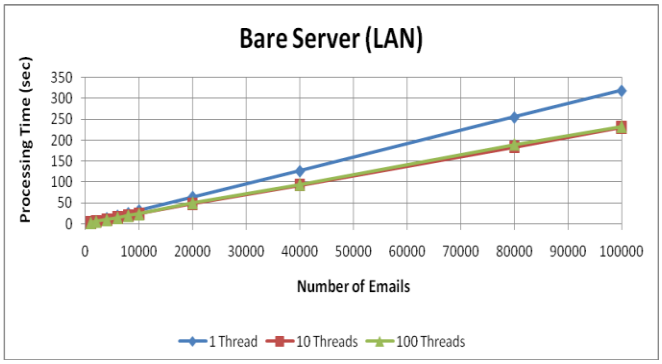

Figure 1. bare Server Processing Time

Figure 1 illustrates the bare email server processing times with respect to a large number of emails. This graph also shows the effects due to different numbers of threads used in processing the emails (1, 10 and 100 threads). These measurements were obtained using the Stress Test Checker v1.0 tool.

The stress tool can be used to send up to 100,000 emails, with multiple threaded connections. Multiple threads reduce the bare email server processing time compared to a single thread, but varying the number of threads from 10 to 100 yields a negligible change in performance. For example,

the processing time for 10 threads versus 100 threads differs by approximately 3 seconds, whereas the difference between 1 and 10 threads is 89 seconds (for 10,000 emails).

The bare task processing is designed as a single thread of execution without interruption. In addition, it also intertwines the protocols and avoids the overhead of running the server application on an OS. The multi-tasking effect in the bare applications has diminishing returns as the number of tasks grows in the system. Multi-tasking has less variance in bare applications than in OS based applications for the following reasons:

- When a task is running, there are minimal interrupts which interrupt the task (since the task was designed to run without interruptions)
- There are no system calls or another API to compete for the CPU cycle
- The CPU cycle was already optimized for running useful work related to tasks (no I/O)
- If I/O is needed, the program is designed to suspend (so other tasks can run)
- The CPU time is essentially pre-optimized to run bare machine applications.

Figures 2-4 show measurements with varying threads (1, 10,100) for the various email servers (bare, Java-based prototype, ShareMailPro, and AxiGen). Figure 2 only shows processing time data for three servers (bare, Java-based prototype, and AxiGen) for 80,000 processed emails when the message size is less than 1KB i.e., 1 Ethernet packet. As shown in Figure 2 for 1 thread, the bare server processing time is 1.99 times faster than the Java-based prototype and 2.36 times better than the AxiGen server.

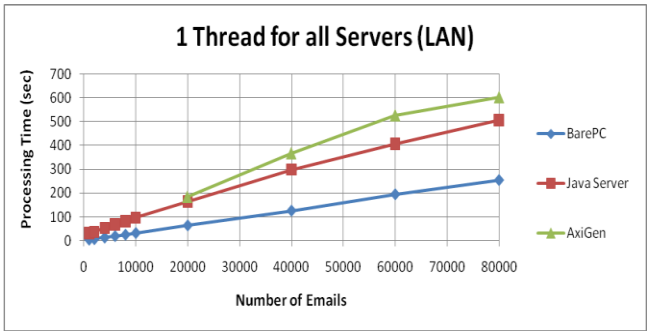

Figure 2. Server Performance Measure with 1 Thread

The processing time of the ShareMailPro server was excessive when testing with the stress tool. Hence, Figure 3 and 4 only show results for up to 10,000 emails to illustrate the performance improvements with respect to the bare server. For 10 threads and 10,000 emails, the bare server processing time is 9.46 times better than the Java-based prototype, 14.7 times faster than the AxiGen server, and 18.29 times faster than the ShareMailPro server; and for 100 threads and 10,000 emails, the bare server processing time is 3.43 times faster than the Java-based prototype, 2.95 times faster than the AxiGen server, and 10.65 times faster than the ShareMailPro server. As the number of threads and the number of emails increased, the performance of the Java-based prototype improved as it leverages multi-tasking capabilities of its OS-based environment. It is expected that the performance of this Java-based server will be closer to that of the bare server if full multi-tasking capabilities are implemented. Similar improvements due to multi-tasking effects were observed in AxiGen and ShareMailPro servers when the number of threads increased. However, the ShareMailPro behavior has larger processing time than the AxiGen server that is likely due to its extensive set of built-in features and complex processing logic.

