2010 12th IEEE International Conference on High Performance Computing and Communications

The Design and Performance of a bare PC Webmail Server

Patrick Appiah-Kubi, Ramesh K. Karne, and Alexander L. Wijesinha
Department of Computer & Information Sciences
Towson University
Towson, MD 21252, U.S.A.
{pappiahkubi, rkarne, awijesinha}@towson.edu

***Abstract*—We describe a Webmail server that runs on a bare PC without an operating system (OS) or kernel, and give details of its architecture, design, and implementation. We also present the results of experiments conducted in a test LAN environment to compare performance of the bare PC Webmail server with conventional Webmail servers Atmail and Mailtraq running on Linux and Windows respectively. Performance is evaluated by measuring the processing time for login requests; inbox requests with a varying number of emails; and composing or retrieving email messages and sending attachments of various sizes. We also measure the throughput for various sizes; and, under stress conditions, the processing and response times with a varying number of connections, and the total and average processing times for the POST command with a varying number of users. The results show that the performance of the bare PC Webmail server is significantly better than that of the OS-based servers. The bare PC Webmail server is an alternative to conventional Webmail systems, and its architecture and design features could be used as a basis for developing future high-performance systems.**

Keywords-bare machine computing; bare PC; application object; Webmail server; operating systems

I. INTRODUCTION

Webmail allows users to access their email through a Web browser. While many servers capable of handling Webmail exist, they typically require a conventional operating system (OS) or kernel for support. We describe a self-contained bare PC Webmail server that runs directly on the hardware without using any OS. The server, which is integrated with a PHP parser tool, is shown to outperform conventional Webmail servers Atmail and Mailtraq running on Linux and Windows respectively. The server is also not vulnerable to attacks targeting an underlying OS.

We provide details of the server architecture, design, and implementation, and include the results of experiments comparing its performance with that of the OS-based servers. The server architecture is based on threading techniques, delay/resume lists, and task stack mechanisms to provide efficient memory utilization and process control. The bare PC Webmail server application contains its own data execution knowledge and control, and does not require any other software support to run. Currently, the bare PC Webmail server runs on Intel Pentium 4 (or above) based PCs and only requires common general-purpose hardware including USB-based bootable devices, network interface cards, and USB-based persistent storage.

II. RELATED WORK

Many approaches have been developed to improve application performance by reducing OS overhead, bypassing the OS, or using similar techniques. These include Exokernel [1], Microkernel [2], Nano-kernel [3], OS-Kit [4], bare-metal Linux [5], IO-Lite [6], and sandboxing [7]. All of these approaches require some form of an OS or kernel.

In contrast, the bare machine computing (BMC) approach, also known as the dispersed operating system computing (DOSC) paradigm [8], is application-centric with no OS, and no centralized resource manager or controller running in the machine. BMC enables applications to directly communicate with the hardware [9]. BMC applications, frequently referred to as bare PC applications, are encapsulated in a single monolithic executable or application object (AO) [10] (stored on a portable storage medium such as a USB flash drive), which is capable of bootstrap loading and of providing its own resource management with no dependence on any external libraries or programs. Several bare PC applications have been built including a VoIP softphone [11], an Email server [12, 13], and Web servers with and without TLS [14, 15]. These applications have been shown to outperform their conventional counterparts.

The numerous Webmail systems in existence today include Webmail servers, email servers with Webmail interfaces, or packages installed on Web servers that enable access to email servers (Webmail clients). Atmail [16], MailTraq [17], Axigen [18], and Squirrelmail [19] are just a few examples of such Webmail systems. Several design and implementation issues relevant to improving the performance of the Open Webmail system are discussed in [20]. The bare PC Webmail server, whose architecture and design is based on the BMC approach, differs from the above and similar conventional OS-based Webmail servers and Webmail systems.

