TOWSON UNIVERSITY

COLLEGE OF GRADUATE STUDIES AND RESEARCH

LEAN SECURE WEBMAIL SERVER ON A BARE PC

By

Patrick Appiah-Kubi

A Dissertation

Presented to the faculty of

Towson University

In partial fulfillment

Of the requirements for the degree

Doctor of Science

In Information Technology

December 2011

Towson University

Towson, Maryland 21252

TOWSON UNIVERSITY

COLLEGE OF GRADUATE STUDIES AND RESEARCH

DISSERTATION APPROVAL PAGE

This is to certify that the Dissertation prepared by Patrick Appiah-Kubi., entitled "Lean Secure Webmail Server on a Bare PC" has been approved by this committee as satisfactory completion of the requirement for the degree of Doctor of Science in Information Technology.

Dr. Ramesh K. Karne Chair, Dissertation Committee	Signature	11/11/2011 Date
Dr. Alexander L. Wijesinha Committee Member	Signature	11/11/11 Date
Dr. Siddharth Kaza Committee Member	Signature	11/18/2011 Date
Dr. Yeong-Tae Song Committee Member	Signature	11/28/2011 Date
Dr. Lawrence Shirley Acting Dean, College of Graduate Studies and Research	Signature	29 Nov 2011 Date

Acknowledgements

First and foremost, I would like to give thanks and praise to God almighty who is the creator of the universe for giving me the strength and will to complete this work. I would then express my sincere appreciation to all those who have supported my efforts to complete this dissertation. I am greatly appreciative of my research committee Dr. Ramesh K. Karne (chair), Dr. Alexander L. Wijesinha, Dr. Yeong-Tae Song, and Dr. Siddharth Kaza for supporting this research. I am especially thankful to Dr. Karne and Dr. Wijesinha for all the long hours in lab, and support and advice I have received throughout this dissertation research. My gratitude goes to the College of Graduate Studies and Research at Towson University, for facilitating this work. Gratitude also goes to the late Dr. Frank Anger of NSF for his encouragement and great support of early bare machine research, supported by NSF SGER grant CCR-0120155. Finally, my love and appreciation goes to my wife, siblings and family for their support and prayers throughout this work.

ABSTRACT

LEAN SECURE WEBMAIL SERVER ON A BARE PC

Patrick Appiah-Kubi

Webmail server design and implementation on a bare PC poses daunting challenges and innovative opportunities that are revolutionary in server designs. Making the Webmail server secure through the Transport Layer Security (TLS) makes the design even more immense for a bare PC. This thesis focused on the design, implementation, performance and security analysis of a Webmail server that runs on a bare PC, without an operating system (OS) or kernel. The bare PC approach poses numerous design challenges which are solved in this research. Secure Webmail servers are complex, large and require intricate components to develop. As a result a lean system was developed for this research. The lean concept proliferates through architecture, design and implementation of the Webmail server. The lean concept also helps to build small protocol suit, intertwining of protocols, limited requirement space, simple user interfaces and minimal design options. The dissertation provides a detail architectural design, steps of implementation, experimental setup and the results of experiments conducted in a LAN and WAN environments. The performance of the bare PC Webmail server is compared with conventional Webmail servers. Performance is evaluated by measuring the processing time for login requests; inbox requests with a varying number of emails; and composing emails, retrieving email messages and sending attachments of various sizes. The throughput for various email sizes, stress test, the processing time, response time with a varying number of connections, and the total time, the average processing time for the POST command with a varying number of users, the CPU Utilization and load distribution are all measured and analyzed. The results show that the performance of the bare PC Webmail server is significantly better than that of the OS-based servers.

Designing secure systems on a bare PC is a major milestone that must be crossed. The TLS protocol in its own dimension is a complicated protocol that needs thorough understanding in order to build on a bare PC platform. The secure Webmail server development of bare PC posed many demanding hurdles that needed careful attention. Issues with encryption/decryption algorithms, handshake, and certificate verification, private/public key and key cutting algorithms needed to be resolved. Novel design features of the server include intertwining the TLS, HTTP and TCP protocols to reduce inter-layer communication overhead, and using a separate TLS task per connection to improve performance. This novel architectural and design features could be used as a basis for developing future high performance systems. It could also serve as a stepping stone to the development of new paradigms in the computing environment.

Keywords

Bare machine computing, Bare PC, HTTP GET, HTTP POST, PHP parser, TLS, Application object, Webmail server, Operating system.

Table of Contents

List of Tables

List of Figures

I. BACKGROUND

A. Problem Overview

According to computer historians, computing paradigms have evolved in many directions and dimensions including hardware and software, since the inception of mainframes. This unanticipated offering of tremendous speed improvement and reduced chip sizes over the decade has surpassed the software optimizer expectations and their design objectives. Meanwhile, software developers consistently try to improve performance of execution by means of efficient algorithms, variety of computer architectures, and programming paradigms.

Computer evolutions have now taken new paths that have resulted in more varieties in terms of computer hardware, architectures, software, computing environments, programming tools, and techniques. These surges in alternate computing techniques and mechanisms have also generated complexity and heterogeneity in many aspects of computer systems. This has contributed to the rapid obsolescence and tremendous waste in hardware, software, people skills, and tools. The rapid obsolescence today have reached extraneous heights in every electronic and computer product, which is now generating electronic dump and waste making hardware unmanageable and hard to recycle with existing environmentally safe techniques.

The hardware and software characteristics that have dramatically changed over the years and resulted in more complex and unstable systems have prompted us to develop the Bare Machine Computing (BMC) or bare PC paradigm to leverage on these changes. To fully leverage on the potentials of bare PC, many application objects (AO) have being developed and tested on bare PC environments. Among the numerous studies of bare PC

applications is lean secure bare PC Webmail Server, which is the main focus of this research.

B. Bare Machine Computing Background

Bare machine computing (BMC), invented by Dr. Karne at Towson University, is a computing paradigm wherein a computer application runs on a bare machine without any operating system (OS) or other software loaded in the machine, except the application itself. The applications can be carried on a portable storage medium such as a USB drive or a CD and run on a bare machine (OS-less device) without using a hard disk as shown in figure 1. Currently, these "carry-on" applications run on PCs with an IA32 CPU; however this paradigm applies to other CPU architectures and devices that are bare. This approach offers enormous benefits compared to traditional OS-based computing today. Bare computing provides an application-centric, low-cost, simple and secure solution for running applications without an operating system and independent of any support software.

Any computer with a processor and a memory can be defined as a bare machine. If all bare machines use the same processor architecture, then a bare machine application can run on any bare machine. Bare machine computing provides enormous benefits including simplicity, independence from computing environments, smaller code sizes, longevity, application ownership, and inherent built-in security.

Another view of bare PC can be envisioned as a computer without any ownership or valuable resources in the box, or a cell phone like device without any embedded functionality. The applications are decoupled from the computer or a device and carried on a USB flash drive to run in any computer or device and at any given location. This

notion also makes people carry-on their own applications and run them on any public computer or device available anywhere in the world. This decoupling of computer applications with computer environments offers ownership, security, and mobility in computing and results in a productive information technology era in the future.

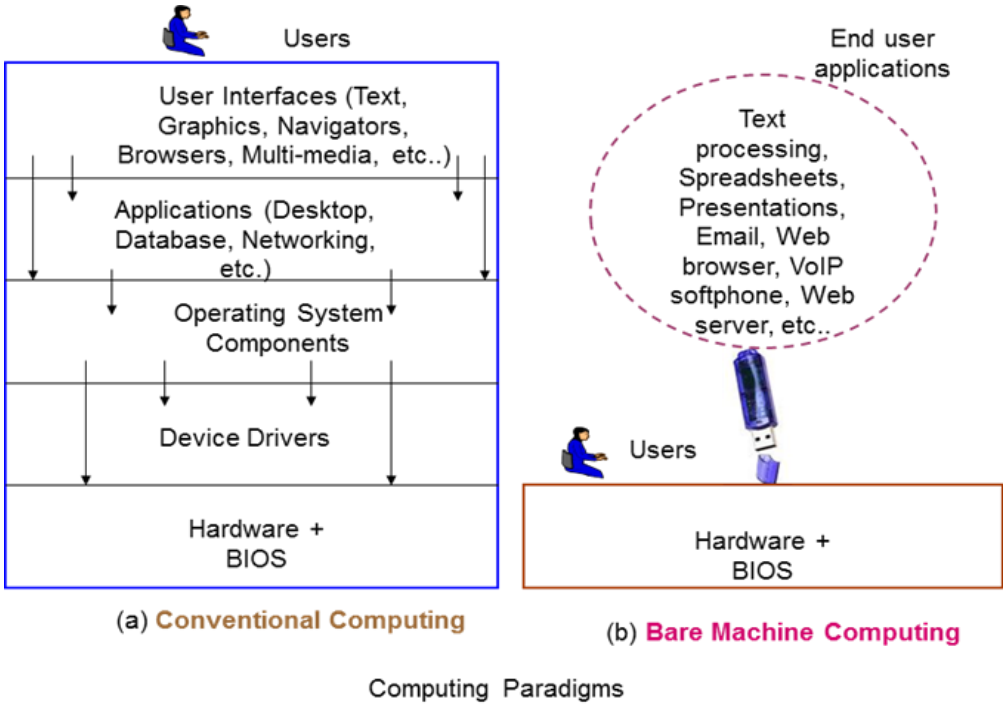

Figure 1. Conventional versus Bare Machine Computing

C. Feasibility of Bare Machine Computing

In order to make a bare machine feasible, one must assume the processor architecture as a default standard and it must remain stable over a reasonably longer period of time. In addition, the processor or its inherited flavors should be used in designing computer systems and pervasive devices. This may sound absurd and an impossible dream to achieve due to diverse needs and environments required for a given computer system or a pervasive device. This is quite feasible today as computer technology provides fast processors doubling in speed in every two years in addition to providing faster and denser

memory technology. The performance twisting and tweaking of computer applications can be achieved through semiconductor technology improvements instead of complex computing environments and program optimizations.

The bare PC approach can be achieved by limiting the proliferation of computer languages to only a few. The proliferation of computer languages causes applications to be written and re-written in many languages to implement the same domain application. This creates tremendous waste and obsolescence of applications on a frequent basis. When bare machine becomes practical, then applications will be written once and used for a long period of time. However, applications can still be enhanced or extended depending on a need for new and enhanced requirements, without totally abandoning the old applications.

Bare machine feasibility also depends on limiting or avoiding the non-standard proliferation of I/O devices for computer systems and pervasive devices. If all peripheral devices use standard USB interfaces, then it will be naturally suitable for bare PC paradigm. Accidentally, this is already happening with the emergence of USB technology in areas of mass storage, network interface cards, keyboards, audio, video, graphics monitors, and mouse devices. When these I/O interfaces emerge as USB devices, then there is no need to put these devices onboard, and they can be placed outside. When these devices are offered as a combined function in future USB devices, then one can carry just one USB device to encompass all I/O functions in the future. In such situations, the computer box simply becomes a CPU and Memory unit thus resulting in a bare machine.

In addition to adopting the above three significant attributes such as single CPU architecture, limited number of programming languages, and standard USB I/O interfaces, there is also a need for making the bare machine independent of any operating system or environment. The applications should directly communicate with the hardware components, thus making applications independent of any computing environment or operating system. This makes applications totally independent of other entities thus making them achieve longevity in addition to simplicity and smaller code sizes. When applications become simple and small, they inherently provide more security and ownership to the user.

D. Design Principles of the Bare Machine Computing

The primary objectives of bare machine computing are to achieve simplicity, smaller code sizes, longevity, pervasiveness, ownership/control, and security. The following design principles are used to achieve these goals and to implement bare machine applications.

- *Application-Object-Centric Approach*

The application program is a single object referred to as an application object (AO). For example, an email client, a webmail server, or a softphone client (or a combination of several such applications) is an AO. When a bare machine is booted using a portable storage medium such as a USB drive, an AO is loaded and run by a user. A single AO runs in the bare machine at any given time. However, a single AO can consist of many processes or tasks running at the same time, which are integral part of the AO.

- *No External Software*

No external software can be loaded into memory when an AO is running. That means, there is only one AO running in the machine at any given time. This also implies that there is no operating system or any other system program running in the machine other than the AO itself.

- *Direct Interfaces to I/O*

An AO has direct I/O interfaces. This means there is an application programming interface (API) to the hardware. For example, an AO (which is a self-contained application program) can communicate directly through this API to read data from an Ethernet packet through the network card. An AO can stop and start the CPU, write to the video buffer, invoke an interrupt, process an error or exception, manipulate the interrupt descriptor table (IDT), write to an audio device, write to a video device, load data into memory, read from memory and so on.

- *Self-Boot and Load*

A typical bare PC AO consists of its self-boot code, loader, and application program as bare PC does not have any OS controlling the system. When a machine is powered, it will be booted from the boot device such as a USB flash drive where an AO resides. The BIOS in the bare machine loads the boot sector into memory for execution. This boot code loads the AO from the (user controlled portable) boot device and jumps to the application program within the AO. User access mechanisms and authentication programs can be run first to validate the user and the AO before the actual execution can begin.

- *Self-Cleanup*

After execution of an AO, RAM memory in the bare machine (code and program data) is “erased”, since memory is a volatile storage device used for storing only temporary data and instructions. The machine can then be shut down without leaving any traces of execution after the user removes the USB drive containing the AO.

- *Static Binding*

An AO is statically bound at compile time. There is no ability for anyone to modify the code as there is no support for dynamic linking and binding. The entire code is a single monolithic executable application throughout its life span.

- *Self-Managed Dynamic Memory*

In OS-based applications, the dynamic memory is allocated by an application programmer and managed by the memory management part of an OS. An application programmer and the program have no control on the allocated and de-allocated memory at any given point in time. In bare PC applications, an AO manages its own dynamic memory. The memory in this approach is real memory and there is no need for virtual memory and paging, thus reducing application complexity. This also simplifies the AO as an AO programmer defines and organizes the memory space. An AO programmer can also impose strict memory access. As real memory is cheap and easy to extend to its maximum limits, lack of memory is not an issue in bare PC. In addition, as an AO is small in size, it does not require more memory during execution due to its AO-centric approach. In an OS-based (or even in a lean kernel–based) system, memory is managed by the kernel thus taking the control away from an application.

- *No Shared Dynamic Memory*

Static memory can be shared between processes or tasks running in the AO. This sharing of memory between processes is facilitated to pass parameters between processes only. All shared variables are restricted to pass only between two processes thus avoiding concurrency control and locking mechanisms. The static shared memory is small and located in real mode to provide limited access and access through the hardware API only. Shared dynamic memory is not offered to avoid concurrency control issues and security vulnerabilities.

- *No Concurrency Control*

Concurrency control mechanisms are avoided by design in bare PC applications. Whenever, there is a need for communication between two AOs, asynchronous message passing is used. In a single AO environment, only one and only one process at a time is allowed to access given resources and no more than two processes in a single AO can share the same resources. An AO programmer avoids concurrency control by design using additional memory and extra logic to avoid concurrency control. In an OS based system, this is not possible to achieve as the OS is interrupt-driven and applications have no control on the sequence of execution. In our system, interrupts and CPU are in total control of an AO programmer.

- *No Local Persistent Storage*

A bare machine requires no local hard disk for persistent storage. An AO consists of its program and data and it is loaded into main memory. The files are stored elsewhere (either on a network server or an USB flash drive). As the USB flash drives continue to

increase in capacity, it is possible to store even larger AOs and the necessary files on the same unit. This capability maybe viewed as a form of removable persistent storage.

During the execution of an AO, there will be no swapping of processes and there is no need for storing anything on a local hard disk, even when there is a hard disk in the machine. An AO using main memory provides the ability to design simpler bare machine applications and eliminates the need for long-term storage of resources that need to be protected as in conventional systems. This results in an additional level of security for bare machines (they only contain processors and memory).

- *Self-Process Management*

An AO programmer can create, run, and terminate a process. One or more processes can be created as a process pool at compile time. Each AO may have its own requirements for creating and managing their own number of processes and for inter-process communications. The process pool is inserted into a stack and used as needed by removing a process from the top of the stack and inserting it into a circular list of processes. When a process is in the circular list, it is ready to execute. There can be many processes in this circular list that will then be executed on a first-come-first-serve (FCFS) basis. When a process is complete, it is returned to the stack and ready for reuse.

In addition to regular processes, there are two other processes that are required in bare PC. The MAIN process is the first process that runs soon after start of an AO and runs until an AO terminates. This process simply invokes all other processes and manages them from the circular list. In addition, there is one RECEIVE process which is dedicated to process messages that arrive from the network. The MAIN and RECEIVE processes

have the highest priority and they are treated differently from the processes in the circular list.

Processes other than MAIN and RECEIVE are created by an AO programmer. There can be many types of other processes which can be placed in the circular list for processing in a FCFS manner. For example, there could be Webmail server processes, web server request processes, and VoIP client processes all simultaneously running in a single AO. The AO programmer has total control over these processes and can run, suspend, or terminate them as needed. The creation of all the required processes are done at compile time as a task pool and inserted into their appropriate stacks.

The task structure designed for bare PC applications is novel and it is optimized for simplicity, ease of use, and performance. This process structure approach is used in bare PC as the AO programmer is completely aware of the AO's behavior during its execution. We adopt a simple scheduling strategy as follows. When a process has no work to do, it is suspended with a delay and returned to the MAIN process. When a process has productive work to do, it will continue execution without an interruption. A process in a circular list will not get executed until its delay expires or an event occurs, which leads to resuming the process (and ignoring the remaining delay). The delay coupled with the resume function achieves optimal CPU scheduling and results in much better performance than on OS-based machines. We have conducted many experiments using different networking applications to measure bare PC performance (most of the results have been previously published).

An AO programmer carries an additional burden of managing the processes that are required in an AO. In fact, an AO programmer serves as a "systems" and an "application"

programmer. One advantage to the programmer is that process knowledge hidden in the OS becomes part of the application thus giving more control and independence to the application and making it simpler to manage. By avoiding the OS middleware for process management, we achieve total autonomy in program execution and related controls. It is likely that this approach will make it easier to develop more secure bare machine applications. When applications are written with full knowledge of execution processes, behavior and necessary resources, and the entire system is smaller, it should behave more predictably and be easier to secure than applications running on conventional (including lean OS-based) systems.

The actual implementation of processes are done using the task structure and task segment state (TSS) memory provided by Intel's IA32 architecture. The processes in bare PC behave similar to other processes in a UNIX or Linux environment. There is no need for thread implementation in bare PC as process management is very efficient and process communication is achieved through static memory located in real mode and accessed through the hardware API.

- *Single Thread of Execution*

When an event occurs or a message is received from the network, the processing of this event or a message is done through a single thread of execution until it is complete. For example, an Ethernet message arriving from the network will go through Ethernet, IP, TCP, Application levels before returning to the MAIN process. Appropriate data structures and state changes are made when processing an event or a message without waiting for other events. State transition diagrams are used to keep track of events and messages and their impact on process flow. Events and messages are processed and

managed without creating any dependencies or the need for complex concurrency control. The single thread of execution is very efficient as it is doing productive work without interruptions. An AO programmer will design the state transition diagrams and data structures that are needed for each application.

- *Limited Interrupts*

An AO programmer's job is simplified by reducing the number of interrupts to an absolute minimum. A timer interrupt is used to keep track of the hardware timer in the system. The hardware timer granularity is defined at 250 microseconds. The hardware API allows reading this timer value. Network interface card transmit interrupts are acknowledged and their status can be read through hardware API. Keyboard interrupts capture the keyboard buffer data which can also be accessed through the hardware API for synchronous or asynchronous I/O. Other I/O operations such as mass storage (USB flash drive), mouse (USB), and other USB interfaces are managed without involving any interrupts.

- *Single Buffer Copying*

In OS-based computing, single buffer copying requires intricate mechanisms to bypass the operating system kernel and memory channels. In bare PC, a single buffer can be passed to other procedures within a process without any constraints. Appropriate circular lists and data buffers are designed to keep the buffer alive until it is no longer needed. As there is no kernel or memory management system, there is no need to copy buffers at various levels of process flow. A single buffer suffices for data copying and usage during its single thread of execution.

- *Limited Open Ports*

In an OS-based system, there are many ports open to communicate with other systems to support the open systems concept. In spite of today's emphasis on limiting open ports and increased OS-based security measures, there are more opportunities for users to unknowingly allow attackers to gain access and install Trojans and spywares on systems. In the simple bare PC environment, only the few (needed) ports are even implemented in a given AO so that it is impossible to open additional ports. For example, if a bare machine system is running a secure webmail server and a Web server, only ports 443 and 80 are implemented. In an OS-based system, it is possible to prevent the opening of other ports, but an attacker may subvert controls and gain access to a system; in a bare machine system, there is no unnecessary functionality implemented that an attacker can exploit since the entire code running in the machine is dedicated to serve only the needs of a single AO.

- *Loosely Coupled User Interfaces*

In today's computing, user interfaces and application programs are intertwined to work with the underlying operating system. This makes the software system dependent upon its platform and makes it susceptible to operating system changes. This also results in frequent porting to other platforms. In bare PC, applications are decoupled from the devices they run on and from their user interfaces. For example, an email client application program is essentially the same for a desktop, laptop, hand-held, or a cell phone although it may vary due to screen sizes, resolutions, and graphics capabilities. When user interfaces are decoupled from application programs, it enables them to run on a variety of devices with minimal changes and also results in simplicity and longevity. In

addition, the code size is also small thus making it useful in non-GUI applications with command line and simple lean menu interfaces.