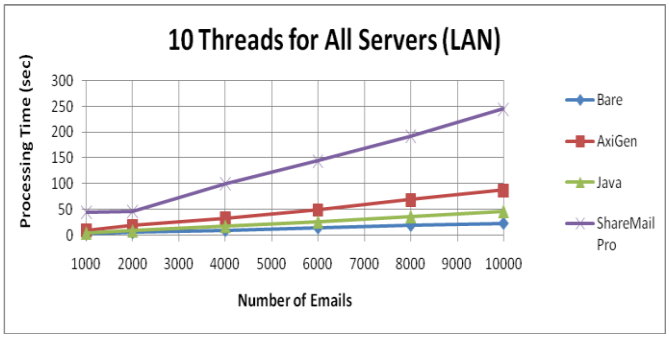

Figure 3. Server Performance Measure with 10 Threads

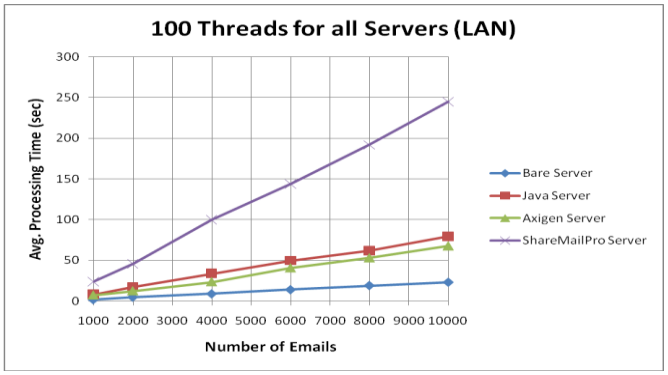

Figure 4. Server Performance Measure with 100 Threads

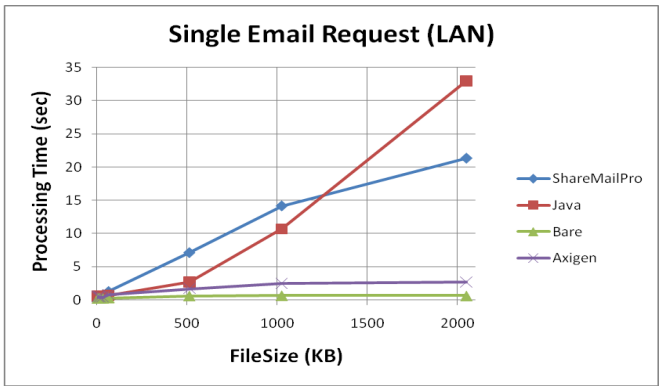

Figure 5. Server Performance for various Email Sizes

Additional LAN measurements were performed varying the email attachment sizes from 1KB to 2MB. As shown in Figure 5, the bare server average processing time for a 2MB email attachment was 3.32 times faster than the AxiGen server, 53 times faster than the Java-based server; and 34 times faster than the ShareMailPro server. Figure 6 shows that with a 2 MB attachment the throughputs are respectively 4.35, 16, and 24 times greater for the bare server compared to the AxiGen, ShareMailPro and the Java-based servers. These results indicate that the bare email server is superior to conventional servers when large email sizes are involved.

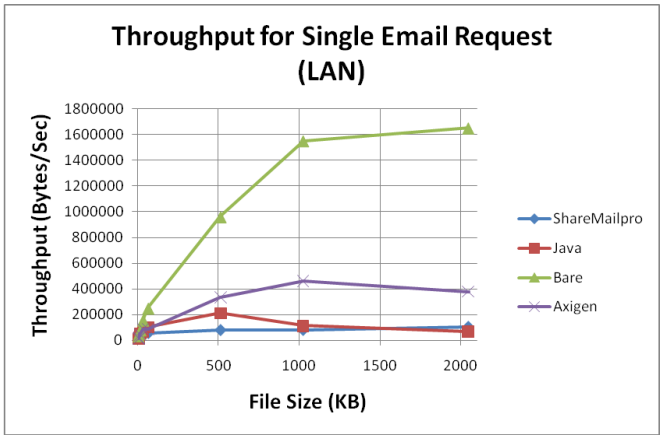

Figure 6. Server Performance for various Email Sizes

2) WAN Measurements

For WAN measurements, the Stress Checker Tool was running on a Windows machine with no other applications running (A), connected to the Internet in a remote location. The email servers were running on another dedicated machine (B) in the research lab. The Wireshark Network Analysis tool was running on yet another Windows Machine (C) located in the research lab. Both machines B and C are connected in the research lab, with B having a static IP address reachable from the Internet.