III. ARCHITECTURE

Fig. 1 illustrates the architecture of the bare PC Webmail server. For ease of reference, numeric labels are assigned to figure components. The server is initiated on a

978-0-7695-4214-0/10 $26.00 © 2010 IEEE
DOI 10.1109/HPCC.2010.116

bare machine by a boot program (**1**) read from a USB-bootable device. The initial sector contains a bootstrap loader that loads the menu program (**2**) followed by the AO. The AO starts by initializing various data structures, parameters, tasks, and objects, and control is then passed to the Main Task (**3**).

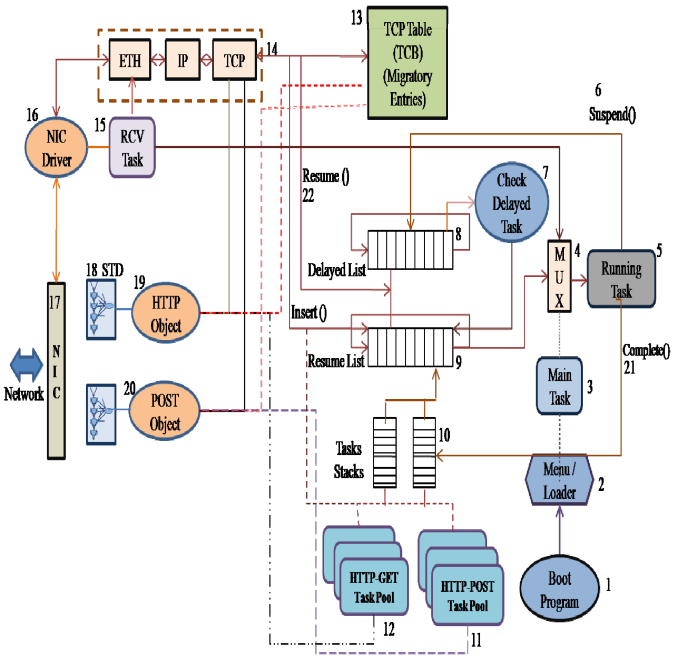

Figure 1. Bare PC Webmail server architecture.

The basic bare PC data structures used by the Main Task include a Delayed List (**8**), Resume List (**9**) and a Multiplexor mechanism (**4**). The latter switches between the Receive (RCV) Task (**15**) and tasks in the Resume List (HTTP-GET or POST), and selects the running task (**5**), which can be suspended (**6**) and returned to the Delayed List (**8**) when it is not running. The Main Task, which runs when the system is idle, or when switching between tasks, checks for delayed tasks (**7**), and places them in the Resume List. When a response arrives for a task in the Delayed List, a Resume () function (**22**) brings the tasks in the Delayed List into the Resume List.

The TCP Control Block/Table (TCB, **13**) is used to store relevant information derived from the TCP, IP and Ethernet headers (**14**). To communicate directly with the host Network Interface Card (NIC, **17**), a bare PC NIC driver was written (**16**). Two components were created within the bare PC Webmail server application to handle all Webmail functions: an HTTP POST Object (**20**) to send emails from clients to the server, and an HTTP GET Object (**19**) to deliver resident emails and files to clients.

At initialization, a task pool of HTTP GET (**12**) and HTTP POST (**11**) objects are created along with their associated TCB table entries for use by the server application. When a client request arrives, a GET or POST task is placed in the Resume Task List (**9**), and an active status flag is set within its associated TCB entry. The TCB entry contains all of the unique data attributes associated with the object, and its executable state information. When the task is complete (**21**), it is returned to its appropriate task stack (**10**) so that it can be reused later. The active flag in the associated TCB entry along with associated data fields are then reset. The "RCV task (**15**)" receives packets and processes them in a single thread of execution without any interruptions or process swapping until the status is updated in the TCB. The HTTP GET or POST tasks work in a similar manner. Other than the Main task, there are only the RCV task and HTTP GET/POST tasks making task scheduling simple. The scheduling mechanism is first-come/first-serve (FCFS) except for Suspend/Resume.