- *Tightly Coupled Device Drivers*

In bare PC, OS-based device drivers do not work as they are dependent on system calls. Thus, there is a need for bare machine device drivers that do not use any OS-related calls. This poses a challenge to bare PC and its expansion in the future. However, for now we will use standard USB interfaces to address the diversified need for device drivers. The device drivers are tightly coupled with the application as the device driver API is part of an AO. This also makes the system more secure as the intruders do not have direct access to devices in the bare machine and the AO controls their access.

- *Self-Manage Exceptions/Errors*

All exceptions and errors are handled in the AO. The code for handling them is also coupled with the application program. This means there are no new errors or exceptions that can be processed by other software and provides tight control over the operational of an AO when it is running in the machine. It also requires a thorough understanding of exceptions and errors and the ability to add appropriate code to handle them.

- *Strict Memory Limits and Access Controls*

In bare PC, each process is given base and limit values for its memory and these are strictly checked in the hardware by Intel's IA32 architecture facilities. Use of code segments, data segments, and stack segments by application programs is thus strictly controlled. As there is no dynamic linking and loading of programs, it is impossible to alter memory limits and tamper with access controls.

II. LITERATURE REVIEW

A. Overview of Bare PC Approach

Operating System (OS-based) systems are based on some sort of centralized resource manager or controller to provide hardware abstractions to applications. The bare PC previously referred to as dispersed Operating Systems computing (DOSC) [28] proposes an extreme end of the spectrum in OS for building computer applications where there is no centralized resource manager or controller running in the machine. Applications [29] in Bare PC directly communicate with hardware (no need for abstraction layers) and the computer is made bare (no hard disk, no resident software, except BIOS). The bare PC approach is application centric and provides full control to the bare PC programmer. Bare PC applications referred to as application object (AO) [30] contain all the necessary application, data execution knowledge, and control. There is no need for the AO to depend on any other software or vendor specific products. The AO is built using a single programming language and works on a given CPU architecture base such as X86. These AOs can also be written to work with any other CPU architectures.

The strength of bare PC applications is derived from its simplicity, smaller code, design by obscurity, design for longevity, and inherent security. When the computing box is made bare, there is no need to secure the box anymore! The bare box can be used to run a variety of applications. The AO is self-contained and it belongs to an owner, which can be made secure at the AO level. As the AOs are application centric, it does not require all OS components as needed in today's OS. Only necessary hardware interfaces and controls are included in the AO thus making the AO small in size, simple in design and development. An AO may constitute a single application such as Webmail server or

it may consist of composite applications including: Webmail server, Web Browser and a Text-editor. Applications like the Web server [23, 24], Email Server [20, 21, and 22], VoIP [32, 33] and TLS on web server [14, 15] have being built on bare PC and these applications demonstrated significant improvements in performance compared to other commercial systems. These applications uncovered the unique features of the bare PC architecture and served as the bases for the design and implementation of the Bare PC Webmail Server.

B. Prior Work on Operating Systems

Operating systems (OS) have grown in size and complexity over the years resulting in ways to reduce or bypass OS to achieve better performance and simplicity in the system. Dr. Engler, in 1998 while working on his doctoral dissertation at the Massachusetts Institute of Technology, proposed an architecture the Exokernel[16, 17], which demonstrated that a minimization of OS functionality could be used to obtain core processes, or for development of applications to achieve core process functionality normally found within kernel system processes of the OS. Numerous studies such as Microkernel[18], OS-Kit[50], Bare-metal Linux[12], IO-Lite[41], Tiny OS[51] and other approaches have tried to reduce the interaction of OS or bypass OS to gain efficiency in the system. In [55], Linux is used to enable direct communication with the hardware by reducing the OS reliability. More recently, sandboxing techniques on the x86 systems [19] have extended OS kernels allowing applications to run guest plug-ins on the host OS.

C. Related Webmail Server Research

Atmail[4], MailTraq[36], Axigen[5], Afterlogic[2], Squirrelmail[46], Facemail[35], Adaptive Email[58], Petmail[44], Icewarp[26], Roundcube[45], Emailman[13], WinWebmail[57], Hexamail[25] are just a few of the numerous Webmail systems in existence today. Some of these systems are designed for high performance, while others such as Cisco's Webex[11] are designed for high reliability and availability. Email architecture to address problems associated with scalability and dependability due to conventional design approaches is proposed in [12, 48]. Techniques to improve performance of the Open Webmail system are discussed in [54]. In [42, 7], an email server architecture, which is based on a spam workload and optimized with respect to concurrency, I/O and IP address lookups, is shown to significantly improve performance and throughput. The design and implementation of an email pseudonym server providing anonymity to reduce server threats is capable of reducing risks due to OS-based vulnerabilities [37]. The notion of semantic email is discussed in [38]. Some performance and design details of Webmail server is described in [27] and Webmail client orientation is described in [40]. The security aspects of Webmail servers have been studied by many authors [39]. Webmail servers use HTTPS/TLS protocol to protect email messages in transit. However, all existing TLS-capable Webmail servers are OS based, and there is no TLS-capable Webmail server that runs on a bare PC. There are alternate approaches to email security. S/MIME [6] provides encryption, authentication, message integrity and non-repudiation for MIME messages exchanged between users (i.e., end-to-end). The design and implementation of a secure email system that provides encryption and signing, and additional features such as elimination of spam and

prevention of harmful attachments is described in [10]. The implementation of a secure Webmail system that uses CallerID for access is discussed in [52], and the specification of a Web-based system for secure transmission of email messages is given in [31].

III. INTRODUCTION

A. Webmail Server Background

Webmail is a web-based email service that allows users to access their email through a web browser instead of using desktop email clients (such as Microsoft Outlook, Pegasus Mail, Mozilla Thunderbird and Eudora) as shown in figure 2. It allows users to access their email account from any internet-connected computer anywhere unlike the application-based email system. In a conventional Webmail system, all the necessary protocols and intricacies are hidden behind the Web server providing the user a common Web interface. That is, a Webmail client has access to the Webmail server through HTTP, PHP, and other Web interfaces. The handling of email messages and relaying them to other servers is done by the Webmail server which is transparent to the client.

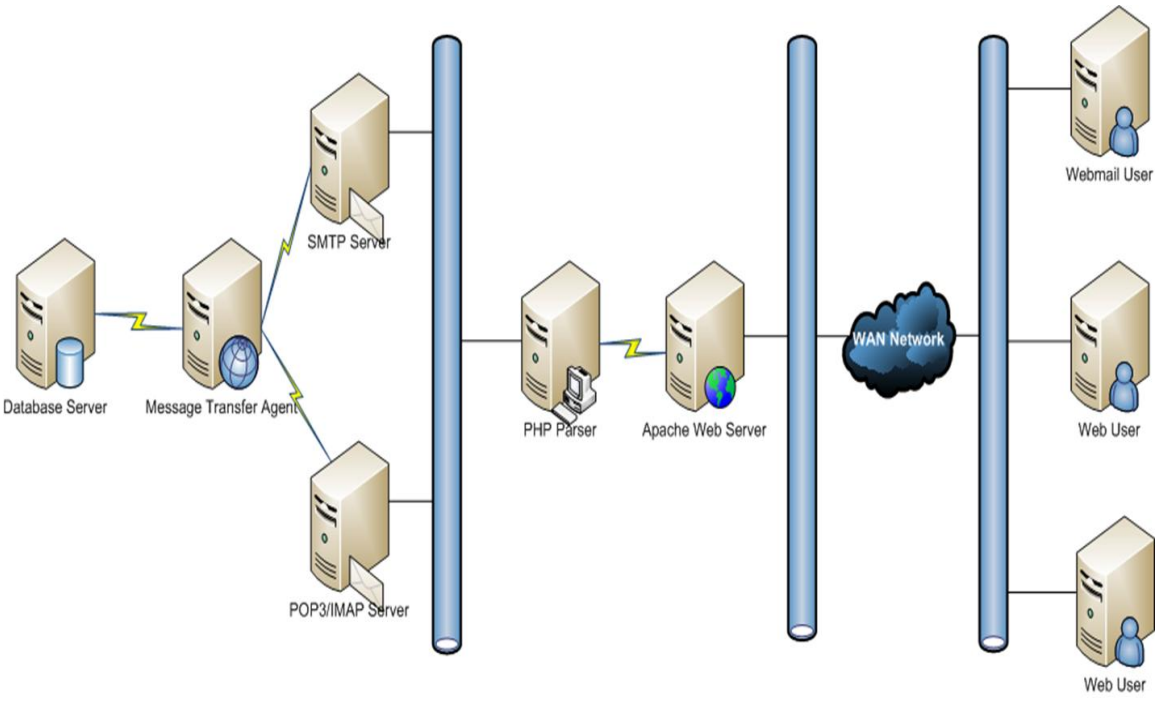

Figure 2. Conventional Webmail System

A conventional Webmail server may require a Web Server interfacing with an attached Email server or sometimes a single server performing both functions. A Webmail server poses additional complexities in handling client requests as the requests are intertwined with regular HTTP requests and mail services. A Webmail server can also use the TLS protocol [49] via HTTPS to provide security when sending and receiving email messages [1] making it more complex. When dynamic HTTP requests are involved, it also causes additional complexity in connecting the Webmail with a database server and PHP parser. The three components HTTP, PHP, and SMTP (email) static as well as dynamic introduces more complexity in the design of Webmail servers. For pervasive devices, a mini-browser running on an optimized stack is typically used to connect to a TLS-capable exchange server. When necessary, the browser may connect to a proxy server on the Internet to download compressed versions of Web pages to save bandwidth and reduce download time. In such environments, a Webmail server running a scaled-down version of TLS that retains its security strengths can be used as the exchange server.

B. Bare PC Webmail Server Approach

The bare PC webmail server encompasses all the email server functionality in a single server with great performance and ease of use without dealing with different servers for various types of applications as shown in figure 3. The design of this innovative server was as a result of experiences gained in developing the bare PC TLS-capable web server, and SMTP/POP3 servers. A bare PC Web server interfaces with any commercial client adhering to the client requests and their interfaces. It does not have any control on the client user interfaces. The Webmail server is user friendly to bare PC environment as the

functionality and complexity can be dealt with at the server level. This is because the Webmail server generates and serves all Web pages to the client thus allowing the Webmail server designer to control the design of the system.

The above characteristics of the bare PC Webmail server allow us to design and control the complexity of a Webmail server that outperforms other conventional Webmail servers. The bare PC webmail server architecture is based on threading techniques, delay/resume lists, and task stack mechanisms to provide efficient memory utilization and process control. It contains its own data execution knowledge and control, and does not require any other software support to run. Currently, the bare PC Webmail server run 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. The server is also not vulnerable to attacks targeting an underlying OS.

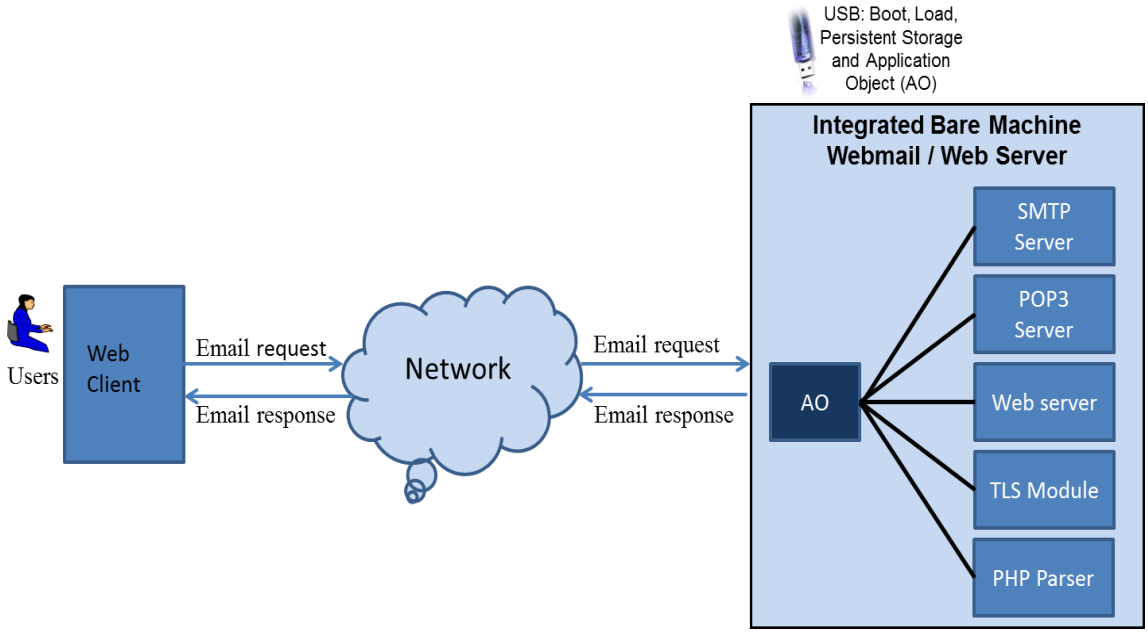

Figure 3. Bare PC Webmail Server

Bare PC applications are built to be secure since all underlying OS vulnerabilities are eliminated at design level. However this security is only possible at the system level, but once the packet or message leave the system and is within the network, the security of the message cannot be ensured unless a security component is added. To solve this problem a TLS protocol via HTTPS is added to provide security when sending and receiving email messages. Since a bare PC server application is self-supporting, it is unlike its OS-based counterpart that relies on services provided by the OS. For example, a bare PC server application contains lean versions of the necessary protocols, manages memory, schedules tasks on the CPU, and directly accesses the underlying hardware. Furthermore, the application layer and transport layer protocol code is intertwined within the code for the server application. There is no socket interface for applications in a bare server, and the intertwined parts of the code and the underlying task structure can differ from application to application. Protocol intertwining reduces inter-layer communication overhead compared to a conventional OS-based TCP/IP protocol stack, but complicates the design and implementation of the server.

Adding an auxiliary security protocol such as TLS as an independent module to a conventional OS-based stack is relatively straightforward. Once added, it can be used for many different OS-based server applications. Adding TLS capability to a bare PC server application can differ from server to server due to protocol intertwining and differences in the underlying task structure. Thus, although TLS has been added to a bare PC Web server application [15, 16], it is not possible to directly integrate this TLS code with the existing non-TLS Webmail server application due to intertwining. The secure webmail server minimizes both system and protocol overhead including that of TLS, TCP and

HTTP, and serves trimmed-down Web pages that are suited for browsers running on low-power client devices with small screen displays.

IV. ARCHITECTURE

The server architecture is supported by common general-purpose hardware including USB-based bootable devices, network interface cards, USB-based persistent storage, and Intel x86 based PCs. However, this server can be extended to run on other Intel processor architectures. The internal architectural design is based on threading techniques, delay/resume lists, and task stack mechanisms to provide efficient memory utilization and process control.

However, the architecture does not make use of any components of an OS or any other vendor software. Likewise, the compilation and execution of the bare PC application object for the Webmail server does not make use of any system or API libraries of the compiler which would require an OS. The bare PC approach only uses few BIOS function calls to invoke the timer, and reading device addresses from the Intel Hub Controller. Eventually, these BIOS calls and direct hardware interfaces can be driven to CPU to provide an AO execution. All other interrupts are managed by the AO as software interrupts or direct hardware interfaces.

Figure 4 is a detailed pictorial display of the architectural design of the bare PC Webmail server. For referencing purpose, numeric labels are assigned to items within the figure. The server is initiated on a bare machine by a boot program (**1**) which is read from a USB-bootable device. The initial sector contains a bootstrap loader which loads the menu program (**2**), which in turn loads the AO. The AO starts by initializing various data structures, parameters, tasks, and objects. Control is then passed to the Main Task (**3**).

The basic bare PC data structures used by the Main Task include a Delayed List (**8**), Resume List (**9**) and a Multiplexor mechanism (**4**) that switches between Received Tasks (**15**) and Resume List Tasks (HTTP: GET, POST) and selects a running task (**5**). This running task can be suspended (**6**) and returned to the Delayed List (**8**) when it is not being run. The Main Task checks for delayed tasks **(7**), whenever it is free and place them in the resume list. When a response arrives for a Delayed List task, a Resume () (**22**) brings the Delayed List task into the Resume List. A TCP Table (TCB, **13**) is used to store relevant information derived from the headers of TCP, IP and Ethernet messages (**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. These are a POST Object (**20**) to send emails from clients to the server, and an HTTP 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, one of these objects based on an HTTP GET or POST request 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 again. The active flag in the associated TCB entry along with associated data fields are then reset.

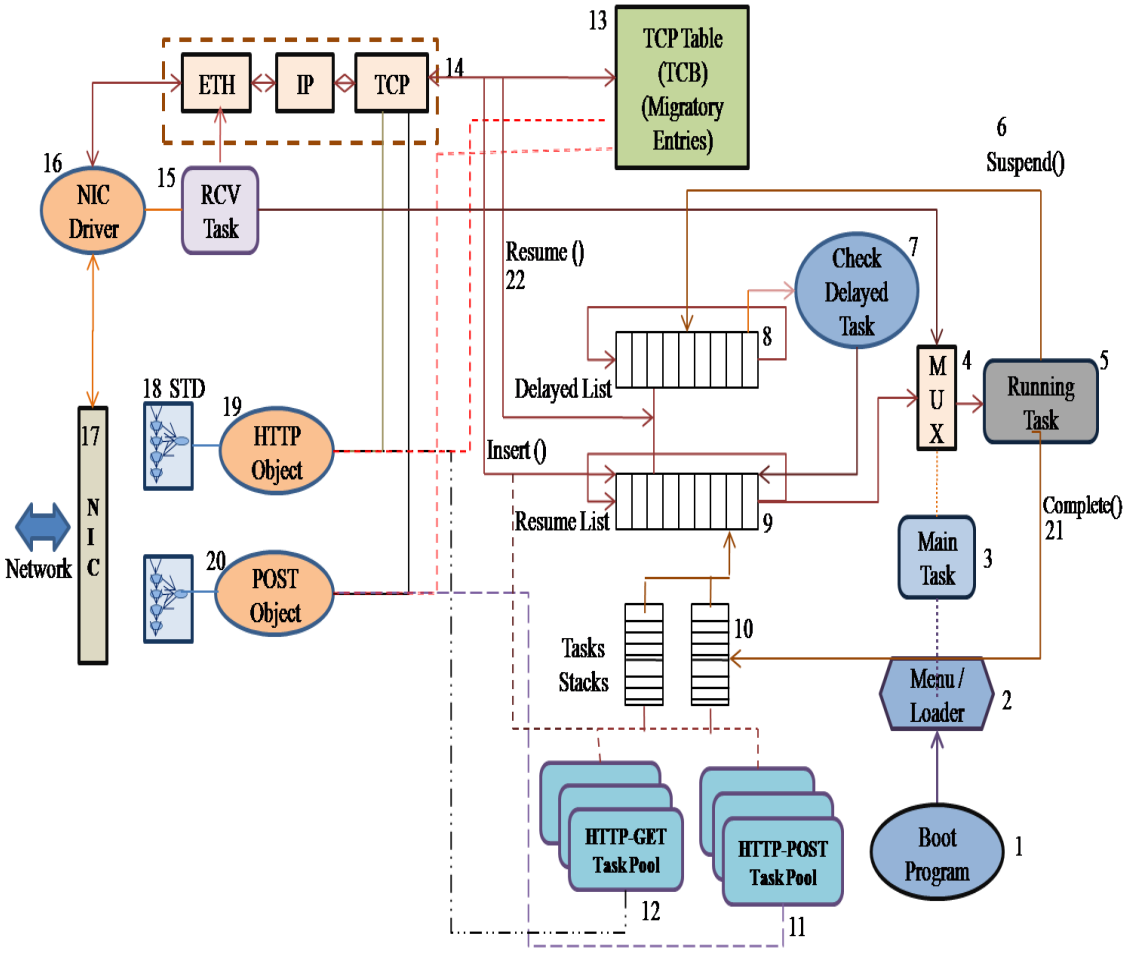

Figure 4. Architecture of the Bare PC Webmail Server

The bare PC architecture for Webmail server is novel and unique in many respects. The Webmail server only performs functions that are truly intended in its design. The scheduling mechanism is optimized for its function, and it can be claimed that it is an optimal solution for this application. 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 TCB. The HTTP GET or POST tasks do the same when it is ready to run. The suspension place and time is determined by the AO programmer at the program time and it is implemented in the code. The AO programmer will only suspend

a task if it is waiting for an event. When a task is complete, the same task will be reused without creating new tasks. All task pools are statically created by the AO programmer. The significant novelty of this server approach stems from the notion that the process execution knowledge and control are shifted from OS environment to AO programmer environment thus making the application self-controlled, self-executed, and self-managed. The system always runs a RCV task or an HTTP GET/POST task thus making the scheduler simple and its mechanism first come first serve. However, if the suspended delayed task expires in the Delayed List, it moves to the Resume List and follows the first-in-first-out (FIFO) priority. There is no need to prioritize any tasks in this model as all client requests will be given equal priority as they are ordered with respect to their arrivals. The Webmail server architecture is a "design for optimization" rather than enhancing performance after the design. This approach can be used throughout any bare PC applications, which are very difficult to achieve in a centralized OS based computing architecture.