The performance of the various servers was also tested over the Internet using a remote client located 40 miles away (the Visual Trace Route tool showed that 18 hops were followed between the client and server locations during the tests, which were performed during a contiguous time period). Figure 7 shows the average processing time of 5,500 email requests with less than 1KB message size (1 Ethernet packet). Only 5,500 emails were used in this test as the test site firewalls terminated the connection due to the large number of emails sent during a short period of time for the same connection. The bare server total processing time measured 1260 seconds, compared to 2270 seconds for the Java-based server and 2338 seconds for the ShareMailPro server (overall, the bare server performs about 1.8 times faster than the other servers). Figure 8 shows the throughput for the above tests, which shows improvement by a factor of 2 compared to the other servers.

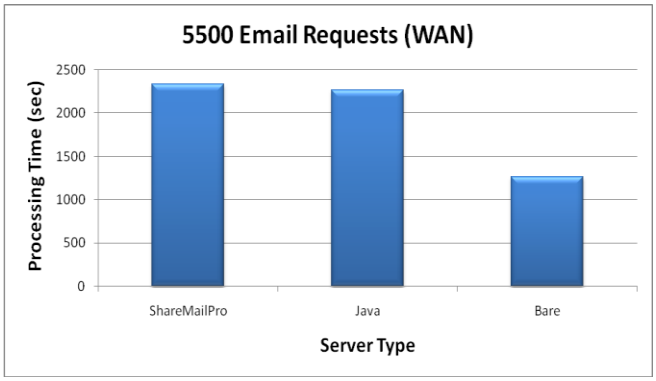

Figure 7. Server Performance with 1KB Data over WAN

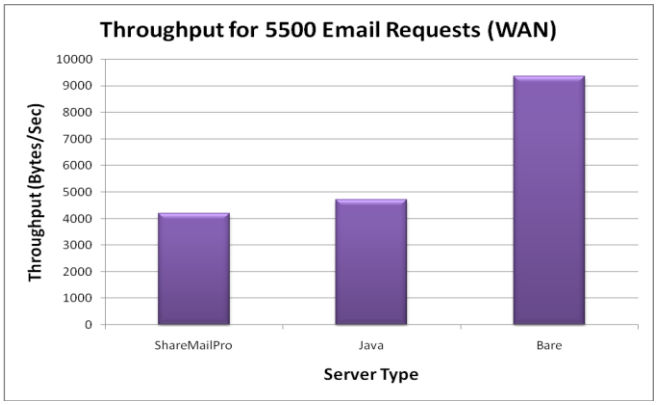

Figure 8. Server Performance with 1KB Data over WAN

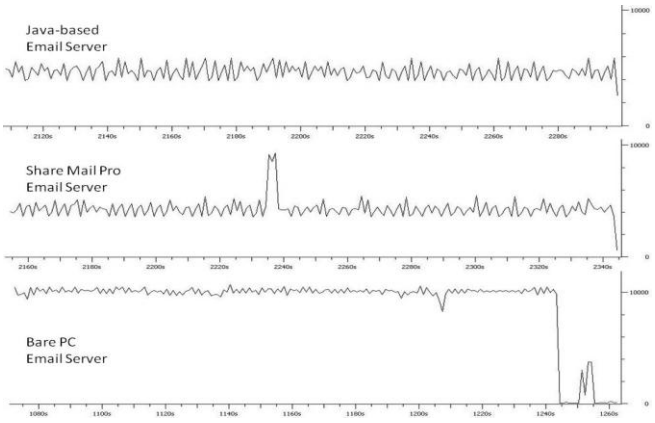

Figure 9. Server Performance Measure of 5500 Messages based on 1KB of Data over WAN