IV. Design

The user interface for the bare PC Webmail server is text-based, which enables a few configuration parameters to be specified at startup. The bare PC Webmail server integrates adaptations of bare PC Web and email servers with the PHP parser tool into a single AO. Flags are stored in memory to determine the type of request (Webmail or regular HTTP) based on the data header. Client-based PHP scripts were also designed for the bare PC Webmail server. Fig. 2 shows the PHP client-server interface design for the Webmail server. The user first requests a Login page and logs into the server. Next, a Mail page is displayed allowing the user to compose mail, check their Inbox and read messages, or logout.

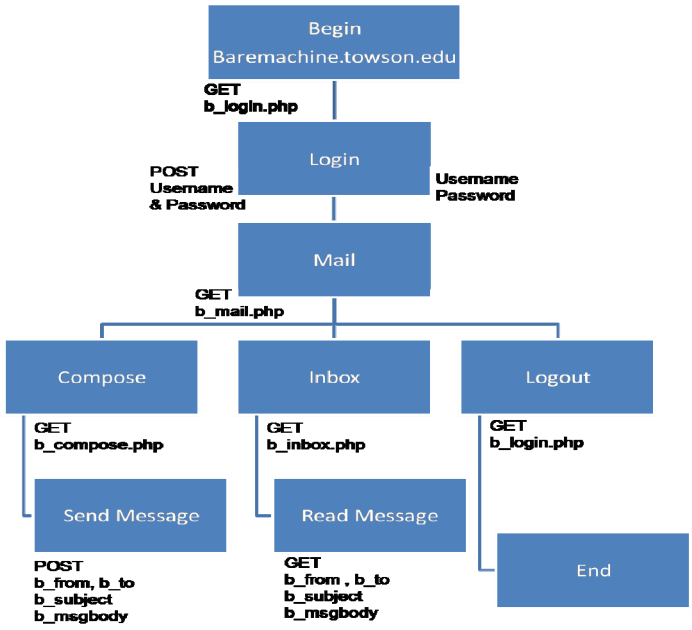

Figure 2. Client-Server interface design.

During the design of the bare PC Webmail server, each GET and POST command was implemented using State Transition Diagrams as in Fig. 1 **(18)**. Each state models HTTP and TCP protocol state transitions corresponding to

the messages exchanged by client and server. The client sets up a TCP connection and sends either an HTTP GET or POST request. The application, HTTP, and TCP protocols are intertwined in the underlying implementation. In case of a GET, the server responds with the data that may be contained in several messages. For POST, the client sends data to the server. The server's 200 OK HTTP message terminates the data transfer, and the TCP connection is closed as usual.

The lean PHP-parser built into the Webmail server parses the fields in the GET or POST data received and relays the data contents to the email component of the server. Unique field names in the form input that represent FROM, TO, SUBJECT, ATTACHMENT and MSGBODY were specified in the PHP script files to aid in parsing the received data to extract the appropriate commands for email storage and retrieval. Certain pointers/flags are kept in memory for efficiency. For example, pointers are specified for b_inbox.php, b_attach.php and b_readmsg.php files enabling them to be parsed once and dynamically modified to insert the messages.

Email storage and retrieval are based on special PHP session keys assigned automatically to each user at login (for reuse in subsequent requests) and dynamically stored in a session table using the User ID, Session Flag, and Session String generated by using random number generators. To retrieve an Inbox message from email storage, the PHP session variables are parsed when the Inbox request arrives. This enables the system to detect the appropriate User ID, which facilitates message retrieval. If another user attempts a login while a previous user is still active, a different PHP session key is assigned to that user so that concurrent user connections can be correctly handled by the server. The server was also designed to handle large messages by extracting the content-length and matching it to the total byte count of messages received for multiple POST data continuations.