V. DESIGN

A. Webmail Server Design

The design of the bare PC Webmail server was a very daunting and challenging task. The first task was to understand the functionality of the entire Webmail system and as such one Linux-based and one windows-based servers were installed and studied carefully. The design also avoids the use of any OS-based features and kernels, and creates an application, which will run on a bare PC.

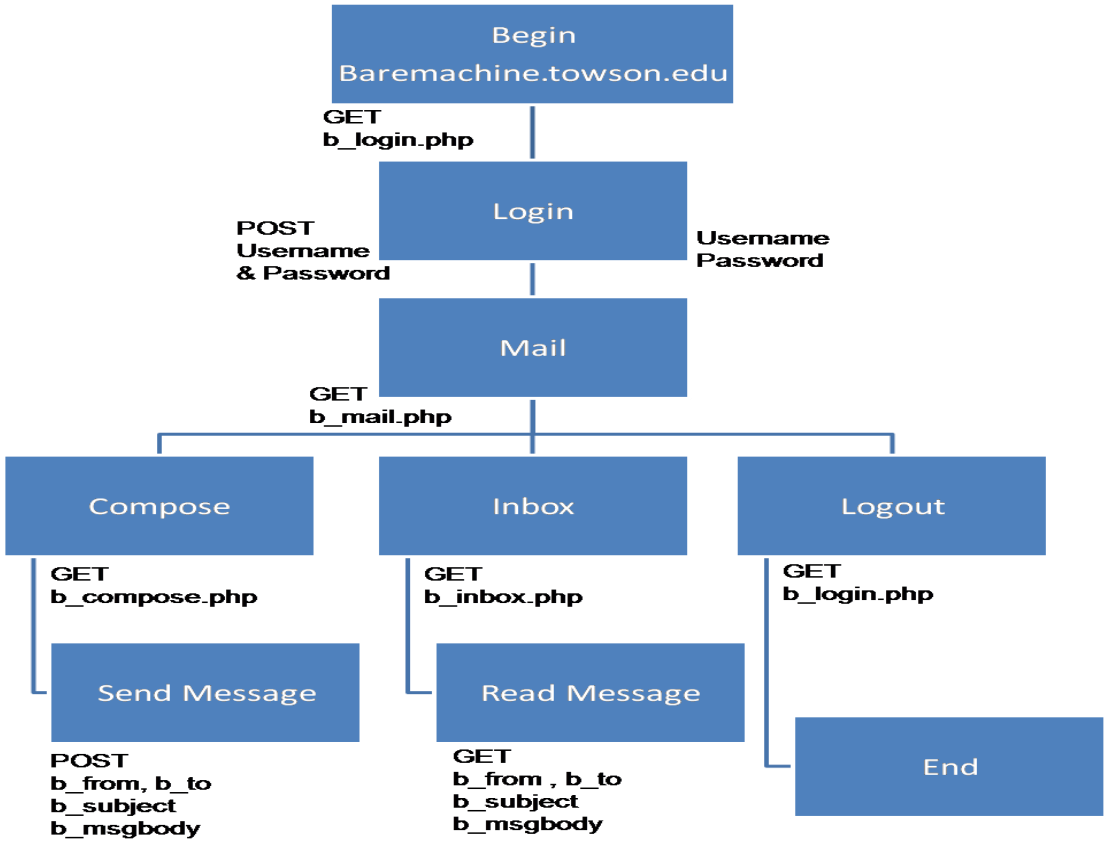

Figure 5. Client-Server Interface Design

The user interface for the bare PC Webmail server is text-based, which provides some configuration parameters to be specified at startup. The webmail server is an integration of a web server functionality, email server functionality, TLS functionality and 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. Figure 5 demonstrates the PHP client-server interface design for the Webmail server. For effective interaction with the secure server, a user first requests a Login page and log into the server. After the Login page, a mail page is displayed, where the user can either compose and send a new email or check inbox and read message or logout of the server. During the design of the bare PC Webmail server, each GET and POST command was implemented using State Transition Diagram Modeling (item 18 in Figure 4).

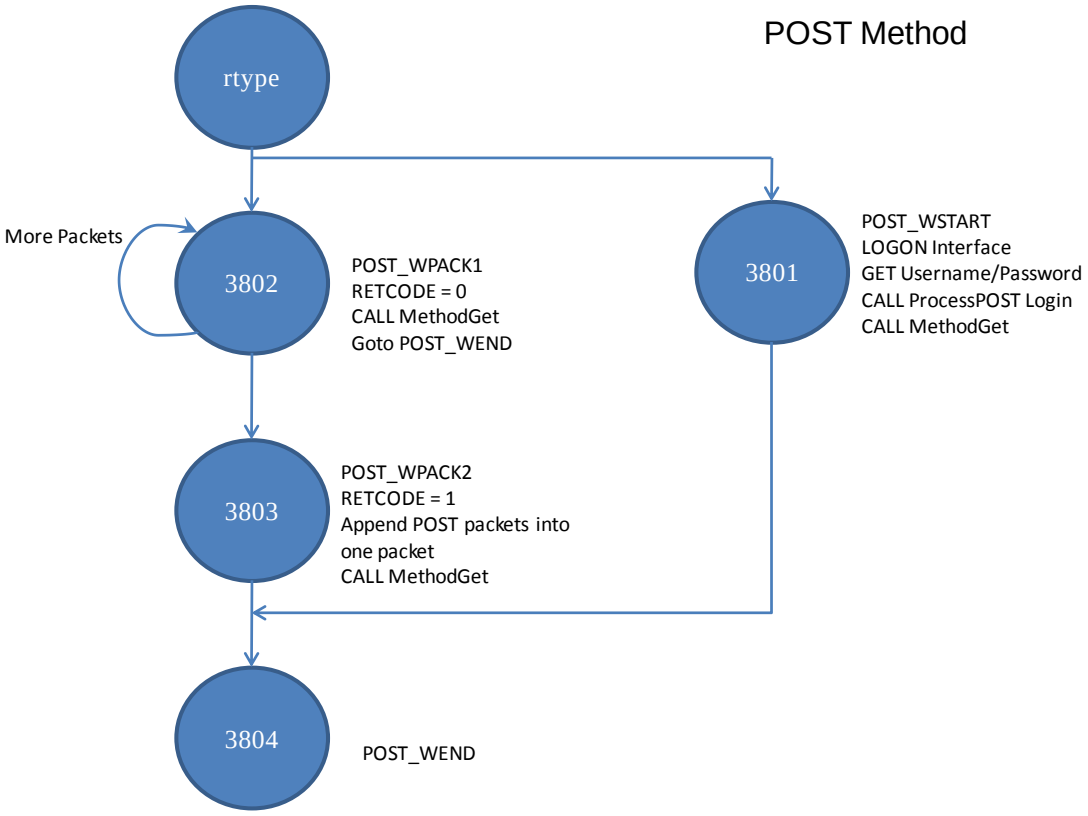

Figure 6. State Transition for Processing POST Request

The post method modeled in the state transition diagram in figure 6 is used for processing POST request for user login authentication request and email message data. The rtype state indicates the request type which identifies whether the request is a user login request or an incoming compose request. The application programmer makes a decision after retrieving the request type and directs the program to the appropriate state to process the received request. Each state was modeled to indicate HTTP and TCP message protocol transitions, modeling the client and server interactions. The client sends TCP SYN and the server responds by sending the TCP SYN-ACK. The client then sends a TCP ACK and either an HTTP GET or POST request to the server as shown in Figures 7 and 8. Figure 7 is the HTTP GET interactions. These interactions can be classified into three categories: TCP Connection Establishment, Data Transfer, and TCP Connection Closing. The data flows from the server to client. Notice that the client application and the TCP protocols are intertwined in this diagram indicating the actual implementation in Bare PC. Figure 8 also illustrates the internal client server interactions to perform HTTP POST requests from a client. This is similar to a GET command; however, in the POST command, the data flows from client to the server. Once, the GET or POST gets to the server, the lean PHP-parser built into the Webmail server parses the necessary parameters to authenticate the user as shown in figure 9. Composed emails are also sent to server for parsing to extract the fields in the data received and relay the data contents to the email component of the server to be stored as an email. 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.

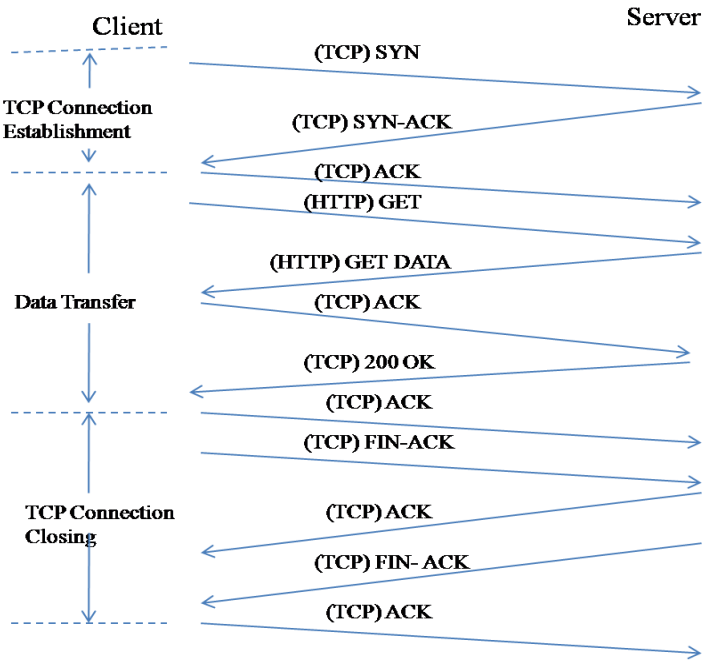

Figure 7. GET Client/Server Interactions

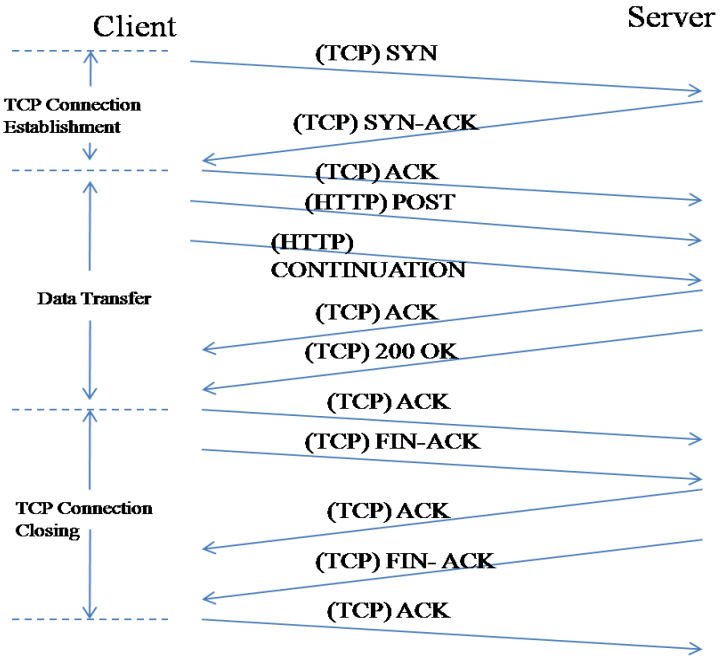

Figure 8. POST Client/Server Interactions

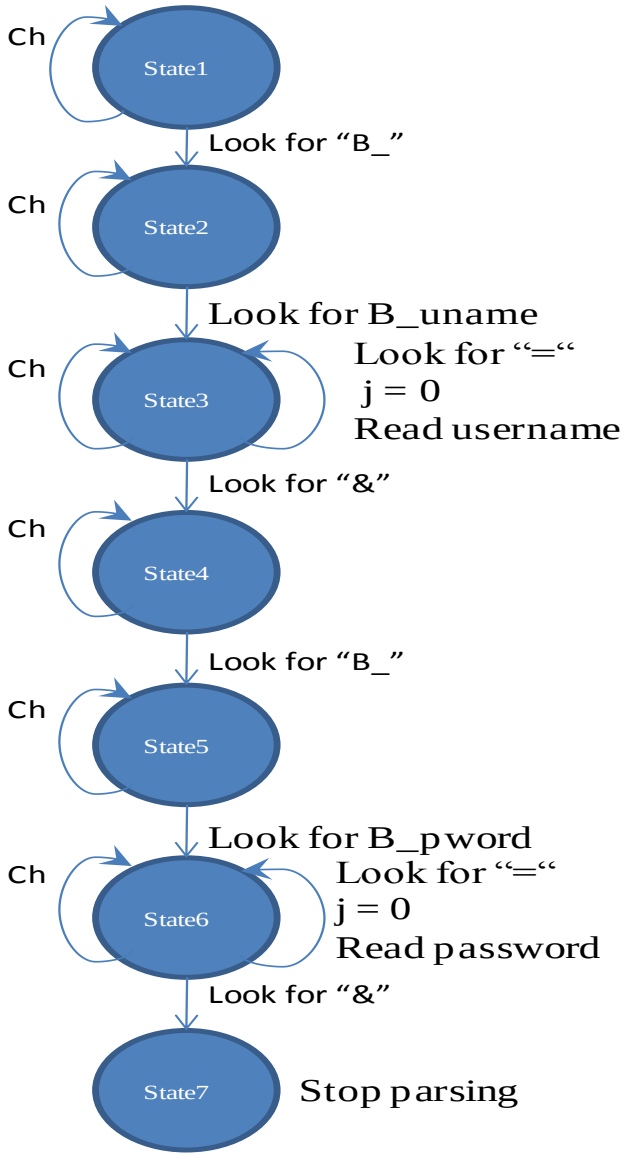

Figure 9. State Transition for Parsing Username and Password

Unique field names in the form input that represent the FROM, TO, SUBJECT, ATTACHMENT and MSGBODY as were specified in the PHP/HTML script files to aid in parsing the received data to extract the appropriate commands for email storage and retrieval as shown in figure 10. After the data have being processed, the client and server send a TCP ACK, TCP FINACK, TCP ACK interaction to indicate the closure of the TCP connection. To ensure efficient and faster execution of Webmail request, certain

pointers (phpparseflagAread, phpparseflagread, phpparseflag) were kept in memory to force the PHP-parser to parse certain parameters only once while other parameters are parsed multiple times.

These pointers are specified for b_inbox.php, b_attach.php and b_readmsg.php files to allow the parser to parse them once and dynamically modify the files to insert the messages before delivering them to the client. Email storage and retrieval are based on special PHP Session keys assigned automatically to each user at login and dynamically stored in a session table as shown in figure 11 using the User ID, Session Flag and Session String generated using random generators. Emails are stored in dynamically in memory using persistent storage mechanisms to maintain the integrity of all the data stored. Figures 12-14 illustrate the storage structures for email storage and handling. Figure 12 is the permanent storage structure for each user based on the username, the available address location and message number. Figure 13 is the temporary message storage location for recently received emails. After all processing and parsing are done the email message is moved from this location to the permanent storage location. A number of email storage allocations are assigned to each user for email storage. Emails for users are stored using the message number to represent the message index as shown in figure 14.