During these Internet tests, Wireshark traces were captured to analyze the throughput of the bare, ShareMailPro, and Java-based servers. Figure 9 shows the three graphs from Wireshark compared in one chart, where the y-axis depicts the maximum scale of 10,000 bytes/sec throughput and the x-axis shows the processing times in

seconds. In the top graph, the Java-based Server shows a running time of 2300 seconds with a throughput range of 4,000 to 6,000 bytes per second. In the middle graph, The Share Mail Pro Server shows a running time of 2340 seconds with a throughput range of 3,500 to 5,000 bytes per second. In the bottom graph, The Bare Email Server shows a running time of 1245 seconds, with a throughput range of 9,800 to 10,000 bytes per second. Notice that on the WAN testing, the Bare Server finished processing 5,500 email messages (1KB each) in 1245 seconds as compared to 2340 seconds for the other servers.

The average processing time for the three servers over the Internet with email attachments of 1 KB to 2 MB is shown in Figure 10. The bare server is 2.4 times faster than the Java-based server and 2.1 times faster than the ShareMailPro server.

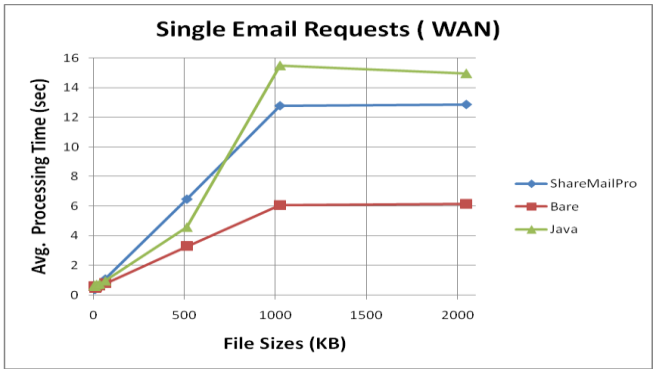

Figure 10. Server Performance for various WAN Email Sizes

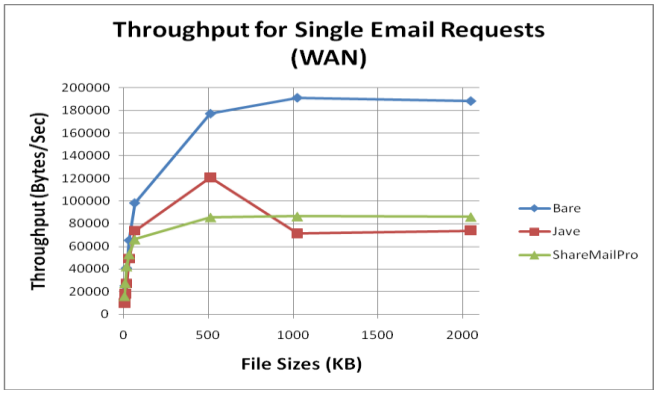

Figure 11. Server Performance for various WAN Email Sizes

Figure 11 shows the corresponding throughput for the three servers with a 2MB email attachment. The throughput for the bare server is 2.54 times that of the Java-based server and is 2.18 times that of ShareMailPro. Again, these throughput improvements demonstrate the capability of the bare server to process larger email sizes in less time compared to conventional servers. However, as expected, the WAN measurements over the Internet take longer times due to network delays and thus exhibit reduced throughput capacities compared to the LAN environment. As these results indicate, bare servers perform better than their OS-based counterparts in both LAN and Internet environments due to their streamlined architectural characteristics and protocol optimizations.

3) Internal Message Protocol Timing

In order to identify any internal performance bottlenecks, request-response timings were determined using a message size of less than 1 KB. Figure 12 illustrates the internal timings associated with LAN-processed messages for four servers (bare, Java-based, AxiGen, and ShareMailPro).

Starting with the client SYN packet through the MAIL FROM command, each server demonstrated the same processing time behavior. From the MAIL FROM command to the start of the Message Body, the AxiGen and ShareMailPro servers exhibit a jump in processing time whereas the Java-based and bare servers remain stable. It is believed that this behavior reflects the additional functionality and features built into the commercial servers (the Java-based and bare servers have the same functional designs). From the transfer of the message body by the client until the ACK is sent by the server, the three non-bare servers had a 140 msec spike in the amount of processing time due to socket overhead involved in OS-based systems. In contrast, such a spike is not seen with the socketless bare server design which intertwines protocols and application thus eliminating the overhead of a conventional OS-controlled protocol stack. All servers exhibit stable processing time behavior from the final message body ACK to closing of the connection.