V. IMPLEMENTATION

The bare PC Webmail application does not use any OS-related libraries or system calls. However, the application itself is developed using a standard MS-Windows environment and written in Visual C++ (without any *.h files), and the MASM Assembler. Most of the direct hardware interfaces are implemented in C/assembly language using software interrupts. The size of the assembly code is approximately 1,800 lines. These direct hardware interfaces include: display, keyboard, timers, task management, NIC, and real/protected mode switching. The 3COM 905CX NIC driver code is approximately 1400 lines of assembly code, with the rest of the code written in C. Similarly, the USB driver uses approximately 133 lines of assembly code, with the rest of the code written in C.

The code implementing the Webmail server architecture depicted in Fig. 1 is written in C++ in an object-oriented fashion. The size of the software is approximately 35,243 lines of code including 11,000 lines of comments, and 14,000 executable statements. This yields a single monolithic executable AO consisting of 792 sectors of code (405,504 bytes), which is placed on the USB. It includes the boot program, startup menu, AO executable, and the persistent file system (used for user profiles, emails, and attachments). The content on the USB is generated by a tool (designed and run on MS-Windows) that creates the bootable bare PC application for deployment with the boot load sector, and copies the executable and associated files to the USB. The tool consists of 469 lines of C++ code.

VI. PERFORMANCE MEASUREMENTS

A. Experimental Setup

The experiments to compare performance of the bare PC Webmail server with the Atmail and Mailtraq Webmail servers running on Linux and Windows respectively were conducted in a dedicated Ethernet test LAN using four Dell Optiplex GX520 PCs. Each PC had an Intel Pentium 4, 3.2GHz Processor, 1GB RAM and a 3Com EtherLink XL 10/100 NIC card. The computer serving as the client had Internet Explorer and a Wireshark network analyzer tool running on Windows. The remaining computers ran the bare PC, Linux, and Windows Webmail servers. All unnecessary services, processes, and programs on the OS-based systems were disabled.

B. Results

Figs. 3 and 4 show respective processing times for the GET and POST login request commands. The client/server TCP connection is closed after each GET request, whereas it is kept open for multiple POST commands. It can be seen that the TCP connection times with GET are negligible for all servers, and also that the bare and Linux servers respond (i.e. ACK) to the GET request in a reasonable amount of time (0.153 and 0.361 milliseconds respectively). However, the Windows server takes 115 milliseconds. The time from GET to first data is respectively 0.258, 183, and 85 milliseconds for the bare, Windows, and Linux servers. The bare server responds with little delay because of its efficient task design and protocol intertwining (RCV and GET each have a single thread of execution). For POST, the dominant time is between the arrival of POST data and server's response (i.e. 302 found). The respective timings are 0.613, 329 and 223 milliseconds for the bare, Windows and Linux servers. The bare server's low timing for the POST command is due to the POST task responding to the request in a single thread of execution without interruption and process switching

Fig. 5 shows the email message processing time reported by Wireshark for a single compose request, whose message size is varied from 1000 bytes to 120,000 bytes. The bare server performs well for message sizes up to 60,000 bytes, but its processing time increases gradually thereafter. The USB file I/O for Webmail requests is contributing to most of the processing time in

the bare server (the other servers store files on the hard disk).

Fig. 6 compares the processing times for an inbox request. The number of emails in the inbox varied from 1 to 10 while the email size remained constant. As seen in the figure, the bare Webmail server processes the inbox request faster than the other servers; increasing the number of emails has very little effect on its processing time.

Fig. 7 compares the processing times with varying message sizes for email message retrieval. In all cases, the bare Webmail server clearly outperforms the other servers, again as a result of its efficient task design and protocol intertwining. For larger amounts of transmitted data (i.e. 60,000 bytes or more), the GET task will be suspended for longer amounts of time while waiting for ACKs from the client. This results in the higher processing times seen in the figure for sizes exceeding 60,000 bytes.