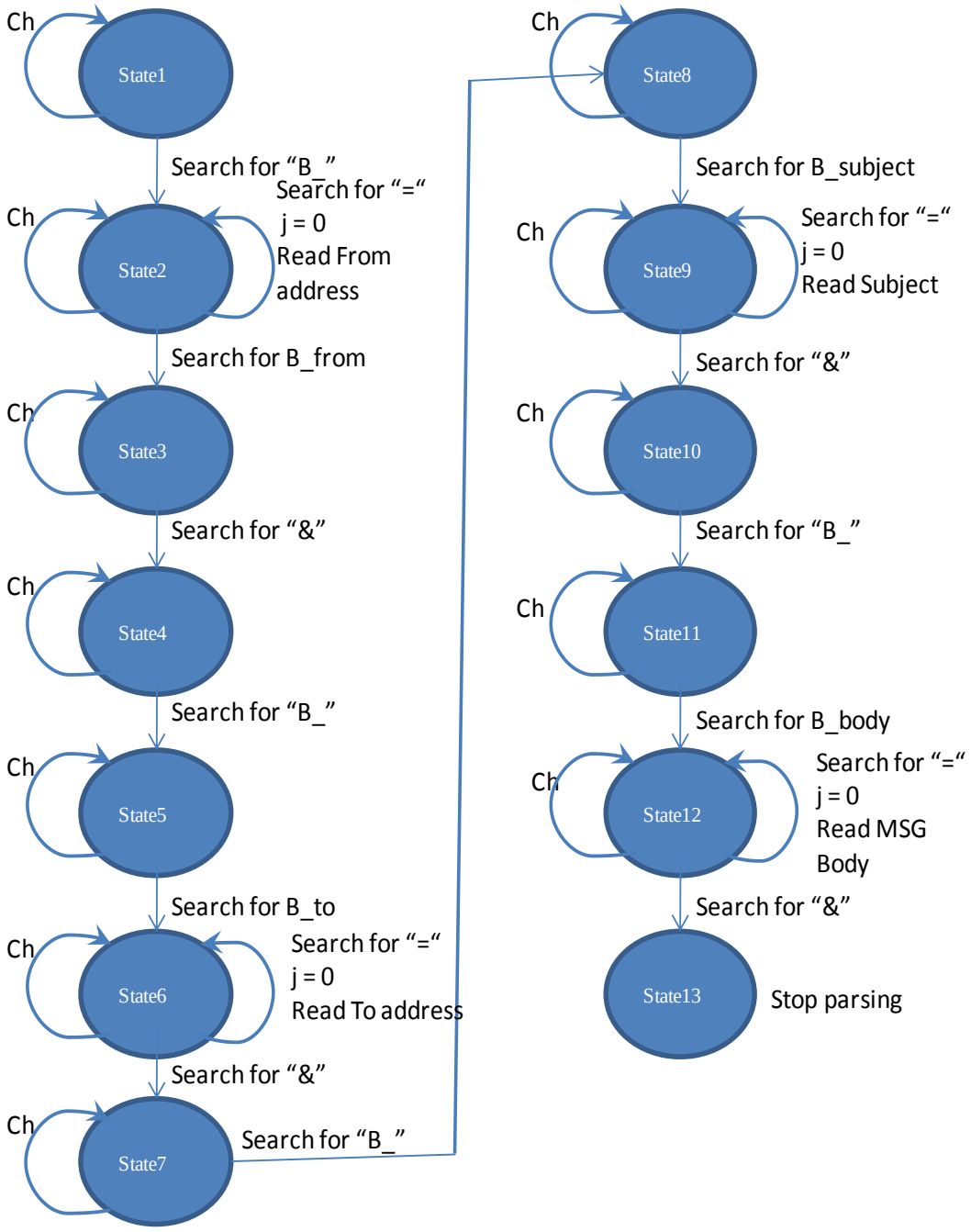

Figure 10. State Transition for Parsing Compose Message

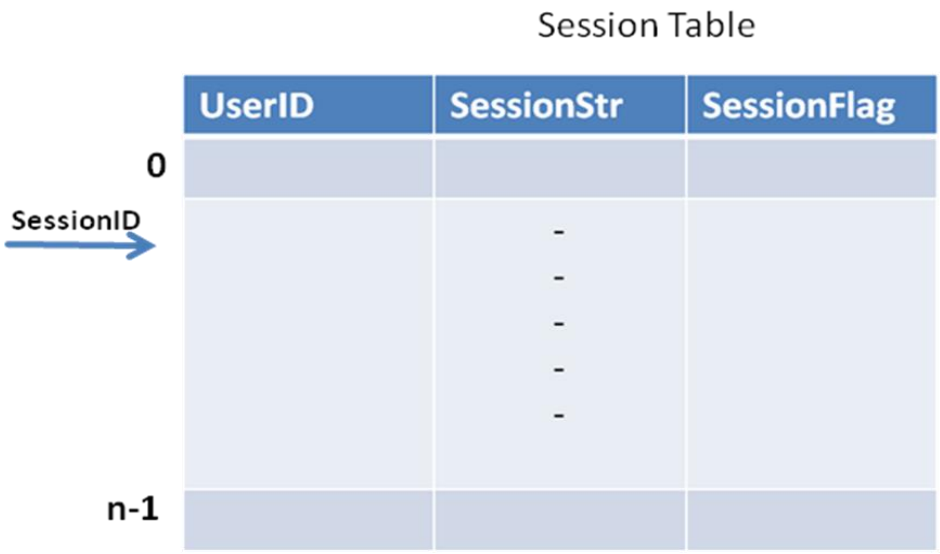

Figure 11. Session List Table

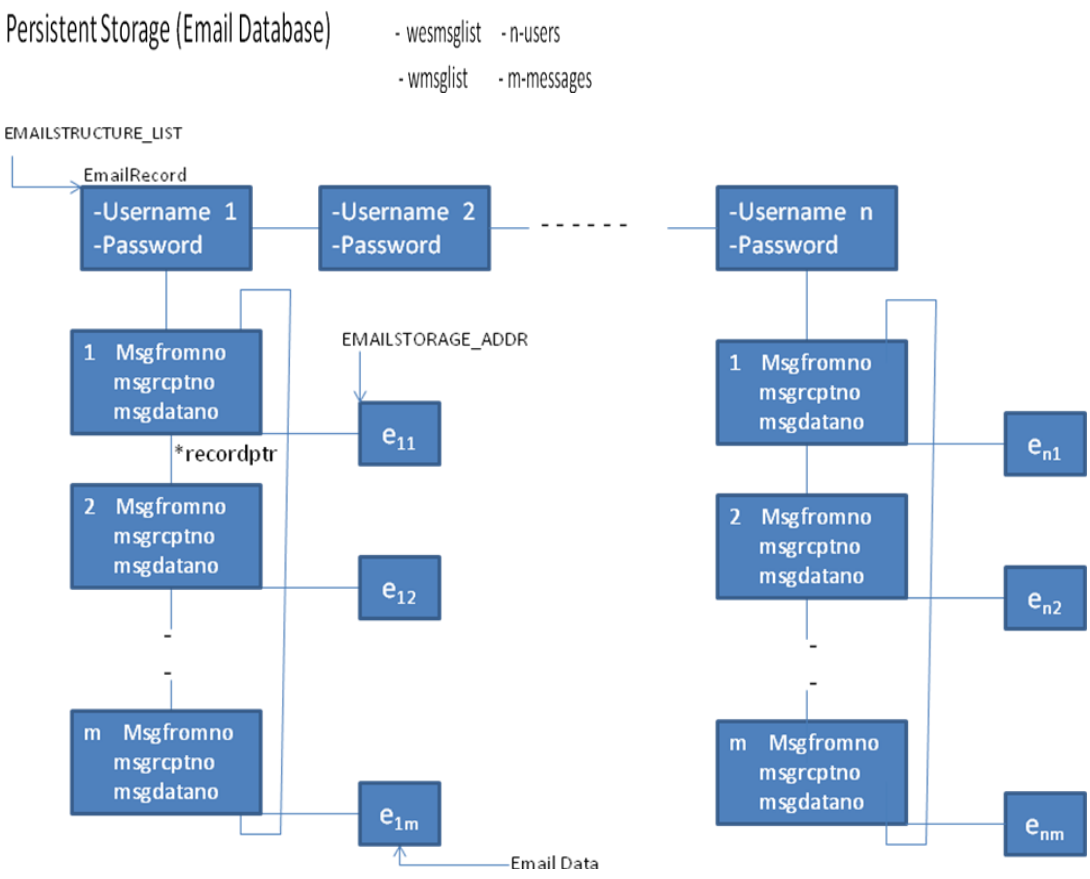

Figure 12. Data Storage Structure

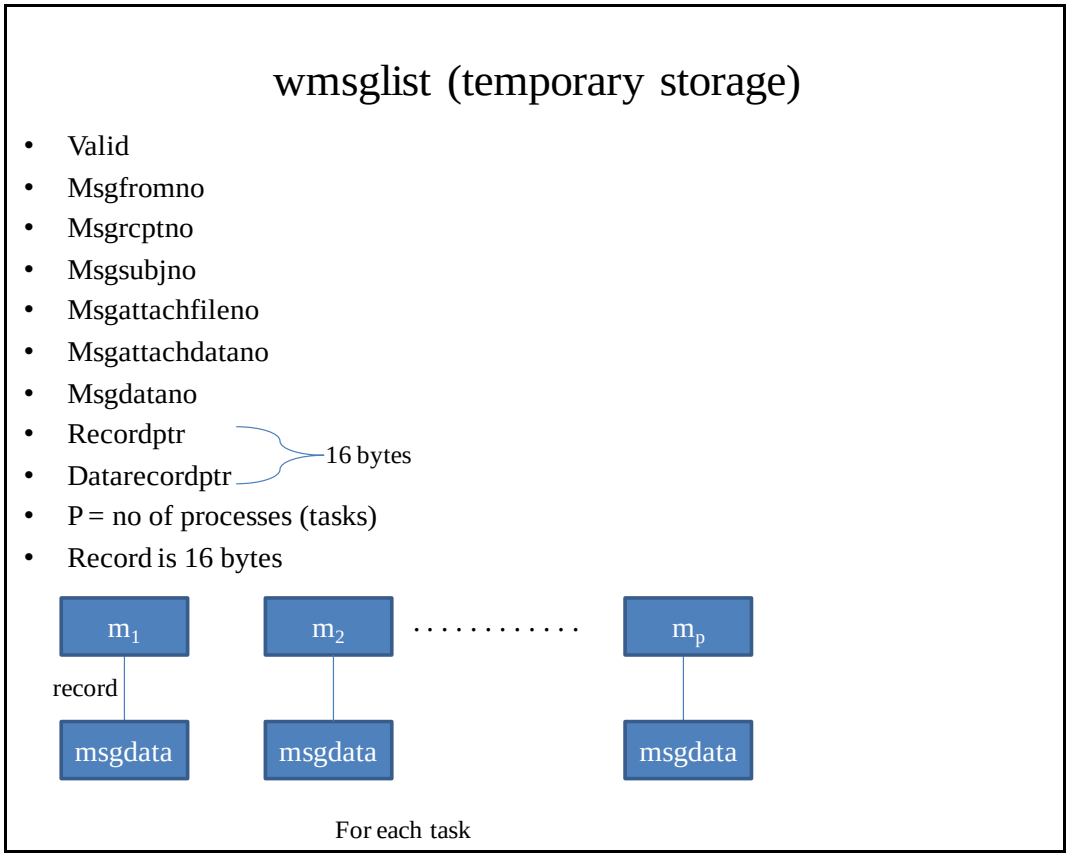

Figure 13. Temporary Data Storage Structure

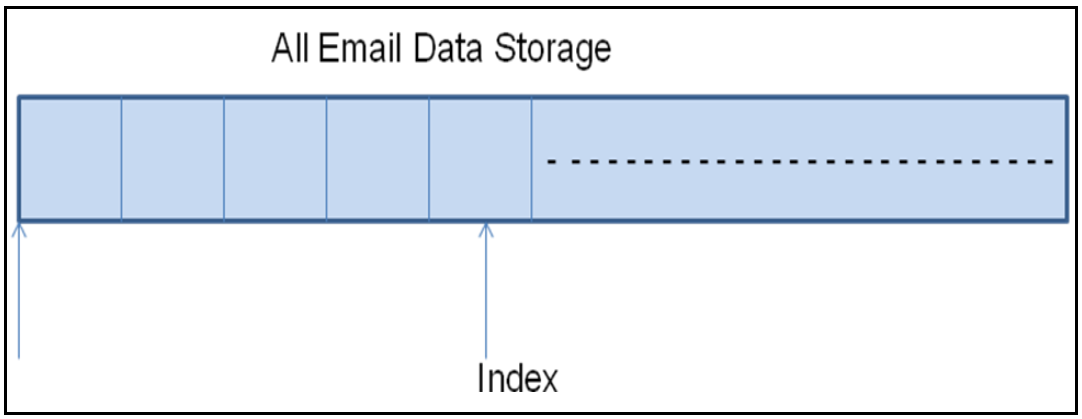

Figure 14. Email Data Storage

Email message list table is used to insert email messages into the permanent data structure. Message insertion to the table is based on the current task, number of email users, user information and user storage availability. This structure ensure that messages

do not overlap in memory as there is not file management system available for the current bare PC design. Figure 15 shows the message list table for inserting email messages.

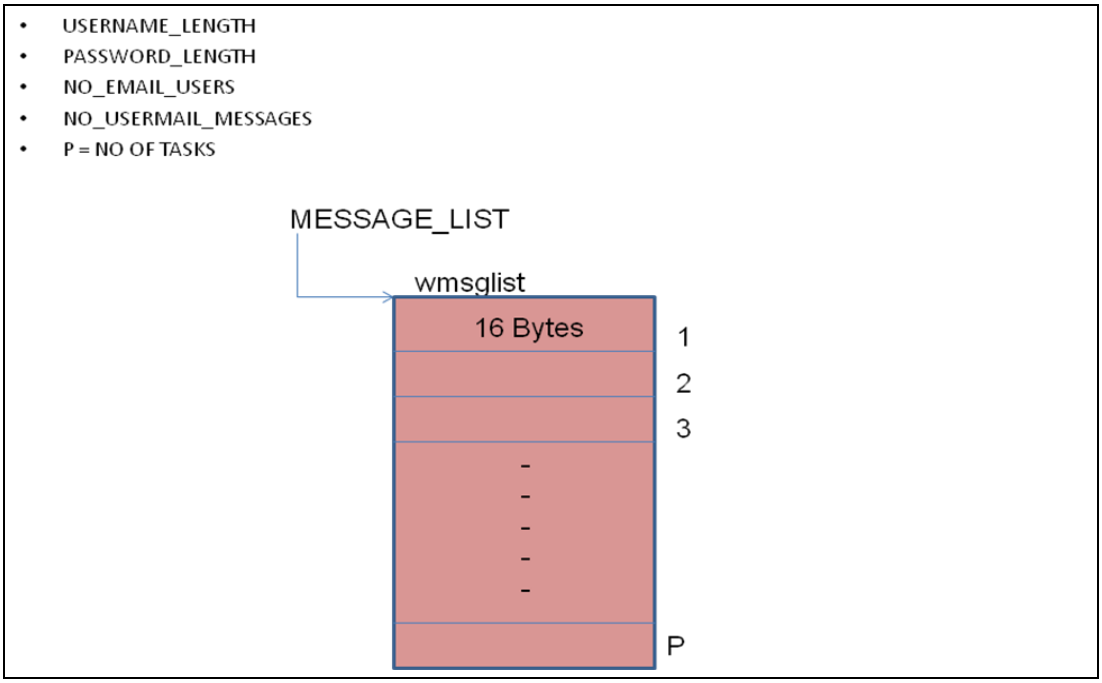

Figure 15. Inserting Messages into Message List

After the appropriate session key is assigned to the user, all subsequent requests by the user are done using the same session key. To request an inbox message from the email storage area, the PHP Session is parsed when the inbox request arrives. Parsing the PHP Session enables the system to detect the appropriate User ID, so that the user messages are retrieved. If another user does a Login while a user is still logged into the server, a different unique PHP Session key is assigned to that user. This ensures that concurrent connections can be handled by the server and each client-server interaction is unique to the particular user. The server was also designed to handle large messages by extracting the content-length and matching it to the total bytes of messages received for multiple POST data continuation command. For efficient memory utilization while

storing emails (small and large), data descriptors are specified in the code for dynamic memory allocation to allow email storage. The application programmer makes this decision at program initialization to ensure that the appropriate memory needed by each user is assigned correctly so as to prevent stack and buffer overflow. The data descriptors used in the webmail design are shown in figures 16 and 17.

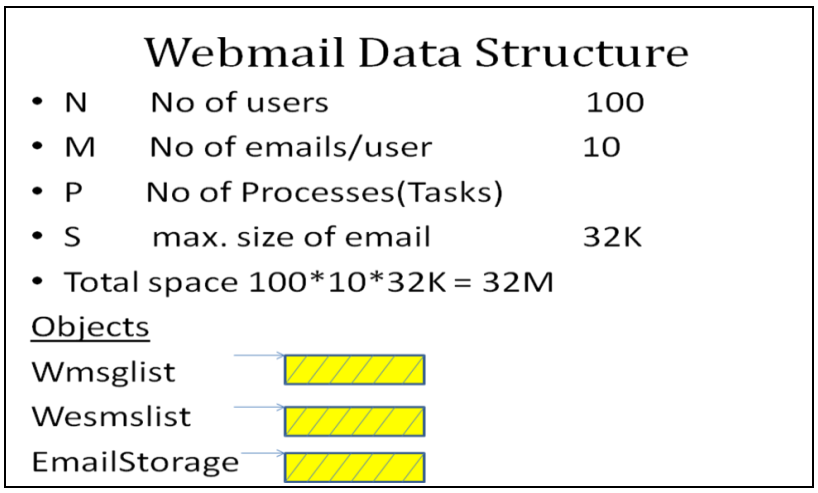

Figure 16. Data Structure Descriptors

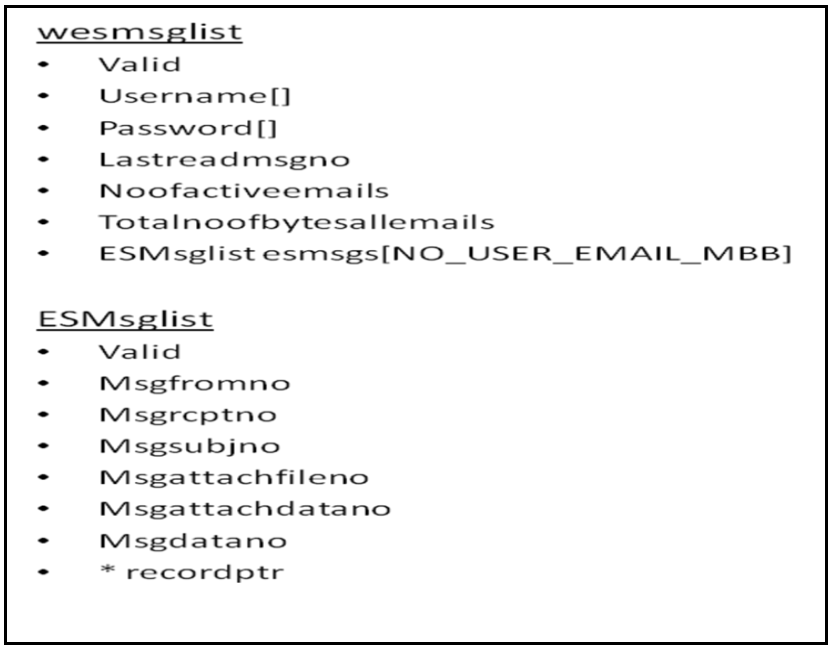

Figure 17. Data Descriptors for Wesmsglist Table

B. Webmail server with TLS

The webmail server was made secure by adapting the TLS module in the bare PC Web server [3] and integrating it with the code for the non-TLS Webmail server. Figure 18 compares the respective client/server message exchanges for non-TLS and secure Webmail servers with protocol intertwining. In both cases, a TCP handshake is done as usual for connection establishment. In the case of the non-TLS Web server, the client next sends HTTP GET and POST commands to be processed. A TLS capable server requires an additional TLS handshake for negotiating security parameters and setting up a master key. The TLS module is responsible for this handshake and for encryption/decryption of subsequent messages including the HTTP GET and POST commands. Processing on a secure server involves several phases: the TCP handshake phase; the TLS handshake phase consisting of client hello, server hello, certificate, server hello done, client key exchange, change cipher spec, and finished (encrypted handshake) messages; the data phase during which encrypted and authenticated HTTP data and alert (TLS close-notify) messages are sent; and the TCP Connection closing phase.

Design characteristics of the TLS-capable bare PC Webmail server are discussed next. A key aspect of the design is the use of protocol intertwining. As shown in Figure 19, protocol intertwining enables requests to be processed efficiently during the above-mentioned phases of the TLS handshake. In essence, the TCP, TLS and HTTP protocols are intertwined within the Webmail server application.

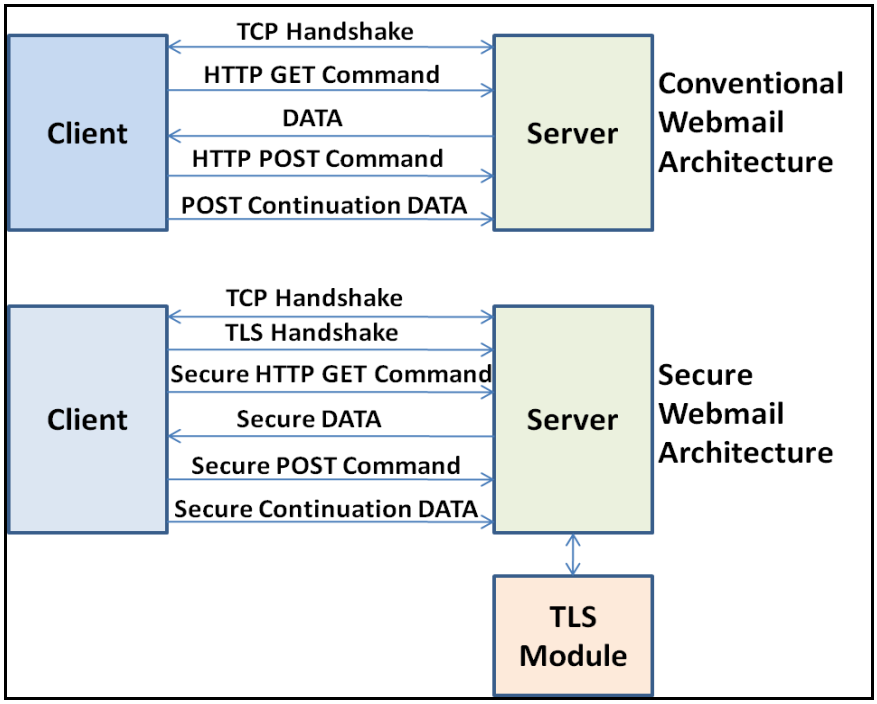

Figure 18. Non-TLS/TLS Webmail Messages

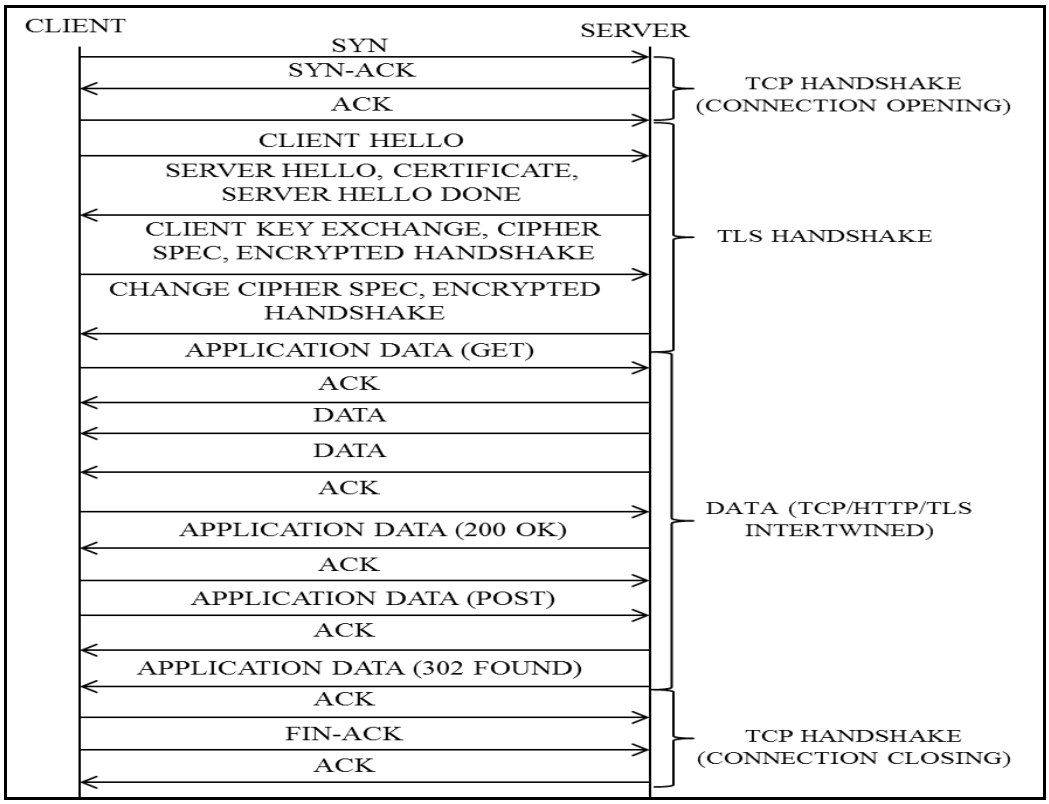

Figure 19. TLS/TCP/HTTP Protocol Intertwining

The tasks and task scheduler are also an integral part of bare PC server design. All bare PC systems have a Main task that runs whenever other tasks are not running, and a RCV task that handles incoming packets. For efficiency, the secure bare PC Webmail server also has a separate TLS task that handles TLS processing as well as the HTTP POST and GET processing. A TLS task is created for each secure TCP connection made with the server. This per-connection TLS task replaces the HTTP GET/POST tasks in the non-TLS bare PC Webmail server, and the code for those tasks is now incorporated into methods (within the GET/POST objects) invoked by the TLS task. These methods in turn directly call the TLS encrypt/decrypt methods when processing requests or responses. A state transition diagram is used to model the event/action behavior of the server. Use of a single TLS task per connection also simplifies processing in case HTTP GET/POST commands come in over a long period of time when the HTTP KEEP_ALIVE option is used. The application manages and schedules the TLS tasks enabling the server to process requests concurrently. Each message request and its state information are stored in a (state) Transition Control Block (TCB) table. In addition, parameters used by re-entrant code are also stored in this table. In a bare PC Webmail server, the simultaneous sharing of resources is avoided by allocating resources independently for each request and maintaining the necessary state information in the TCB table. The TCB information is also used in task scheduling.

Unlike task scheduling on a conventional system that is controlled by the OS, bare PC task scheduling is controlled by the programmer. At development time, the programmer designs a task schedule, where a given task executes as a single thread of execution, thus eliminating the need for a centralized task scheduler.