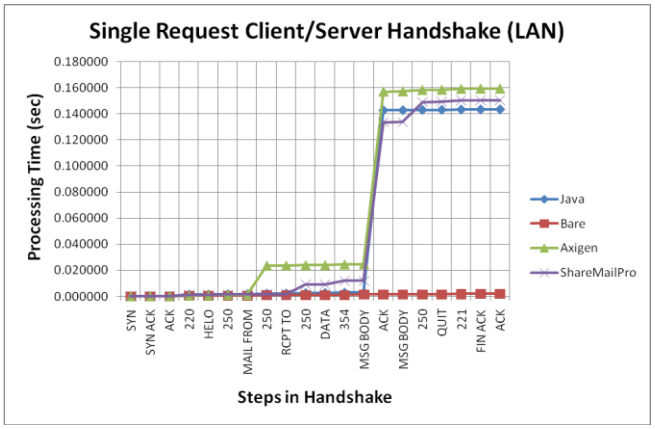

Figure 12. Client/Server Handshake 1KB Data LAN Performance

Figure 13 shows the processing time for a single message on the WAN. Only three servers (bare, Java-based, and ShareMailPro) were used for comparison of internal timings on a WAN (Internet). The spike observed from the message body through the ACK in a LAN measurements are also seen in the WAN tests. The results in Figure 12 and 13 also identify the steps where internal processing times dominate the relatively stable Internet network delays.

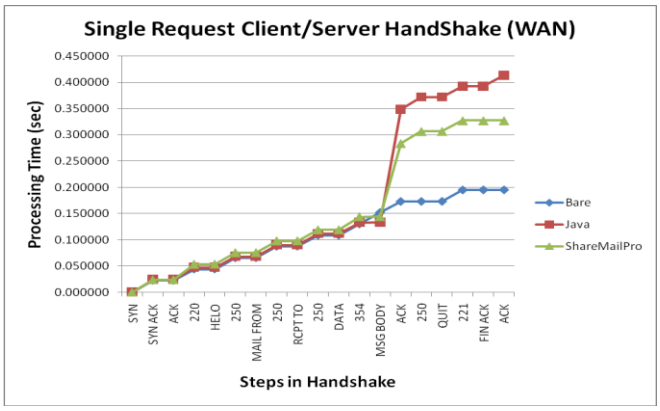

Figure 13. Client/Server Handshake 8KB Data WAN Performance for various Email Servers

Figures 14 and 15 show the results of WAN experiments in which 100 emails were processed with the three servers (bare, Java-based, and ShareMailPro). Figure 14 shows the effect of one thread used for 100 emails for a typical email message handshake, and Figure 15 shows the processing results for four threads with 25 emails for a typical email message handshake. Whereas Figure 13 illustrated the client sending one message to the email servers, Figure 14 represents 100 messages sent from the stress test client. Comparison shows that the WAN processing time behavior pattern is similar to that of the LAN, as expected.

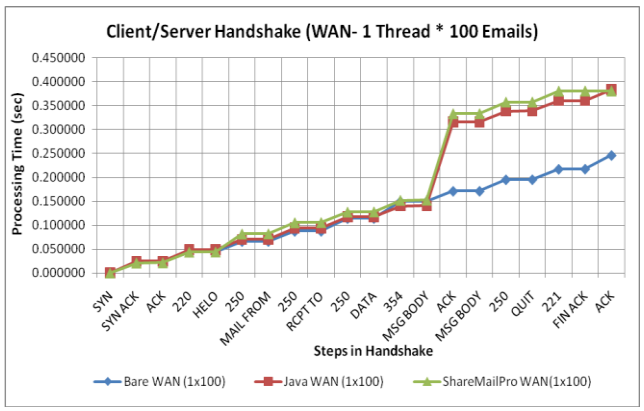

Figure 14. Client/Server Handshake 1KB Data WAN Performance using 1 Thread * 100 Emails

Figure 15 illustrates the processing time, for a typical client-server handshake, for four threads with 25 emails processed by the three servers (bare, Java-based and ShareMailPro). The client server handshake for the ShareMailPro and the Java-based server show more perturbations in processing time than the bare server, which demonstrates relatively stable behavior. In an OS-based environment, additional non-intertwined processing, layering effects, thread scheduling, and context switching may result in unpredictable execution response times. In the bare server, client requests results in a resume event which allows the suspended task to run as a single thread of execution. This results in a more stable behavior for processing times in the bare server.