The performance when sending MIME-Encoded attachment messages is compared in Fig. 8, with varying message sizes from 1000 to 120,000 bytes. The bare server again performs better than the OS-based servers. The dramatic increase in processing time for message size after 20,000 bytes is caused by the write operation in the USB. Each USB write operation is limited to approximately 20,000 bytes of data; thus, it requires multiple write operation commands for large amounts of data. The processing times for the other Webmail servers (although larger) do not vary much with increasing message size since they store files on the hard disk. Fig. 9 shows that the bare server has higher throughput than the other servers for all sizes of messages as expected.

The HTTPERF tool was used to measure the performance of the servers under stress by varying the number of concurrent connections from 100 to 2000 at a rate of 1000 connections/s. Fig. 10 shows the CPU time at the client, and it is lowest for the bare server since it sends the data fastest to the client. Fig. 11 compares the total processing time versus the number of connections for the Webmail servers, and Fig. 12 shows the corresponding response times. The bare server has lower processing and response times, and is more stable compared to the other servers. Fig. 11 shows a spike in processing times between 400 and 600 connections for the Linux server, and it also shows the degradation of the Windows server as the number of connections increases (Windows server performance is omitted in Fig. 12 due to instability). Additionally, the Webstress tool was used to measure the performance under stress for POST with a varying number of users. Figs. 13 and 14 show respectively the total (all requests) and average (per request) processing time, which are negligible for the bare server but higher for the others.

VII. CONCLUSION

In this paper, we presented the architecture, design, implementation, and performance of a bare PC Webmail server. The server architecture and design is based on the BMC approach in which AOs run on the hardware with no OS and no centralized resource manager or controller of any kind. In this server, only the RCV task and HTTP GET/POST tasks compete for the CPU, which simplifies task scheduling. Experiments in a dedicated test LAN using several common Webmail transactions, including some under stress conditions, show that the bare PC Webmail server performs significantly better than conventional Webmail servers running on Windows and Linux. The novel architecture and design features of the bare PC Webmail server could be adapted to build high-performance servers for other applications based on the BMC approach.

REFERENCES

[1] D. R. Engler and M.F. Kaashoek, "Exterminate all operating system abstractions," Fifth Workshop on Hot Topics in Operating Systems, USENIX, Orcas Island, WA, May 1995, p. 78.

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

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

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

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

[6] V. S. Pai, P. Druschel, and Zwaenepoel. "IO-Lite: A Unified I/O Buffering and Caching System," ACM Transactions on Computer Systems, Vol.18 (1), ACM, February 2000, pp. 37-66.

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

[8] R. K. Karne, K. V. Jaganathan, N. Rosa, 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. 55-62.

[9] R. K. Karne, K. V. Jaganathan, and T. Ahmed, "How to run C++ Applications on a bare PC", SNPD 2005, Proceedings of SNPD 2005, 6th ACIS International Conference, IEEE, May 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 (CCNC), IEEE Press, Las Vegas, NV, 2007, pp. 803-807.

[12] G. Ford, R. Karne, A. L. 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.

[13] G. Ford, R. Karne, A. L. Wijesinha, and P. Appaiah-Kubi, The Performance of a Bare Machine Email Server, 21st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2009.

[14] A. Emdadi, R. K. Karne, and A. L. Wijesinha, “Implementing the TLS Protocol on a Bare PC,” 2nd International Conference on Computer Research and Development (ICCRD), May 2010.

[15] L. He, R. Karne, and A. Wijesinha, "The Design and Performance of a Bare PC Web Server", International Journal of Computers and Their Applications, vol. 15, June 2008, pp. 100 - 112.