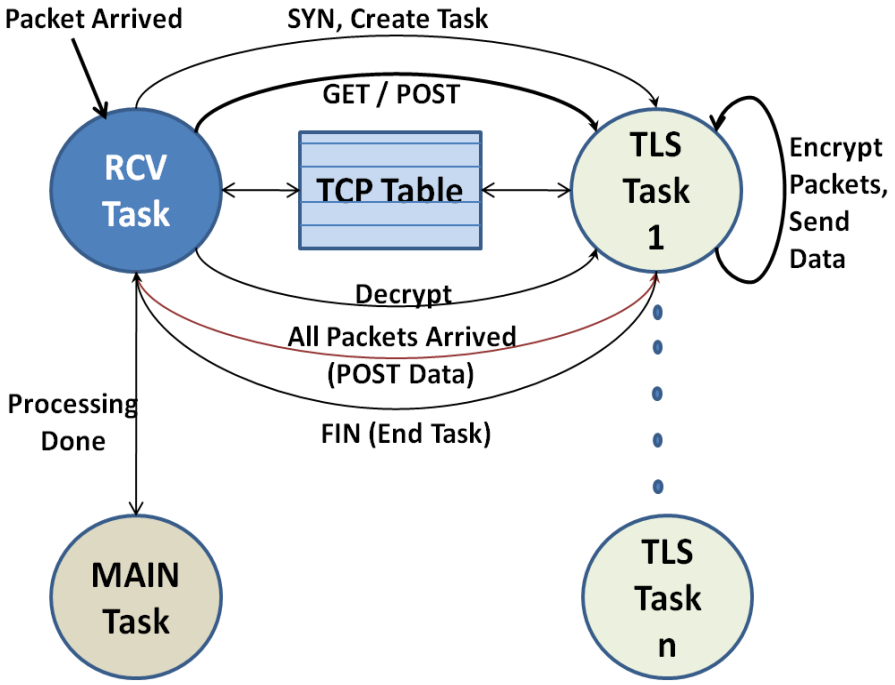

Figure 20. Task Interactions

To ensure efficient CPU utilization on the server, whenever a TLS or HTTP task needs to wait for an event, it is suspended until the relevant event occurs and the task can resume execution. A TLS task is initially created when the TCP SYN segment is received. After the TLS handshake is complete, client GET/POST requests and associated data are decrypted or encrypted under the control of the RCV and TLS tasks. These events and the associated task interactions are shown in Figure 20.

In the non-TLS Webmail server, client GET/POST commands in the clear are processed by the intertwined TCP and HTTP protocols. The server uses a lean PHP parser to parse all the PHP scripts and PHP tags embedded in the bare Webmail user-interface templates. For a POST command, the CONTENT_LENGTH in the HTTP header tells the size of the HTTP POST data sent by the client, which may arrive in several TCP segments.

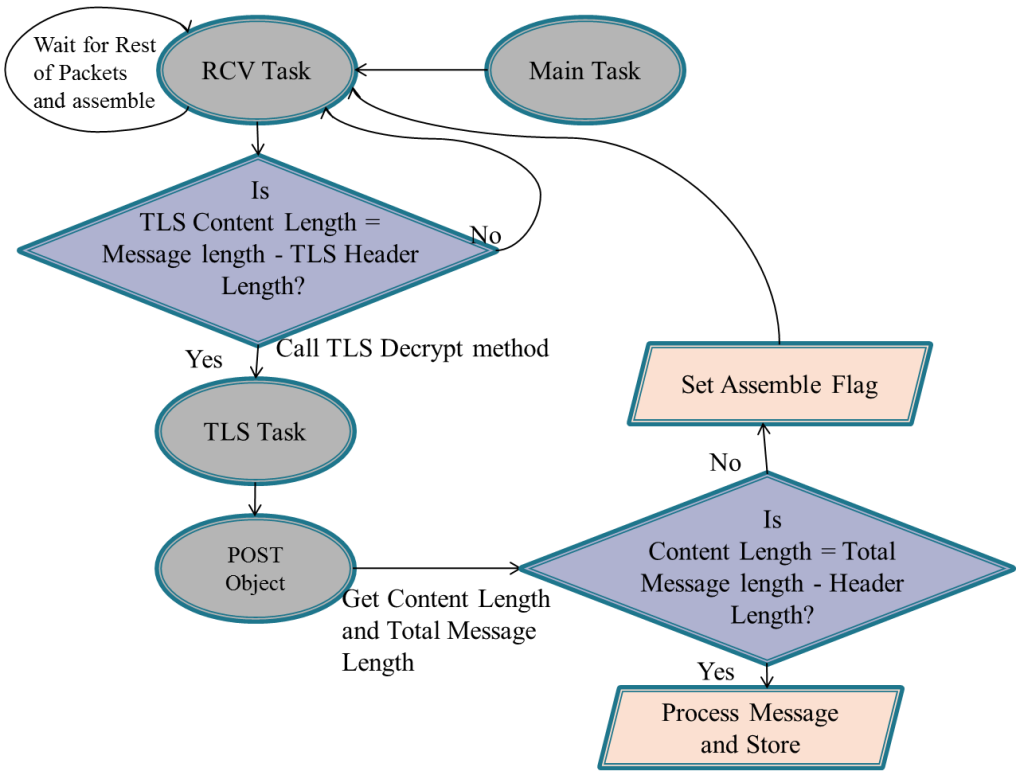

Figure 21. Handling TLS and HTTP Fragments

In case of the TLS-capable server, as shown in figure 21, GET/POST commands are contained in encrypted TLS messages. For a POST command, the encrypted HTTP POST data arrives in one or more TLS fragments. The TLS CONTENT_LENGTH is used to check whether all fragments have arrived so that decryption can be performed. After decryption, the HTTP content length in the HTTP header is used to determine if all the HTTP data has arrived. If the HTTP data is fragmented, the assemble flag is set and the server waits for the remaining packets.

C. Novel Design Characteristics

Bare PC webmail server features some key design characteristics that can be leveraged when building high-performance systems. These characteristics are difficult to replicate in a conventional OS-based application. Some of the unique design

characteristics as discussed in figure 4 which illustrate performance improvements for Bare PC webmail servers are further discussed in the following subsections.

- *Protocols*

Bare PC uses intertwining of the necessary (HTTP, TLS, TCP) protocols during the design of application-specific servers and systems. Protocol 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.

- *Concurrent Requests*

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

- *Concurrency Control*

Each message request and its state information are stored in a (state) Transition Control Block (TCB) table, in addition to the parameters used by the reentrant code. In a Bare PC Webmail 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. The TCB information is also used in task scheduling.

- *Task Scheduling*

Unlike OS-driven task scheduling, the bare PC 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, the program is suspended until the relevant event arrives allowing the task to resume execution. This simple task scheduling strategy has also been useful in building high-performance bare PC Web servers.

- *Client Interactions*

OS-based Webmail server designs are focused on application layer request-response interactions with clients. In the bare PC computing approach, the request-response interactions and behaviors are modeled based upon state transition diagrams. This avoids the overhead associated with OS-based servers due to their use of API libraries and a strictly layered approach. This results in a higher throughput, and a better ability to process more emails in a given time.

- *System Operation*

Conventional applications require OS-based kernel utilities and programs to start the system, load the application, and manage resources. In the bare PC approach, the AO is a self-contained, monolithic executable code capable of bootstrap loading and providing resource management directly within the application. This means that an OS is not needed for the user to boot and execute the application from USB flash memory or similar storage media.

- *Volatile Memory and Persistent Storage*

Conventional Webmail 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 PC approach, it frees up the use of physical memory for specific

applications. Persistent storage in a bare PC computer is provided by USB flash memory, which can serve as a mass storage.

- *Input / Output*

OS-based applications make system calls or use API for I/O, but bare PC application's 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 bare PC applications, which invoke them via in-line calls resulting in increased I/O performance.

D. Design Issues

During the design of the secure bare PC Webmail server, many design issues were encountered and had to be resolved in order to ensure a robust and efficient system. OS-based systems are less susceptible to these design issues since there are already existing API structures and Parser functionalities.

- *File system*

The bare PC Webmail server does not use Inbox Folders for IMAP. This is because the current bare PC architecture file system is under research. Email storage is done in memory using memory address allocation.

- *Large Emails*

Keyword search enables the parser to parse and capture the content-length of the message received. The value of the content-length is compared to the total size of bytes of packets received. This ensured that both small and large messages could be handled by the server and stored or retrieved correctly.

- *PHP Parser*

Bare PC is a novel system and as such several drivers and components had to be developed. During the design of the bare PC Webmail server, there was the need for a PHP parser that could parse the PHP scripts. So a lean PHP parser was designed to parse and capture the email parameters for storage and retrieval.

- *User interface design*

Flexible dynamic user interfaces result in increased server complexity. The user interface in the Webmail server is simplified by designing static PHP pages that can be pre-parsed and indexed for efficient processing without incurring any parsing delays.

- *Web Browser Compatibility*

The bare PC Webmail server was design for functionality and efficiency, therefore during design the application programmer ensures that Web pages could be delivered by any known browser. The bare PC Webmail server was tested with several browsers to ensure that compatibility issues were resolved.

- *Robustness*

The bare PC Webmail server design is very robust. Task scheduling, sending and receiving email messages, concurrency and synchronization, network protocols, and API to bare machine are very flexible and easy to maintain. It also provides more robustness for designing new applications.

- *Concurrent Email Handling*

Session keys were specified for each user to enable the server handle concurrent email request. These session keys were randomly generated using a pseudo random generator algorithm at user login time. Once the session is opened and a key generated

for the user, the user holds the session until he logs off. This ensures that two or more users can access their emails concurrently without interference.

- *Email Storage/Retrieval*

In conventional Webmail servers, the HTML and PHP files are stored on a Web server while the email messages are stored on a Database server. The bare PC Webmail server runs on a bare PC and it are not convenient to carry all the HTTP and PHP files and the email messages in the AO. HTTP and PHP files are transferred into memory during server initialization. Email messages are also stored in memory and are fetched from memory when the request comes in. Currently, all files are fetched into memory as there is plenty of memory to support file storage. This characteristic helps server's mobility when moved from one location to another on the Internet.

- *Throughput*

In the driver design, interrupt service routines (ISR) are used to check the status of transmitted message to make sure the packet is sent successfully. Using ISRs to process the status also improves the performance of the server. However, that make the system more complex and ISRs will interfere with task processing. The HTML and PHP file size will also play a major role in the throughput outcome, which needs to be studied further

- *Response Time*

Interrupt service routine (ISR) implementation for receiving and sending messages to NIC will improve response time. However, this approach needs to be studied in combination of ISRs and Tasks. ISRs make the system more complex and hard to debug in a bare PC environment.

- *Task Scheduling*

The bare PC Webmail server tasks are provided as C++ API calls. Each task is a member function in a task object. Many tasks are designed as needed by simply adding new member functions in the task class, and control the task scheduling. Currently a simple round robin algorithm is used to schedule tasks. Tasks are scheduled when a hardware timer interrupt comes at a periodic rate of 55 milliseconds. The clock rate can be varied by dividing the rate in multiples of 2.

- *Memory Address Allocation*

All email messages, HTML and PHP files reside in memory. It is therefore prudent to manage memory allocation more efficiently to avoid email messages being overridden by other messages and files. During initialization, memory address allocations are assigned to each AO. Inside the AO further memory assignments are done to avoid buffer overflow and ensure that messages are stored correctly at the right address. This memory management also helps with message retrieval by using the correct memory address. When bare PC Webmail server is made secure then application implementation will include implementations of the AES, HMAC, SHA-1, MD5, and RSA algorithms. Memory required by these algorithms would typically be dynamically allocated on conventional systems. However, bare PC systems do not support dynamic memory allocation. Instead, all memory is pre-allocated at compile-time to avoid implementing complex memory management functions and incurring memory allocation overhead.

- *Spam Control*

Bare PC Webmail server is designed to be very secure and robust. The code size is very small and in control of the application programmer. Therefore there is limited

possibility for external security treats to bare PC and its application. However the bare PC Webmail server does not have spam control mechanism in the current design. It however has a TLS support to ensure encryption and decryption of email messages. Further research on spam control and Web filters need to be conducted to allow the bare PC Webmail server to block spam and other web treats.

- *MIME Encoded Attachments*

Bare PC Webmail server supports text only attachments. Due to the lean nature of Bare PC, certain file types are currently not supported by the bare PC Webmail server. Further studies need to be conducted to enhance the capabilities of bare PC to support any application and all file types.

- *Certificate generation*

The RSA certificates and keys are pre-generated outside the server (on a Linux machine) and stored in a file. An adaptation of trivial FTP is used to convert this data from ASCII into Hex format and transferred into the server's memory. There is support for RSA key, AES/CBC cipher and SHA MAC for this lean secure server.

VI. IMPLEMENTATION

A. Bare PC Implementation

The bare PC Webmail server implementation was done using a standard MS-Windows environment, Visual C++, and the MASM Assembler for software development. However, this approach does not include any OS-related libraries and system calls. Instead, the AO uses direct hardware interfaces designed for bare PC computing. Most of the direct hardware interfaces are implemented in C/assembly language using software interrupts. The size of this assembly code is approximately 1,800 lines. These direct hardware interfaces include: display, keyboard, timers, task management, and real/protected mode switching. The 3COM 905CX NIC driver code is approximately 1,400 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 implementation of the Webmail server architecture depicted in Figure 3 is written in C++ in an object-oriented fashion. The C++ program for the entire server consists of 58,284 lines of code including 19,485 lines of comments, and 29,380 executable statements. The size of the TLS module is 15,645 lines. The single monolithic executable occupies 512 sectors (389 KB). The bare PC Webmail server and other bare PC servers do not use a local disk (they only require detachable mass storage). The server application directly communicates with the hardware (in this case an X86-based CPU). This approach can also be used to build pervasive devices, gateways, routers, or sensors that host a small efficient secure bare PC Webmail server. The software is placed on the USB and includes: the boot program, startup menu, AO executable, and the persistent file system (used for User Profiles, emails, and attachments). The USB containing this

information is generated by a tool (designed and run on MS-Windows) that creates the bootable Bare PC application for deployment. The tool, which generates the boot load sector, and copies the executable and associated files to the USB, consists of only 469 lines of C++ code.

In a bare PC server, the design and implementation can get very complex if flexible dynamic user interfaces are used. The user interface have being optimized by designing static PHP pages which can be pre-parsed and indexed for efficient processing without incurring any parsing delays during transactions. As the data is encrypted throughout the client and server data transactions, it poses daunting challenges in a bare PC to test and debug the server. Implementation of such a system requires a thorough understanding of the protocol layers, system functionality and behaviors. A snippet of the secure webmail server implementation in C++ is shown in figure 22. As seen in the snippet, the implementation presented its own complexities and challenges. Implementing such complicated systems requires complex algorithms and intertwining of the necessary protocols to eliminate inter-protocol processing overheads and also to ensure operational functionality. To handle large encrypted email messages, flags are set inside the switch statement to check for incomplete packets that are still being processed by TCP or TLS and waits till all the packets arrive before parsing the message for the necessary parameters.

```
//COMPOSE interface
      case POST_WPACK1:  //3802
         if ((io.AOAgetTimer() - tcp.getTCBSMTPSTimer(tcbrno)) >= POST_WPACK1_TIMEOUT)
            {
              sendReset1(tcbrno);
              return -2;
            }
       if(tcp.getTCBState(tcbrno) == ESTAB &&  tcp.getTCBPacketArrived(tcbrno) == 1 )
        {
        tcp.resetTCBPacketArrived(tcbrno);
         len = tcp.getTCBTLSPtrOut(tcbrno); //length of the total data as of now
         if  (postm[0] == 'P' && postm[1] == 'O' && postm[2] == 'S' && postm[3] == 'T')
              {
                  headerlen = parsePostData(postm, sessionstr, &contentlength, len);
                 postm = postm + headerlen; // no need to parse the header again
            retcode = tcp.setSMTPCState(tcbrno, contentlength);//save the content length
            if (contentlength  > (len - headerlen + 40))
                 {
                    retcode = tcp.setTCBTLSAssembleFlag (tcbrno, 1);//set the flag
                   retcode = tcp.setTCBTLSonepacket(tcbrno, 0); //reset onepacket flag
                    retcode = tcp.setTCBTLSlcounter(tcbrno, 0);  //reset tlslcounter
                    retcode = tcp.setTCBSMTPsndnxt(tcbrno, 0);   //reset data in pointer
                    retcode = tcp.setTCBTLSPtrOut(tcbrno, tcp.getTCBTLSPtrOut(tcbrno) - 32);
                   return 0;
                 }
                 else
                  retcode = tcp.setTCBTLSPtrOut(tcbrno, 0);    //reset tlsptrout
                 retcode = tcp.setTCBTLSAssembleFlag(tcbrno, 0);  //no more assembly, start again
                userid = tcp.getTCBPostUserId(tcbrno);
                 retcode = parseMessagePOST(postm, len);
                 tcp.setTCBSMTPCCnt(tcbrno, 0);              //reset DBDataInSize
                 tcp.setDBDataOutSize(tcbrno, 0);            //reset DBDataOutSize
                 retcode = tcp.setTCBTLSonepacket(tcbrno, 0); //reset onepacket flag
                 retcode = tcp.setTCBTLSlcounter(tcbrno, 0);  //reset tlslcounter
                 retcode = tcp.setTCBSMTPsndnxt(tcbrno, 0);   //reset data in pointer
                 if (retcode == 0)
                 {
                 retcode = methodGETPOST(tcbrno, currenttask);
                 dflag = 1;
                 tcp.setSMTPSState (tcbrno, POST_WEND);  // end POST process
                 tcp.setTCBState(tcbrno, 0x05);              //bring it back to ESTAB state
                 tcp.resetTCBPostCount(tcbrno);              //reset
                 }
                 else
                 {
                    io.AOAPrintText("PostObj: should not happen!", Line23+20);
                    io.AOAprintHex (retcode, Line23+100);
                    return -111;
                 }
                 statec = tcp.getSMTPSState(tcbrno);
                 tcp.setTCBState(tcbrno, 0x05);              //bring it back to ESTAB state
              io.AOAprintHex(statec, Line16+80);
                 return 0;                              //return to the caller
              } //end of POST condition if
          }
         break;
```

Figure 22. Snippet of the Implementation Code for Processing Compose

B. User Interface PHP/HTML Script Implementation

The user interface PHP/HTML scripting was implemented in Macromedia Dreamweaver. This is a windows based application that is used for standard web designs. There are six PHP/HTML scripts for user interactions with the webmail server. Screen shots of the user interfaces are shown in figures 23-26. Figure 23 is the user login interface; this script consists of 300 lines of PHP/HTML codes and 30 lines of comments. The main screen page shown in figure 24 consists of 880 lines of PHP/HTML codes and 50 lines of comments. On the main screen the user can either compose a new message by invoking the compose page shown in figure 25, view the inbox for existing emails as shown in figure 26 or logout of the system. The compose screen script consists of 920 lines of PHP/HTML codes and 130 lines of comments while the inbox screen consists of 600 lines of PHP/HTML codes and 50 lines of comments. There is a script for reading emails and this consists of 198 lines of codes and 22 lines of comments. Another script allows the user to read attachments files and it consists of 150 lines of codes and 32 lines of comments.