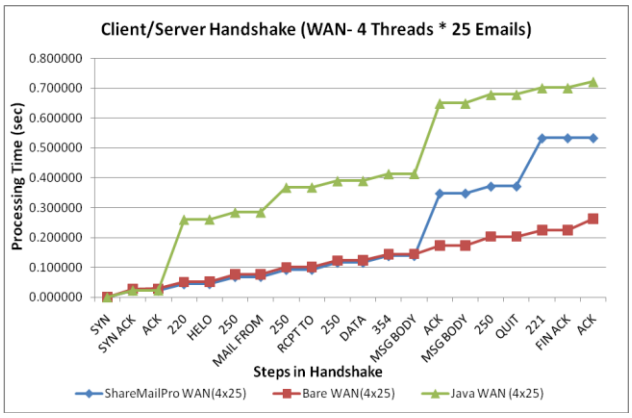

Figure 15. Client/Server Handshake

4) Effects of Multi-level Queues and Scheduling

In the original approach, a single ready queue was used with a First-Come-First-Served (FCFS) scheduling technique. Later, a double queue approach was explored in which a Resume Queue (RQ) handles active email event requests and a Delay Queue (DQ) is used for suspended tasks. In the double queue approach, when a task is initiated it is placed in the RQ, and when it is suspended it is placed in the DQ. When an associated event arrives, the task is resumed by taking it from the DQ and inserting it into the RQ. The RQ acts as a conventional ready queue in this approach.

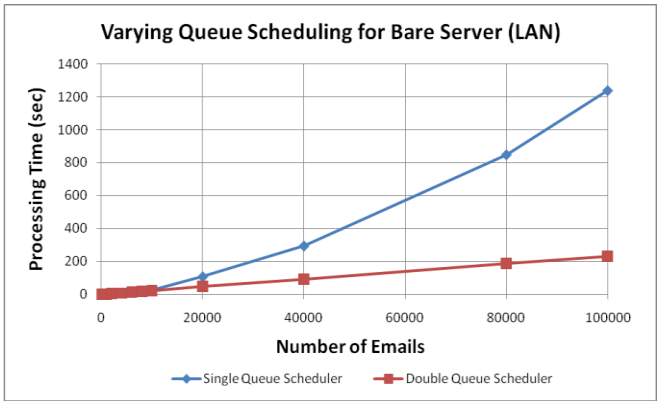

Figure 16. Performance Measures by Different Queue Scheduling Algorithms

Figure 16 demonstrates the effect on processing time for a quantity of 100,000 emails when these two queuing approaches are used. The double queue scheduling resulted in 5.3 times faster in processing time than a single queue approach. As shown in Figure 17, the maximum queue size is reduced from 488 to 137 in double queue scheduling, which resulted in faster performance for the bare server.

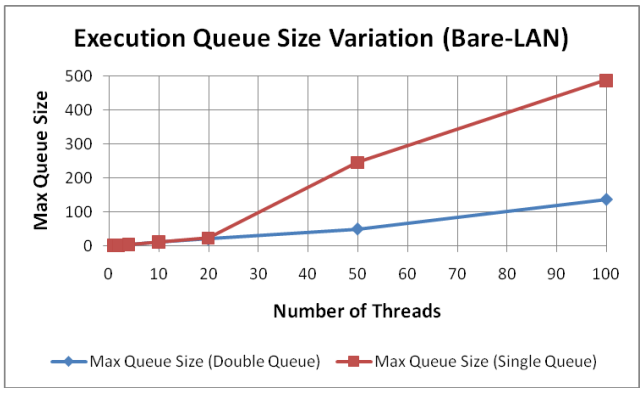

Figure 17. Performance Measurements for Different Queue Sizes vs. Number of Threads.