[16] Atmail-Linux Email Server, http://atmail.com/linux-email-server/

[17] Mailtraq email server-the complete email server, http://www.mailtraq.com/

[18] Mail Server for Linux & Windows | Axigen, http://www.axigen.com/

[19] SquirrelMail-Webmail for Nuts!, http://squirrelmail.org/

[20] C. Tung and S. Tsai, “Tuning Webmail Performance-The Design and Implementation of Open Webmail,” National Cheng Kung University, www.openwebmail.org

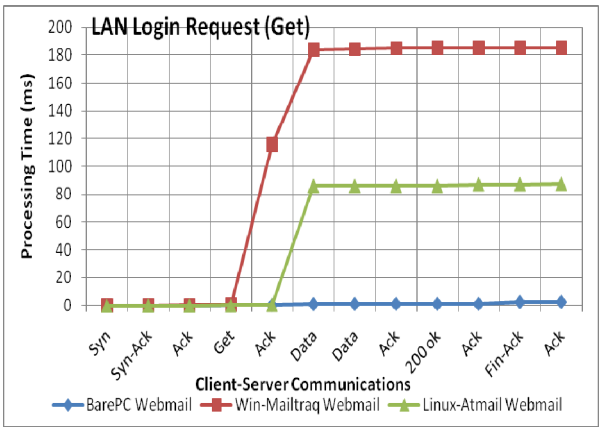

Figure 3. Processing time: login request (GET).

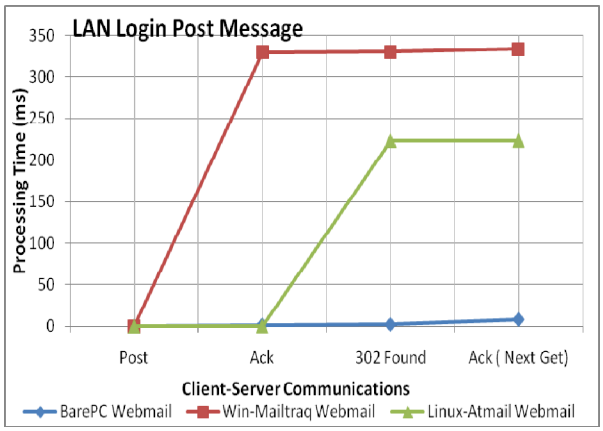

Figure 4. Processing time: login request (POST).

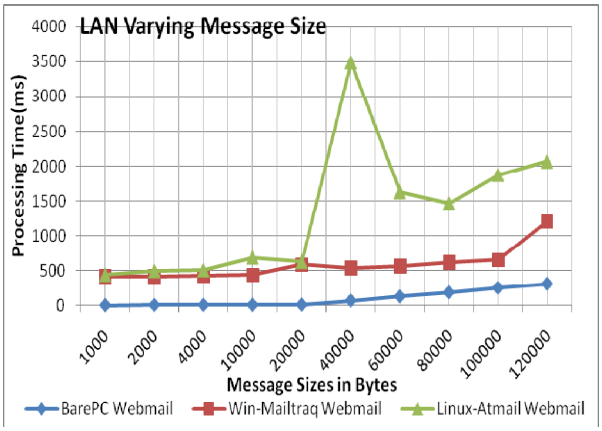

Figure 5. Processing time: compose (varying message sizes).

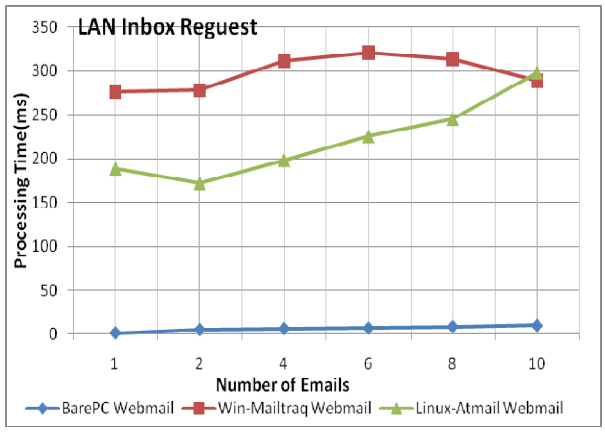

Figure 6. Processing time: inbox request (varying number of emails).