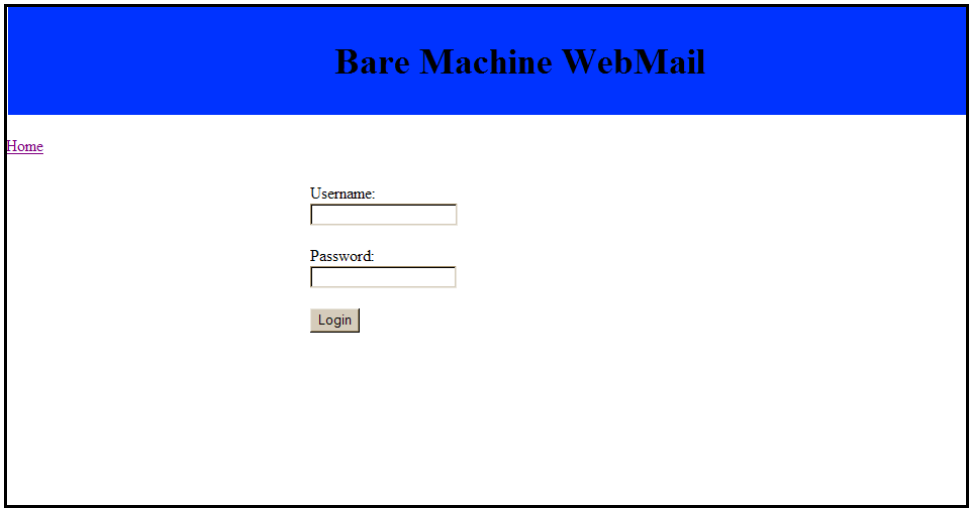

Figure 23. Login Screen

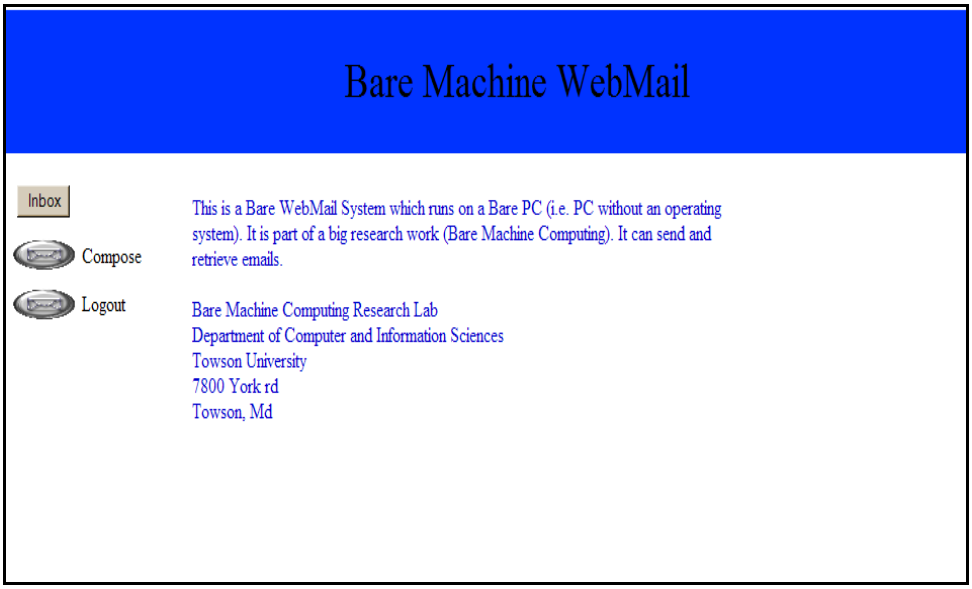

Figure 24. Main Screen

Figure 25. Email Compose Screen

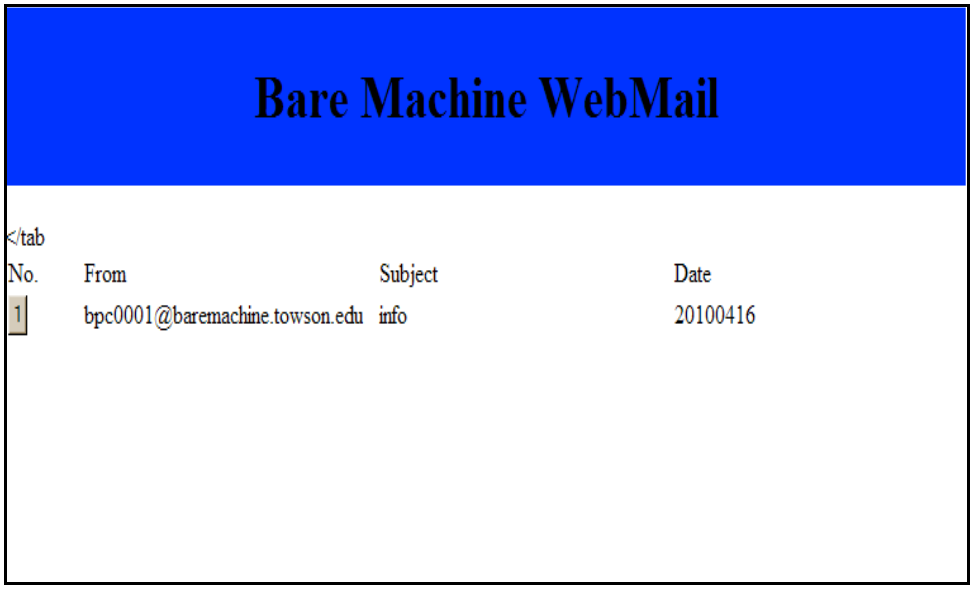

Figure 26. Inbox Screen

A snippet of the PHP/HTML scripts for the login page is displayed in figure 27. It shows the implementation of the user login form entries that will be sent to the server for authentication before the user can gain access to the system. User profiles are manually created and transferred into the server's memory at run time. These profiles are used by the server to validate the user's credentials and allow access for the user to use services on the server. The PHP/HTML scripts are also loaded into the server's memory at server initialization and they reside there until the server shuts down. User request for these pages are delivered through web clients.

```
<?php
/**
 * login.php -- simple login screen
 * Patrick Appiah-Kubi
 */
/** This is the login page */
define('PAGE_NAME', 'login');
define('B_PATH','../');
require_once(B_PATH . 'functions/global.php');
//require_once(B_PATH . 'functions/i18n.php');
require_once(B_PATH . 'functions/plugin.php');
require_once(B_PATH . 'functions/constants.php');
require_once(B_PATH . 'functions/page_header.php');
require_once(B_PATH . 'functions/html.php');
require_once(B_PATH . 'functions/forms.php');
global $custom_session_handlers;
if (!empty($custom_session_handlers)) {
   $open    = $custom_session_handlers[0];
   $close   = $custom_session_handlers[1];
   $read    = $custom_session_handlers[2];
   $write   = $custom_session_handlers[3];
   $destroy = $custom_session_handlers[4];
   $gc      = $custom_session_handlers[5];
   session_module_name('user');
   session_set_save_handler($open, $close, $read, $write, $destroy, $gc);
}
header('Pragma: no-cache');
do_hook('login_cookie');
$loginname_value = (b_GetGlobalVar('loginname', $loginname) ? htmlspecialchars($loginname) : '');
$custom_css = 'none';
// Load default theme if possible
if (@file_exists($theme[$theme_default]['PATH']))
  @include ($theme[$theme_default]['PATH']);
displayHtmlHeader( "$org_name - " . _("Login"), $header, FALSE );
echo "<body text=\"$color[8]\" bgcolor=\"$color[4]\" link=\"$color[7]\" vlink=\"$color[7]\" alink=\"$color[7]\"
onLoad=\"_loginpage_onload();\">" . //add code here
    "\n" . addForm('redirect.php', 'post', 'login_form');
$username_form_name = 'login_username';
$password_form_name = 'secretkey';
do_hook('login_top');
if(b_getGlobalVar('mailtodata', $mailtodata)) {
   $mailtofield = addHidden('mailtodata', $mailtodata);
} else {
   $mailtofield = '';
}
echo  html_tag( 'table',
       html_tag( 'tr',
         html_tag( 'td',
               '<b>' . sprintf (_("%s Login"), "Bare PC") . "</b>\n",
               'center', $color[0] )
            ) .
            html_tag( 'tr',
               html_tag( 'td',  "\n" .
                  html_tag( 'table',
                     html_tag( 'tr',
                        html_tag( 'td',
                           _("Name:") ,
                        'right', '', 'width="30%"' ) .
                        html_tag( 'td', addInput($username_form_name, $loginname_value, 0, 0, ' onfocus="alreadyFocused=true;"'),
                        'left', '', 'width="70%"' )
                        ) . "\n" .
                     html_tag( 'tr',
                        html_tag( 'td',
                           _("Password:") ,
                        'right', '', 'width="30%"' ) .
                        html_tag( 'td',
                           addPwField($password_form_name, null, ' onfocus="alreadyFocused=true;").
                             $mailtofield .
                                                   addHidden('just_logged_in', '1'),
                        'left', '', 'width="70%"' )
                     ) ,
                  'center', $color[4], 'border="0" width="100%"' ) ,
               'left',$color[4] )
            ) .
            html_tag( 'tr',
               html_tag( 'td',
                  '<center>'. addSubmit(_("Login")) .'</center>',
               'left' )
            ),
         '', $color[4], 'border="0" width="350"' ) . '</center>',
      'center' )
   ) ,
'', $color[4], 'border="0" cellspacing="0" cellpadding="0" width="100%"' );
do_hook('login_form');
echo '</form>' . "\n";
do_hook('login_bottom');
?>
</body></html>
```

Figure 27. Snippet of the PHP/HTML Login Script

VII. PERFORMANCE MEASUREMENTS

A. Initial Measurements

For proof of concept, initial functionality tests were conducted on local area network (LAN) and wide area network (WAN) environments. Basic operations such as login GET and POST requests, composing and sending, retrieving regular emails and attachments. Results of these tests were compared with OS-based servers that have similar functionalities and capabilities.

1) LAN Measurements

- *Experimental Setpup*

The experimental setup involved four Dell OptiPlex GX520 PCs with Intel Pentium 4, 3.2GHz Processor, 1GB RAM and 3Com EtherLink XL 10/100 NIC card. A LAN and WAN networks were set up for the test using a Linksys 16 speed dual port hub to connect the four computers. One of the computers running on windows environment with Internet Explore and a Wireshark network analyzer tool installed on it served as the client to the servers. Another one was used for Bare PC and the other two for Windows and Linux. For fair comparison, unnecessary services, processes and programs that add overhead to the OS-based systems were disabled. Figure 28 below is the experimental setup for the LAN test.

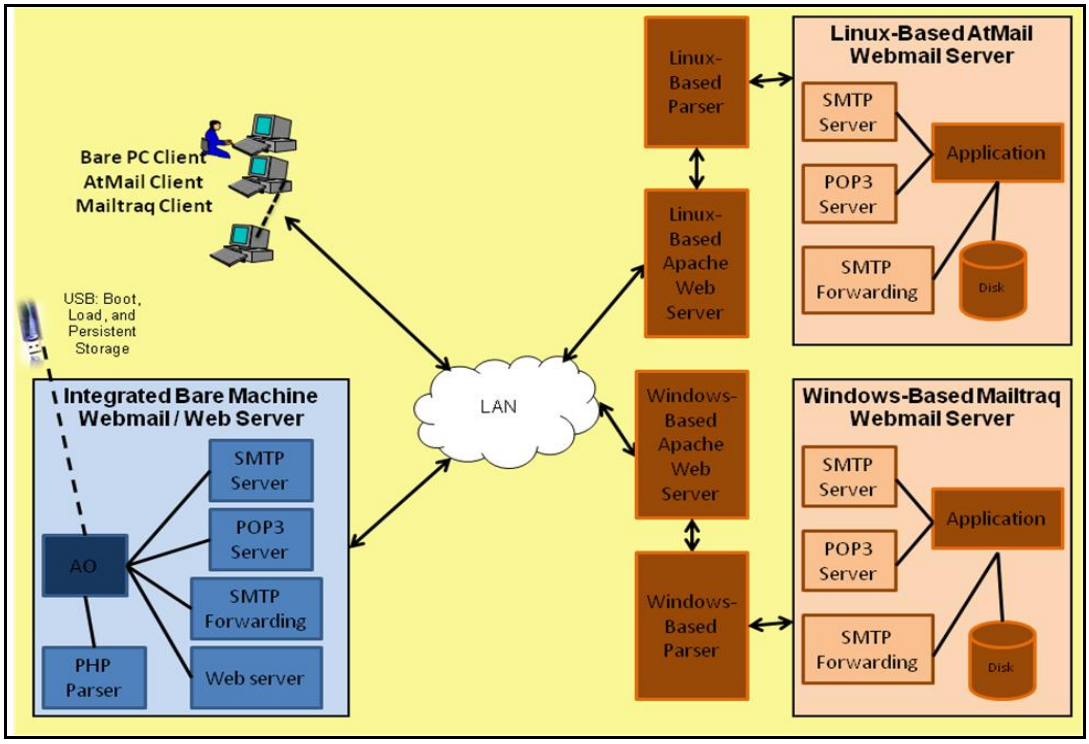

Figure 28. Experimental Setup for the LAN Test

- *LAN Results*

Figure 29 shows the Login GET request command from a Web client to the server. The client/server connection for GET requests are closed for each connection. The Webmail servers under test behave the same way during the TCP connection establishment. However, there is a noticeable difference in the server's response to GET request. The bare PC and Linux servers respond (i.e. ACK) to the GET request in a reasonable amount of time (bare PC server responds in 0.153 milliseconds in comparison to the Linux server in 0.361 milliseconds). However, the Windows server takes a period of 115 milliseconds. The first data for the client request sent from the servers vary in a wide range. The time from GET to first data for bare PC server is 0.258 milliseconds, 183 milliseconds for the Windows server, and 85 milliseconds for the Linux server. The

total Login request time for bare PC is an order of magnitude improvement over the Windows and Linux servers. The faster response time from GET to first data in bare PC indicates the fast processing time by the RCV task in bare PC.

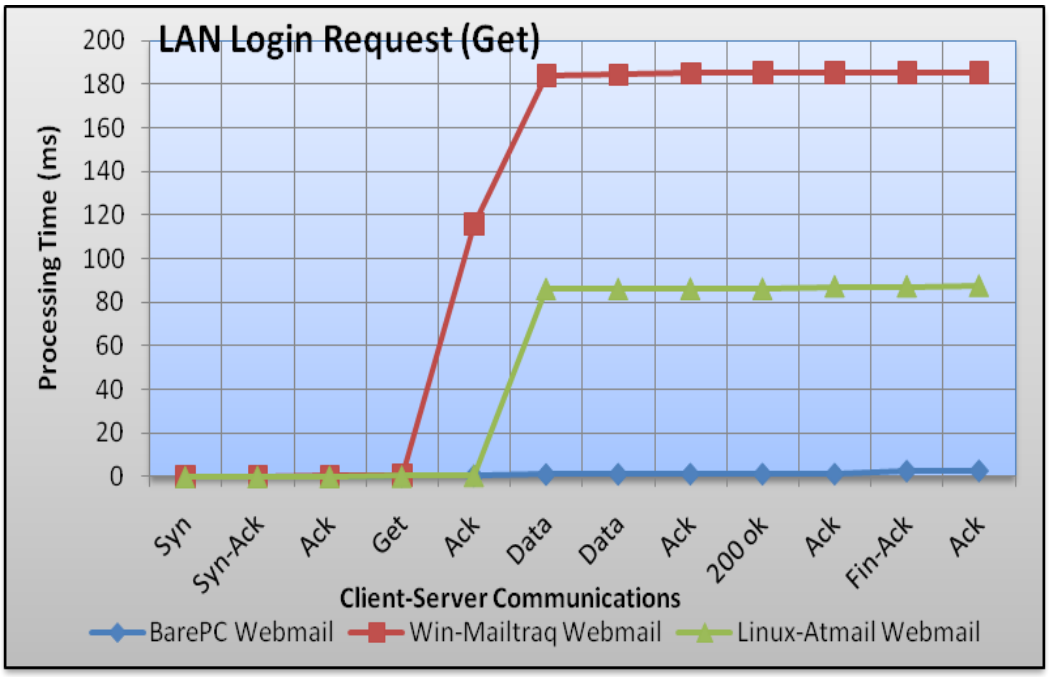

Figure 29. LAN Login Request (GET)

Figure 30 shows the Login POST request command from the Web client to the server. In this case, the client connection to the server is OPEN for multiple POST commands. The dominant time in this request is between the arrival of POST data and server's response (i.e. 302 found). The bare PC server takes 0.613 milliseconds; the Windows server takes 329 milliseconds while the Linux server takes 223 milliseconds between the time for POST data and response. The fast response in bare PC for the POST command is attributed to the POST task responding to the request in a single thread of execution without interruption and process switching. This single thread of execution used to

process POST data is designed by the AO programmer at design time thus making bare PC Webmail server a high performance server.

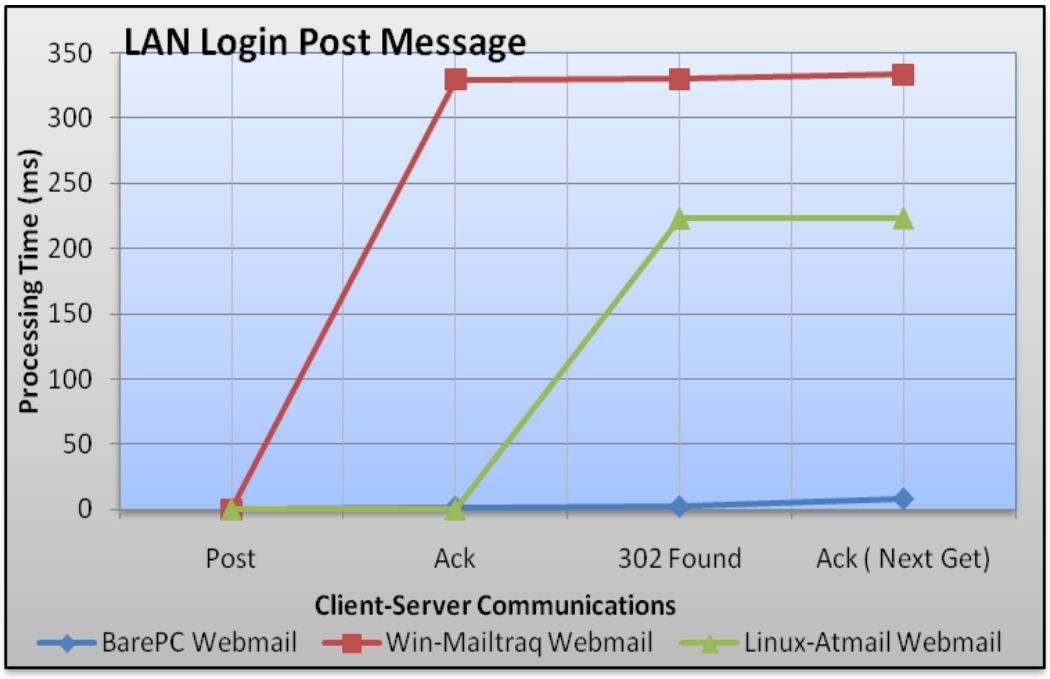

Figure 30. LAN Login Request (POST)

Figure 31 illustrates the email message processing time for a single request. The message is included as part of the content of the email. The message size is varied from 1000 bytes to 120,000 bytes and the processing times are captured with the Wireshark tool. The bare PC Webmail server is resistant to large email sizes until 60,000 bytes, but its processing time increases gradually after 60,000 bytes. The bare PC Webmail server stores the emails on a USB flash memory while the Windows and Linux servers store their files on the hard disk. The USB file I/O for Webmail requests is contributing to most of the processing time in the bare PC server. The hard disk file I/O for the other server seems to be faster than the USB file I/O in the bare PC system. The Linux-based server shows a significant spike between 20,000 and 40,000 bytes, which could be

attributed to internal processing mechanisms in the system. This requires further studies and analysis to determine the cause of the spike.

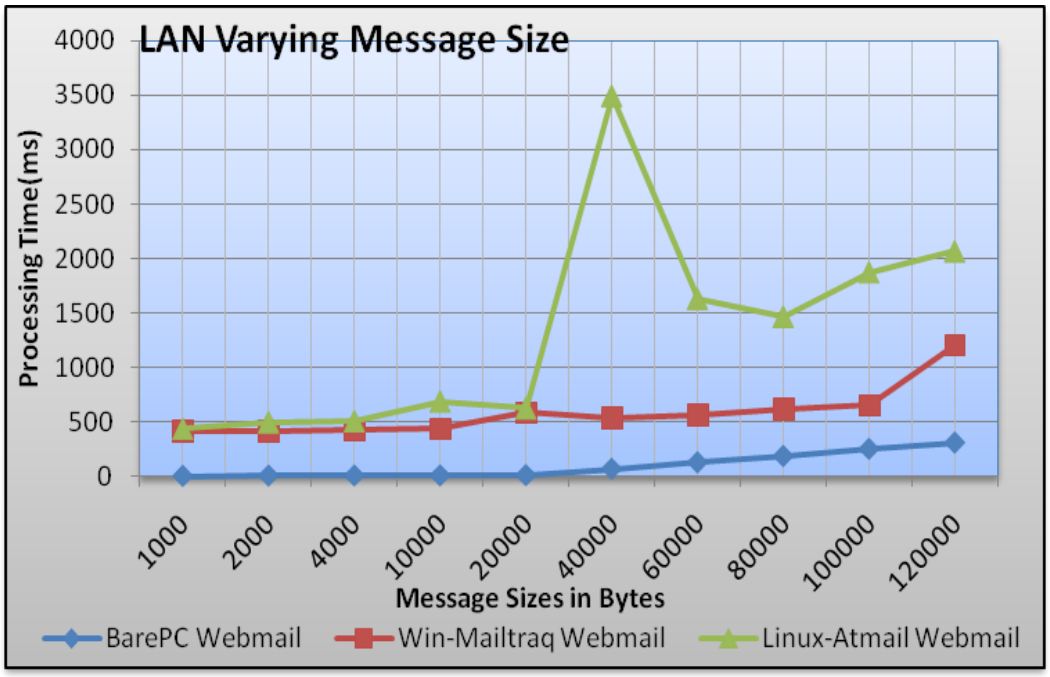

Figure 31. Compose with Varying Message Sizes

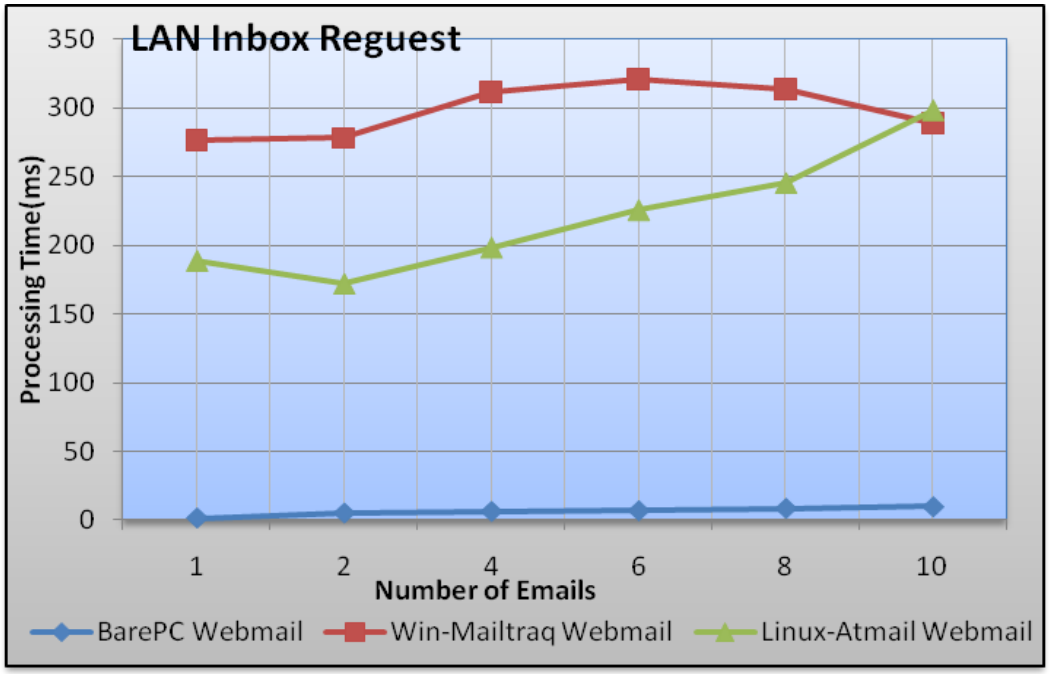

Figure 32. Varying Number of Emails in Inbox

Figure 32 compares the processing times for inbox request made to the Webmail servers. The numbers of emails in the inbox varied from 1 to 10 to see their effect on processing time, whilst the email size remained constant. As seen in the figure, the bare PC Webmail server processes the inbox request faster than the other servers.