V. CONCLUSION

This research and paper has studied the performance of a bare machine (OS-less) email server and compared it with a functionally equivalent Java server prototype and two commercial servers (AxiGen and ShareMailPro). LAN/WAN measurements, internal message protocol timings, and improved task scheduling methods were presented. The LAN measurements indicated that the bare server performs 1.99 times faster than the Java-based server, and 2.36 times faster than the AxiGen server. For WAN measurements, the bare server performed 1.8 times faster than the Java server and 1.86 times faster than the Shared Mail Pro server. The internal handshake timings, as measured, identified the protocol timing points that are critical to performance. The bare server's performance was also shown to improve by a factor of 5.3 by using a double queue scheduling technique. These results enabled us to identify several novel bare machine architectural characteristics that could be exploited when designing high-performance email application-specific servers.

ACKNOWLEDGMENT

This research team is grateful to the late Dr. Frank Anger of NSF for his encouragement and support of early bare machine research, supported by NSF SGER grant CCR-0120155.

REFERENCES

[1] D. R. Engler and M.F. Kaashoek, "Exterminate All Operating System Abstractions," 5th Workshop on Hot Topics in Operating Systems, USENIX, May 1995, p. 78.

[2] J. Erman, A. Mahanti, M. Arlitt and C. Williamson, "Identifying and Discriminating Between Web and Peer-to-Peer Traffic in the Network Core," Network Issues in the Web, WWW 2007, 8 May, Banff, Alberta, Canada

[3] B. Ford, M. Hibler, J. Lepreau, R. McGrath, and P. Tullman, "Interface and Execution Models in the Fluke Kernel," Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, USENIX Technical Program, New Orleans, LA, February 1999, pp. 101-115.

[4] B. Ford, and R. Cox, Vx32: "Lightweight User-level Sandboxing on the x86", USENIX Annual Technical Conference, USENIX, Boston, MA, June 2008.

[5] G. Ford, R. Karne, A. Wijesinha, and P. Appiah-Kubi, "The Design and Implementation of a bare PC Email Server," 33rd Annual IEEE International Computer Software and Applications Conference (CompSAC 2009), Seattle, WA, July 2009, accepted for publication.

[6] L. He, R. K. Karne, and A. L. Wijesinha, "Design and performance of a bare PC web server," International Journal of Computer and Applications, Acta Press, June 2008.

[7] L. He, R. K. Karne, and A. L. Wijesinha, "A Study of bare PC Web Server Performance for Workloads with Dynamic and Static Content," Symposium on Advances on High Performance Computing and Networking (AHPCN-09), The 11th IEEE International Conference on High Performance Computing and Communications, June 2009.

[8] 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, ACM, San Diego, CA, October 2005, pp. 55-61.

[9] R. K. Karne, K. V. Jaganathan, and T. Ahmed, "How to Run C++ Applications on a Bare PC?" Proceedings of the 6th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, IEEE Computer Society, Washington DC, 2005, pp. 50-55.

[10] R. K. Karne, "Application-Oriented Object Architecture: A Revolutionary Approach," 6th International Conference, HPC Asia 2002, Centre for Development of Advanced Computing, Bangalore, Karnataka, India, December 2002.

[11] G. H. Khaksari, A. L. Wijesinha, R. K. Karne, L. He, and S. Girumala, "A Peer-to-Peer bare PC VoIP Application," Proceedings of the IEEE Consumer and Communications and Networking Conference, IEEE Press, Las Vegas, NV, 2007.

[12] V. S. Pai, P. Druschel, and W. Zwaenepoel, "Flash: An efficient and portable web server," Proc. Usenix 1999 Annual Technical Conference, pp. 199–212, Monterey, CA.

[13] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson and B. Tierney, "A First Look at Modern Enterprise Traffic," Internet Measurement Conference 2005, USENIX Association, Proceedings of the 5th ACM SIGCOMM, October 2005.

[14] A. Pathak, S. Roy, and Y. Hu, "A Case for a Spam-Aware Mail Server Architecture," 4th Conference on Email and Anti-Spam (CEAS) 2007, Microsoft Corporation, Mountain View, CA, August 2007.

[15] "Tiny OS," Tiny OS Open Technology Alliance, University of California, Berkeley, CA, 2004, http://www.tinyos.net/

[16] T. Venton, M. Miller, R. Kalla, and A. Blanchard, "A Linux-based tool for hardware bring up, Linux development, and manufacturing," IBM Systems Journal., Vol. 44 (2), IBM Corporation, NY, 2005, pp. 319-330.

[17] "The OS kit project," School of Computing, University of Utah, Salt Lake, UT, June 2002, http://www.cs.utah.edu/flux/oskit