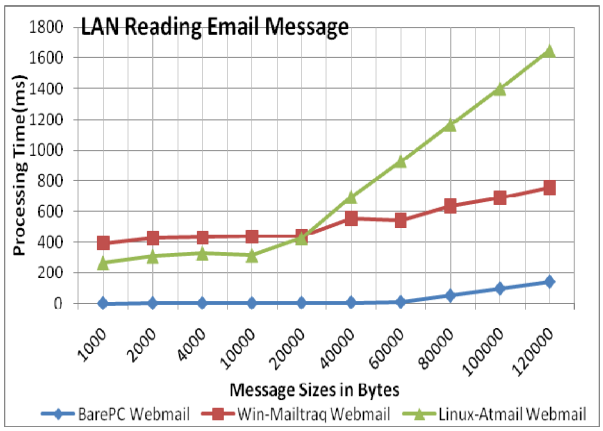

Figure 7. Processing time: email retrieval (varying message size).

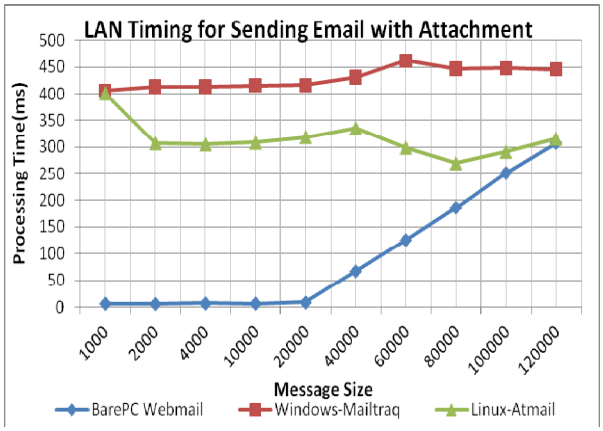

Figure 8. Processing time: email attachment (varying message size).

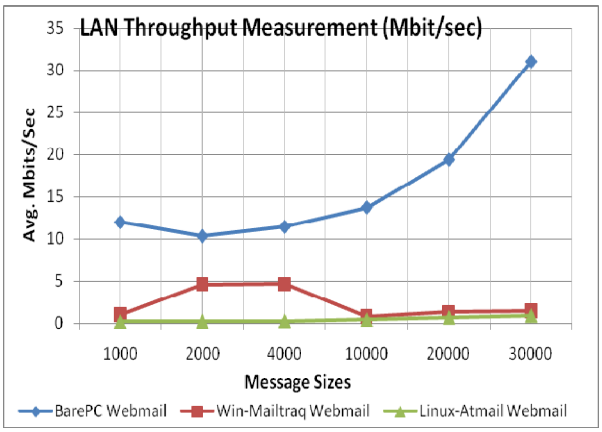

Figure 9. Throughput:

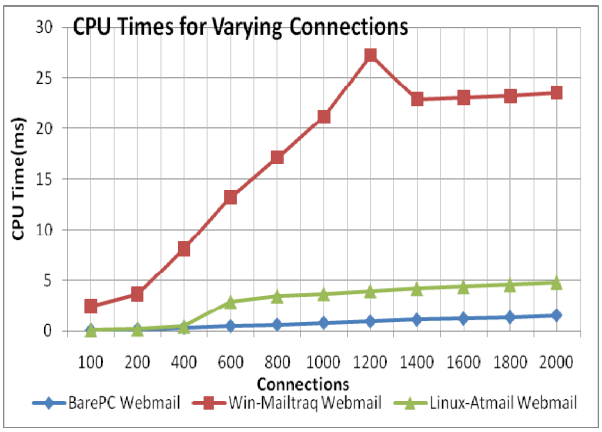

Figure 10. CPU processing time: client (varying number of connections)