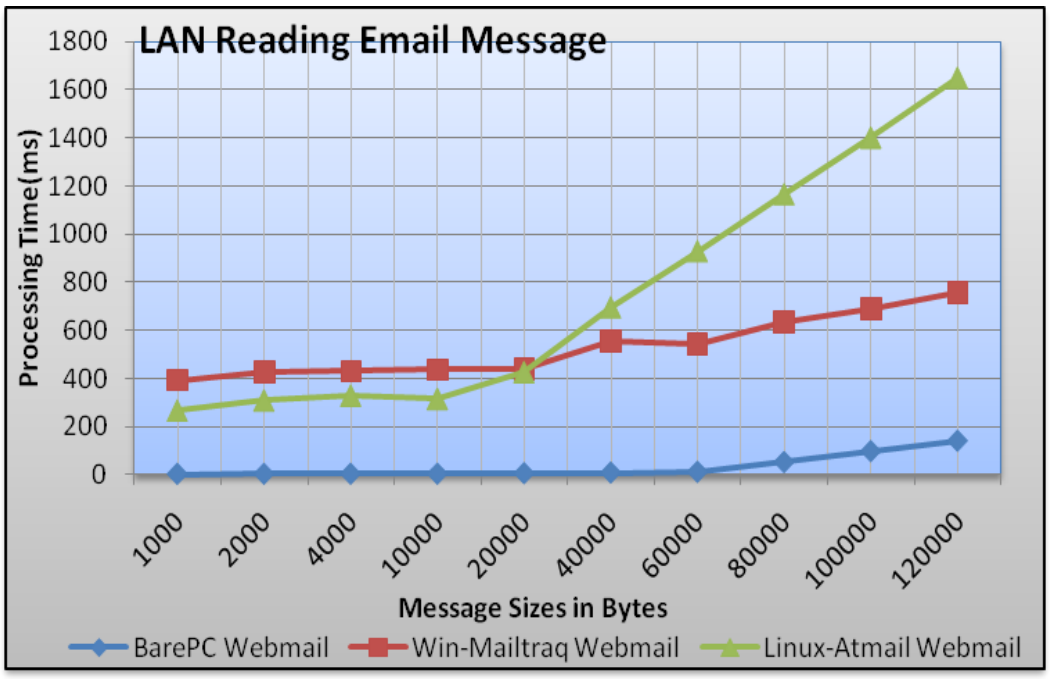

Figure 33. LAN Retrieving Varying Message Size

Figure 33 is a comparison of the processing times for the Webmail servers with varying message sizes for email retrieval. In bare PC Webmail server the HTTP GET tasks operates as a single thread of execution without interruption and process switching. All incoming packets are placed in the Ethernet DPD buffers while the server waits for the ACKs from the client. For large amount of transmitted data (i.e. 60,000 bytes or more) the task will be suspended, while waiting for ACKs from the client. This is clearly demonstrated in the graph in the processing time above 60,000 bytes of data.

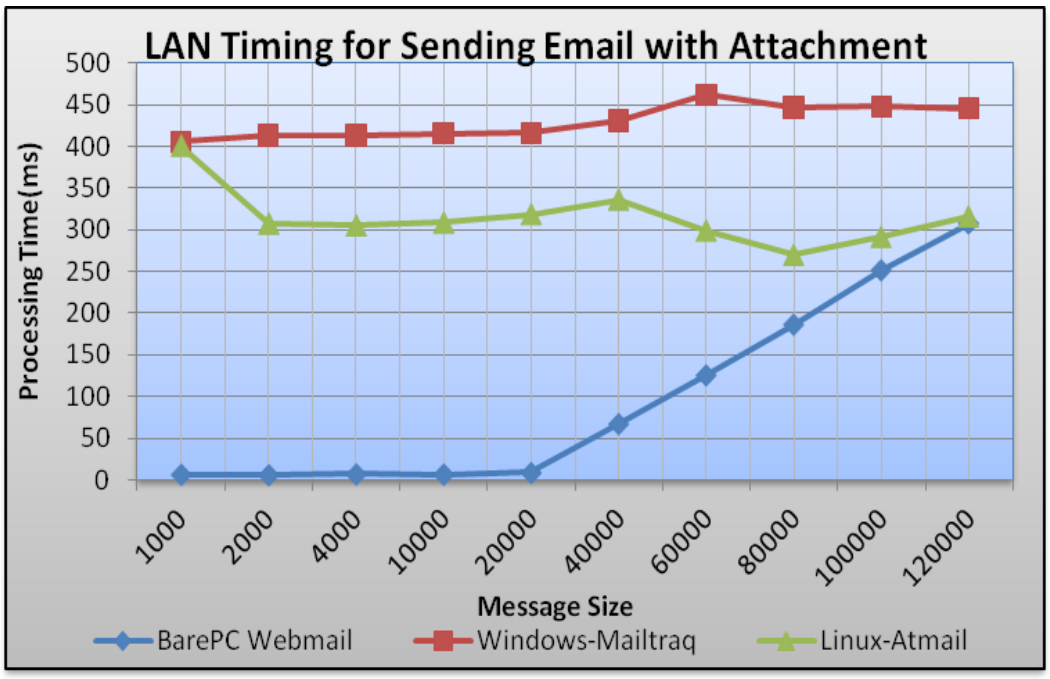

Figure 34. LAN Sending Email with Attachments

The bare PC Webmail server was also designed to send and retrieve attachment messages. The performance of this functionality is evaluated in Figure 34. The figure gives a fair view of the performance measurement for sending emails with attachments using varying message sizes of 1000 to 120,000 bytes. Bare PC shows a stronger performance over Windows and Linux in the processing time just as it did in the regular email without attachment. The dramatic increase in processing time for message size after 20,000 bytes is caused by the write operation in the USB. The USB driver is limited to 40 sectors for a single write operation. For more than 40 sectors it requires multiple write operations which could affect the total processing time. It was observed that the write operations in the USB are much slower than the read operation due to the inherent characteristics of USBs. Also, each USB write is limited to approximately 20,000 bytes of data for each write operation due to its limitations in the data descriptors, thus it

requires multiple write operations for a large data. The other Webmail servers show a flat processing time as they are storing files on the hard disk which demonstrates uniform write time.

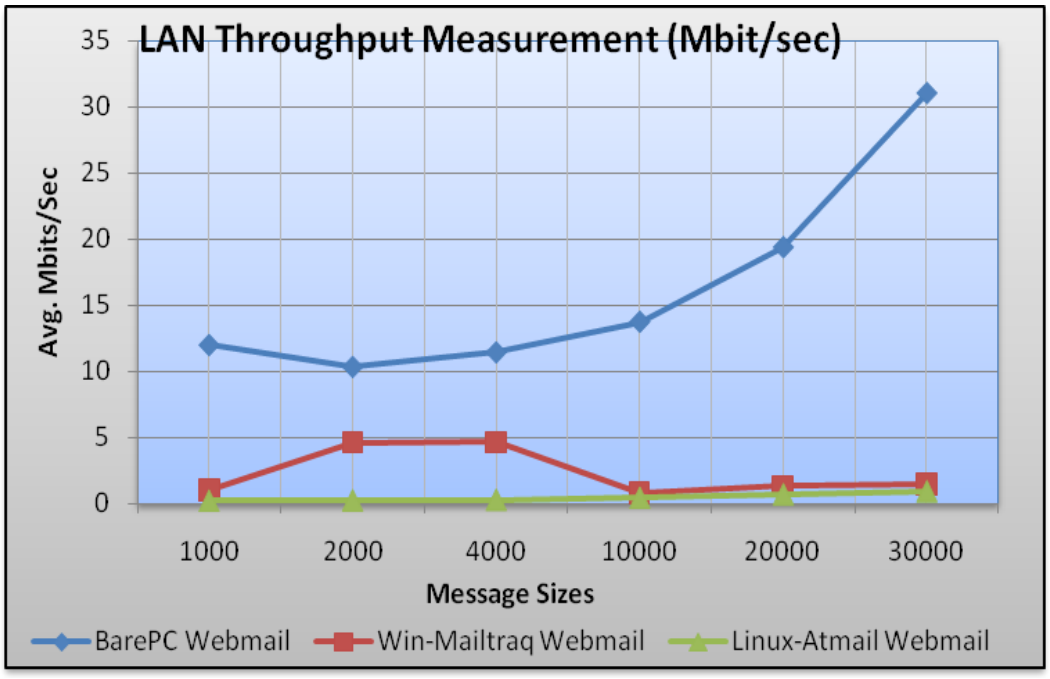

Figure 35. Throughput Measurement

Figure 35 shows throughput test results of the Webmail systems based on the average Mbit/sec versus varying message sizes. Bare PC Webmail server exhibits a better throughput over the other servers due to the justifications made in the other test results.

2) WAN Measurements

- *Experimental Setup*

A WAN connection as shown in figure 36 was established between the Webmail servers and a client PC located 50 miles away with about 30 router and firewall hops between the two destinations. The Visual Trace Route tool verified the hop count during the tests, which were performed in a contiguous time period. Similar measurements were

conducted over the Internet and a Wireshark tool was used to capture the packets and processing times.

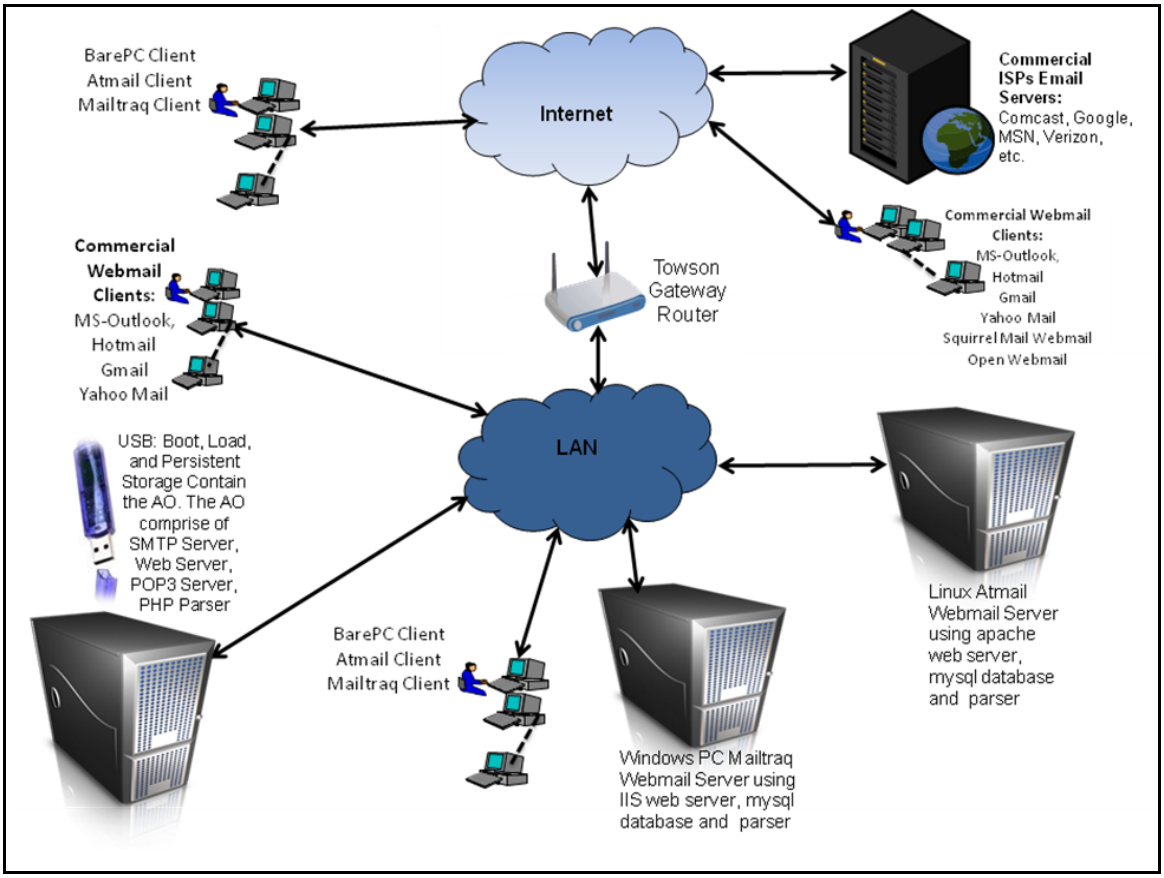

Figure 36. Experimental Setup for the WAN Test

- *WAN Results*

Figure 37 compares the internal timings for both WAN and LAN connections to the Bare PC Webmail server. It can be observed that the LAN timings are more smooth and linear than the WAN timings. The spikes between SYN-ACK and ACK as well as 200 OK and ACK are due to the router hops between the two locations. These timing spikes are caused by the client and network delays that cannot be controlled by the server.

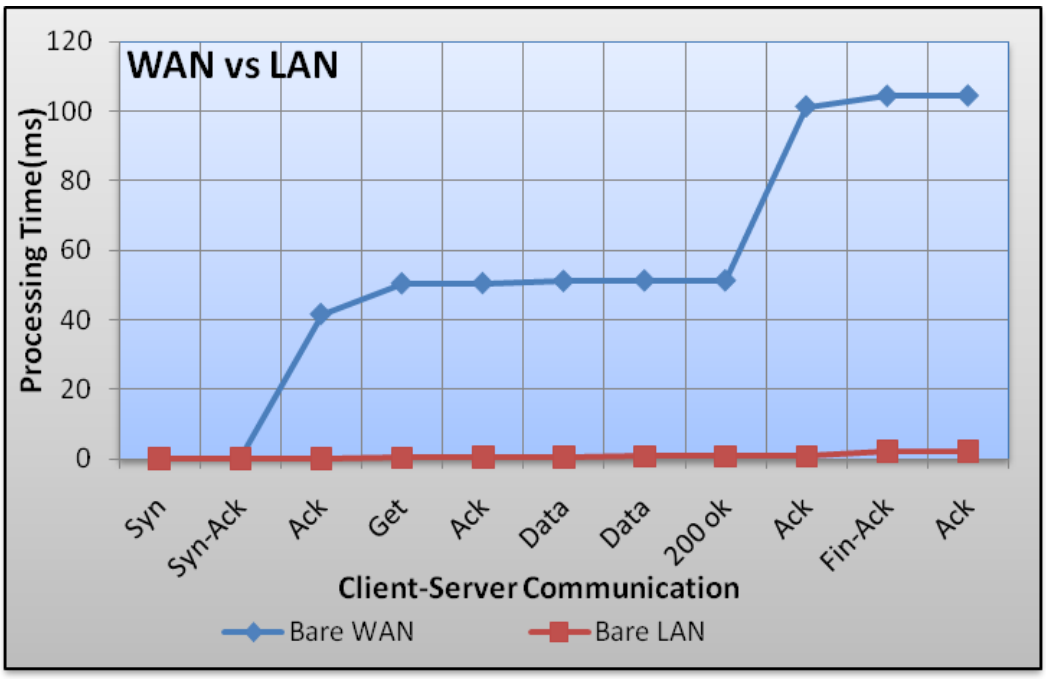

Figure 37. WAN/LAN Single Request Timing

Figure 38 illustrates the internal processing time against the client-server communications for Login GET request. Until, the GET timing point, all servers behave the same way and then their behavior in processing time varies until the operation is complete. SYN, SYN-ACK, ACK is pretty much establishing a connection between a client and a server which is expectedly the same for all servers as the bare PC Webmail server can't contribute to any performance improvements. However, the internal processing efficiencies kick-in once a GET requests arrives. The bare PC Webmail internal processing efficiencies contribute to the slower increase in processing times after the GET signal and a continued behavior of processing time until the end of the request.

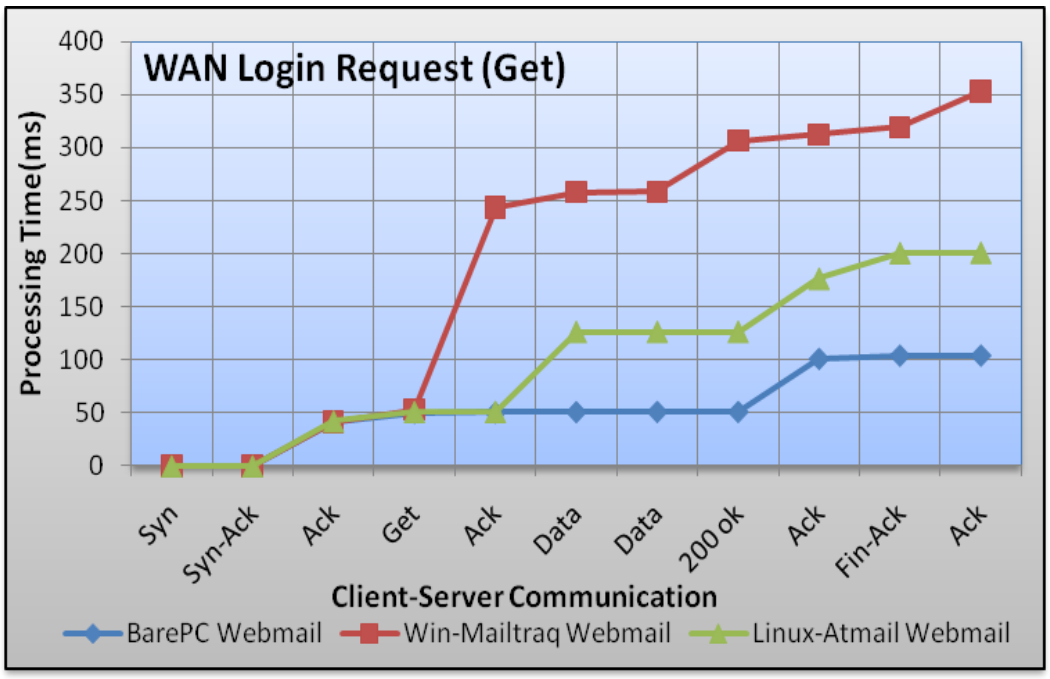

Figure 38. WAN Login Request GET Timing

Figure 39 is the internal processing time versus the client-server communication for Login POST request. Bare PC and Linux servers behave the same way to the POST response (i.e. ACK) timing point. However, the Windows server's response to POST took much longer time. Bare PC responds to the POST command in 0.153 milliseconds, while the Windows and Linux based systems respond to POST command in 297 milliseconds and 0.20 milliseconds respectively. Notice that the processing time for the Bare PC Webmail server is stable until the "302 found" response. The handshake between "302 found" and ACK initiated by the client took more time due to the client's delay in sending ACK to the "302 found" response.