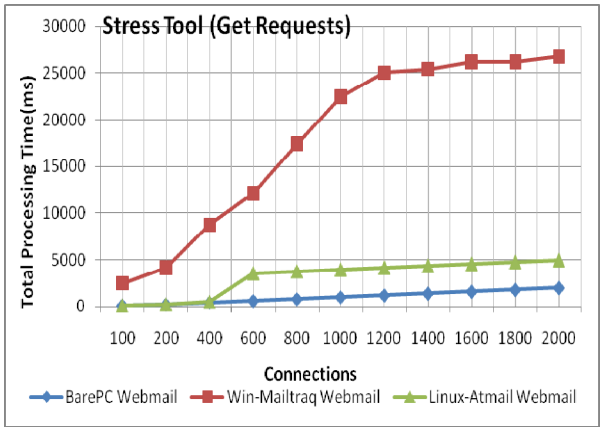

Figure 11. Processing time: GET (varying number of connections).

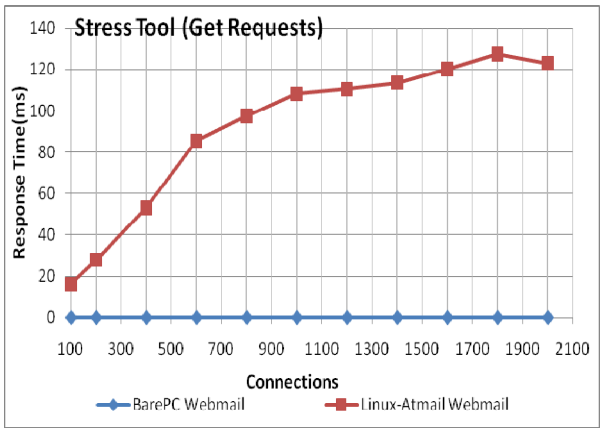

Figure 12. Response time: GET (varying number of connections).

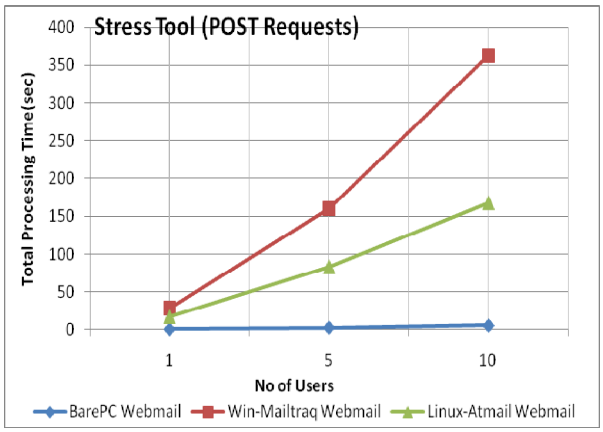

Figure 13. Total processing time: POST (varying number of users).

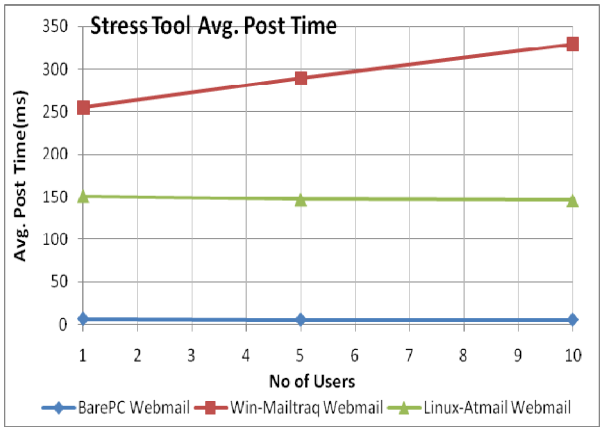

Figure 14. Average processing time: POST (varying number of users).