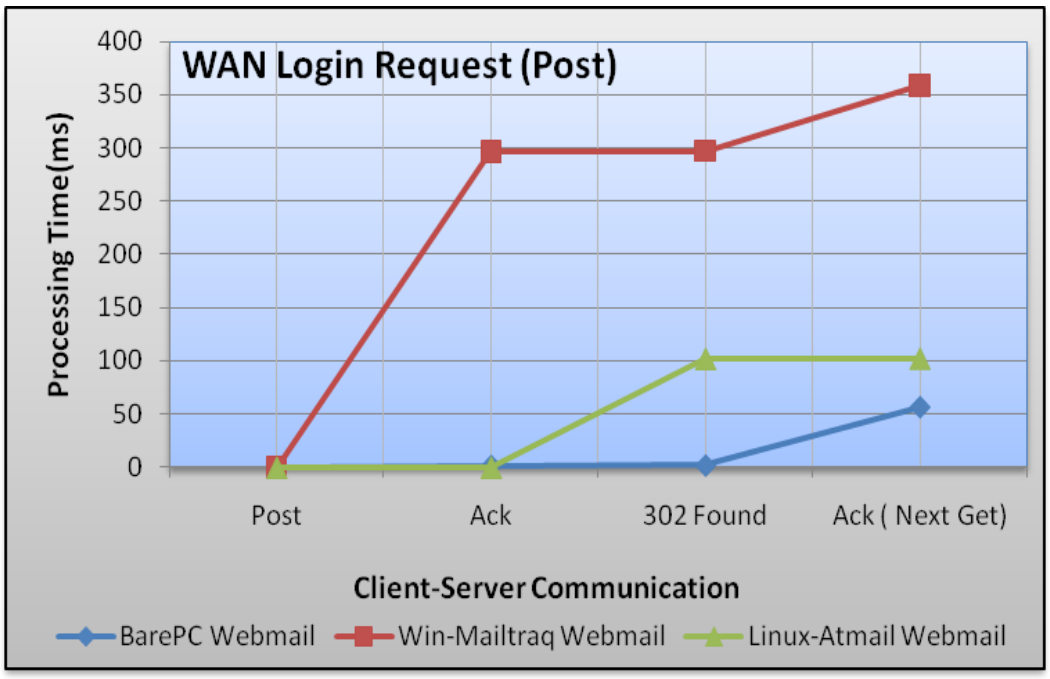

Figure 39. WAN Login POST Message Timing

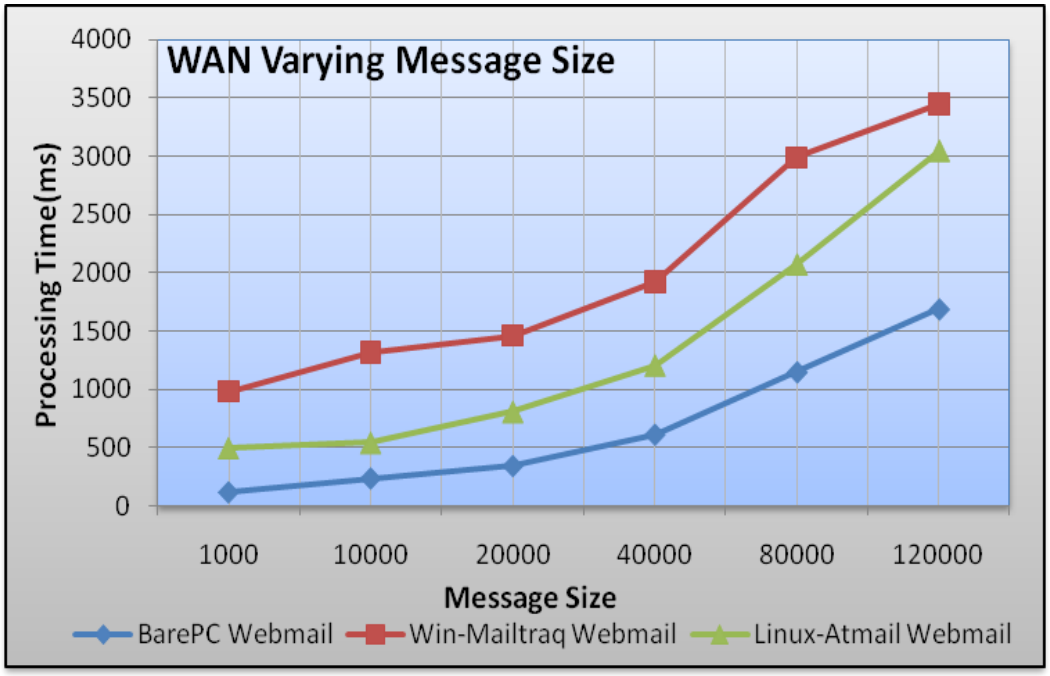

Figure 40. WAN Compose with Varying Message Sizes

As part of the WAN measurements, message sizes varying from 1000 to 120,000 bytes were composed and sent to the servers. The processing times were captured with Wireshark and plotted in Figure 40. The results indicate that the processing times of all Webmail servers increase linearly as the message size increases. This trend was expected due to the number of "hops" the data had to travel through and the accumulation of propagation time for large message sizes. However, the bare PC server compensated for the large message size propagation delays on the network with the local optimal processing times.

Figure 41 is a chart of an inbox request made to the servers. Each server has 6 messages to be retrieved for a single user. The processing time for retrieving the messages is captured with Wireshark and displayed in the figure. Bare PC retrieves all six messages into the inbox in 0.392 milliseconds, while Windows and Linux retrieve the messages in 277 milliseconds and 230 milliseconds respectively. The results achieved indicate that bare PC servers perform better than their OS-based counterparts in both LAN and Internet environments due to their streamlined architectural characteristics, protocol intertwining and associated optimizations.

Similarly Figure 42 illustrates processing time for reading individual emails in the inbox using varying message sizes. Notice that the processing time for reading messages on bare PC Webmail server is stable up to 40,000 bytes and then it increases slowly for large message sizes. This increase may be due to the Internet propagation time encountered during the transmission of large messages.

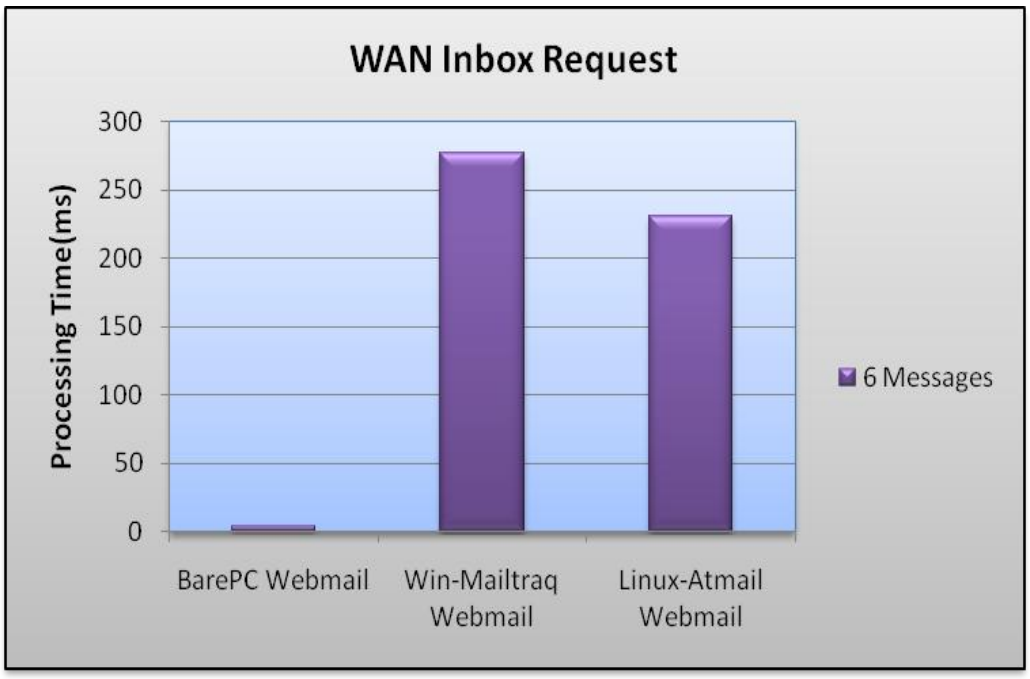

Figure 41. WAN Inbox Request for 6 Messages

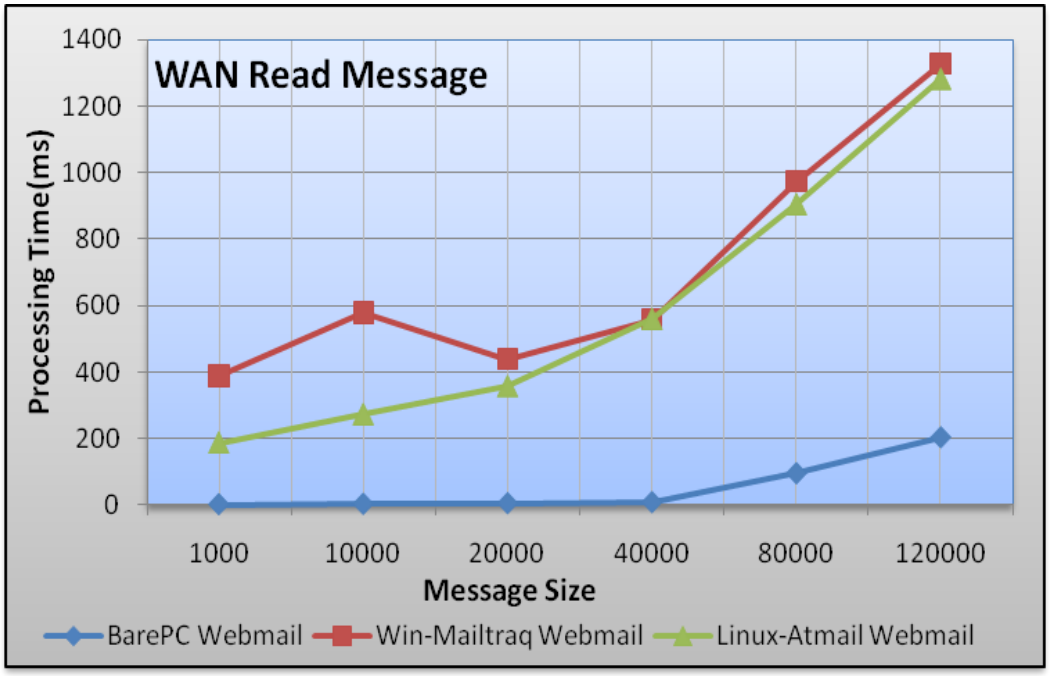

Figure 42. WAN Retrieving with Varying Message Size

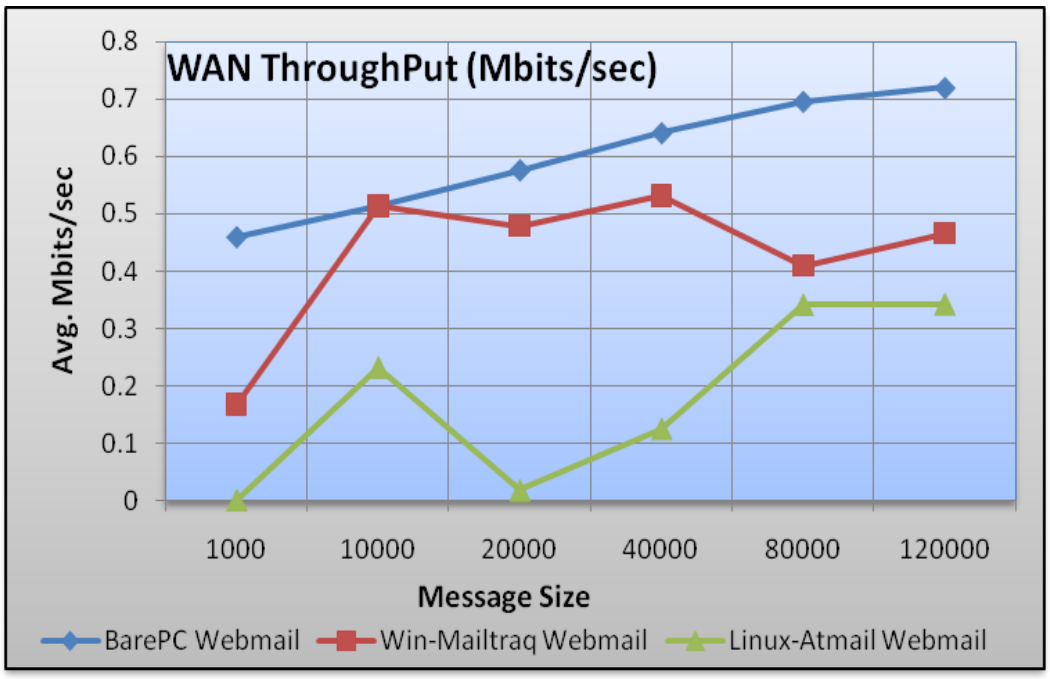

Figure 43. WAN Throughput Measurement

Throughput (Mbits/sec) measurements using varying message sizes were also captured during the Internet test and compared in Figure 43. The throughput for bare PC server is on the average 1.28 times better than the throughput of Windows and 3.13 times better than the throughput of Linux. Notice the fluctuations in throughput for the Windows and Linux servers in comparison to the bare PC server; this demonstrates the relatively stable behavior of the bare PC Webmail server. Bare PC Webmail server only has two major task types RCV and HTTP GET/POST and there is not much context switching involved in the server. Also there is a single thread of execution without interruption in the bare PC Webmail server. These factors contribute greatly to the behavior of bare PC. The AO programmer takes scheduling decisions (suspend, resume, complete) at program time, which makes the system very efficient and application driven. A conventional OS based server is managed and controlled by the underlying OS or

kernel thus contributing to the overhead in the server processing time. This performance display is not just the overhead of OS alone; it is also partly due to the bare PC Webmail server architecture which was designed for performance.

3) Stress Test Measurements

The two most popular tools used to measure the performance of the Webmail server were HTTPERF that run on Linux and WebStress Tool that run on Windows. The WebStress tool was used to test how the Webmail servers process concurrent POST requests. The number of users who can concurrently send POST messages to the servers was varied from 1 to 10. Multiple POST messages were sent and the total processing time was captured by the tool. The results of the test were plotted in Figure 44. As indicated in the chart, the Bare PC Webmail server on the average performed 53.9 times faster than the Windows and 27.9 times faster than the Linux.

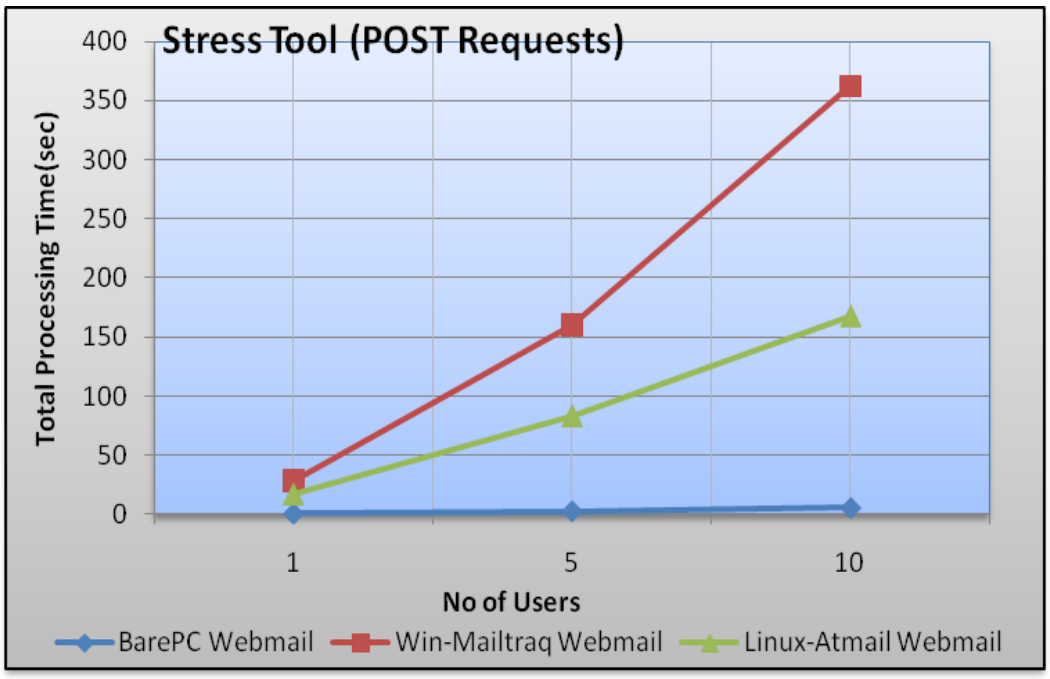

Figure 44. Stress Tool Measurements

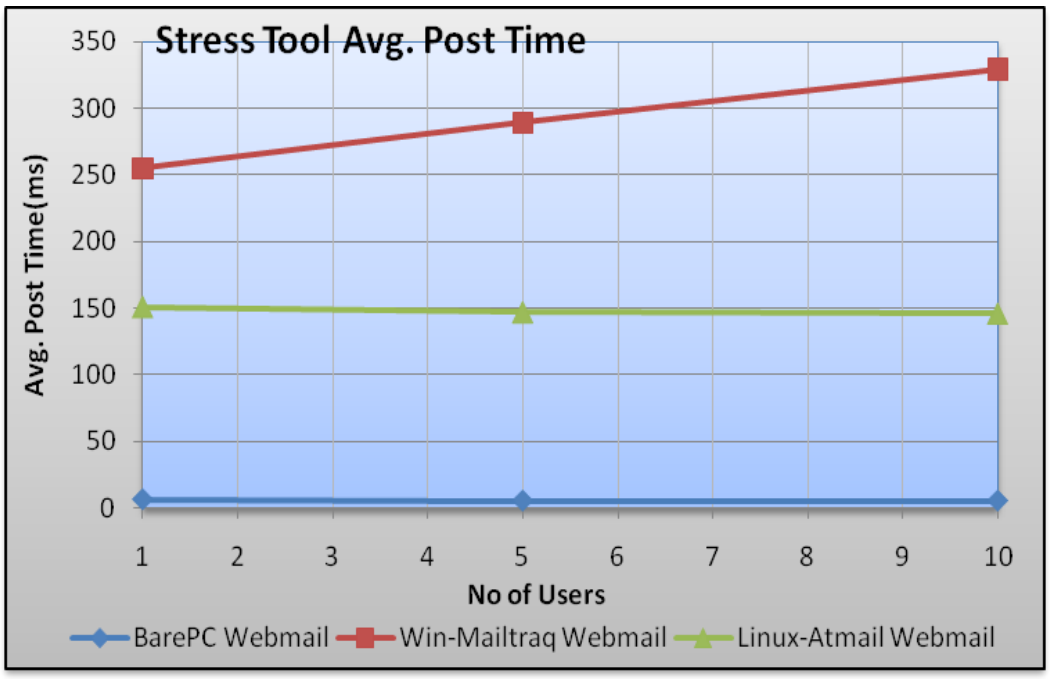

Figure 45. Avg. Post Time against Number of Users

Figure 45 validates the observations made in Figure 45 using the WebStress tool. This demonstrates the average time the server takes to process a single POST request and send its response. As it can be seen, the Bare PC Webmail server takes much smaller processing time on the average to process the POST request.

The HTTPERF tool was used to further test the performance of the servers. The test was conducted by varying the number of concurrent connections from 100 to 2000 with a connection rate of 1000. The total processing time, response time and CPU time (client side) were captured by the tool and plotted. Figure 46 compares the total processing time versus the number of connections for the Webmail servers and Figure 47 shows the corresponding response times. Notice the spike between 400 and 600 connections on the Linux system and the huge acceleration of the Windows system as the number of connections increase. The Bare PC Webmail server once again demonstrates the stable

and steady behavior for this stress test. Due to high fluctuations and large values in response times by the Windows-based Webmail server, its chart was omitted from Figure 47.

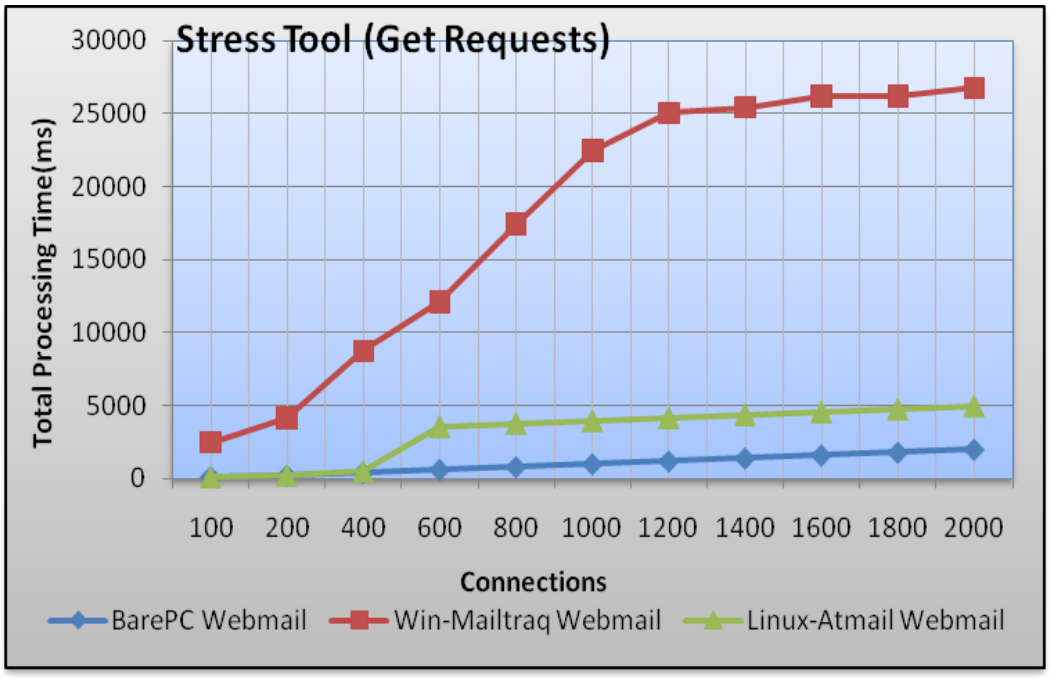

Figure 46. Connections against Processing Time

Figure 48 shows a plot of CPU time measured at the client side against the number of connections with a connection rate of 1000. As shown in the chart, there are similarities in the behavior of the three servers, however in the OS based servers; more CPU time was required due to the OS overhead and other application related processes. The Bare PC Webmail server's small CPU time demonstrates more capacity of the Bare PC to handle additional workload.

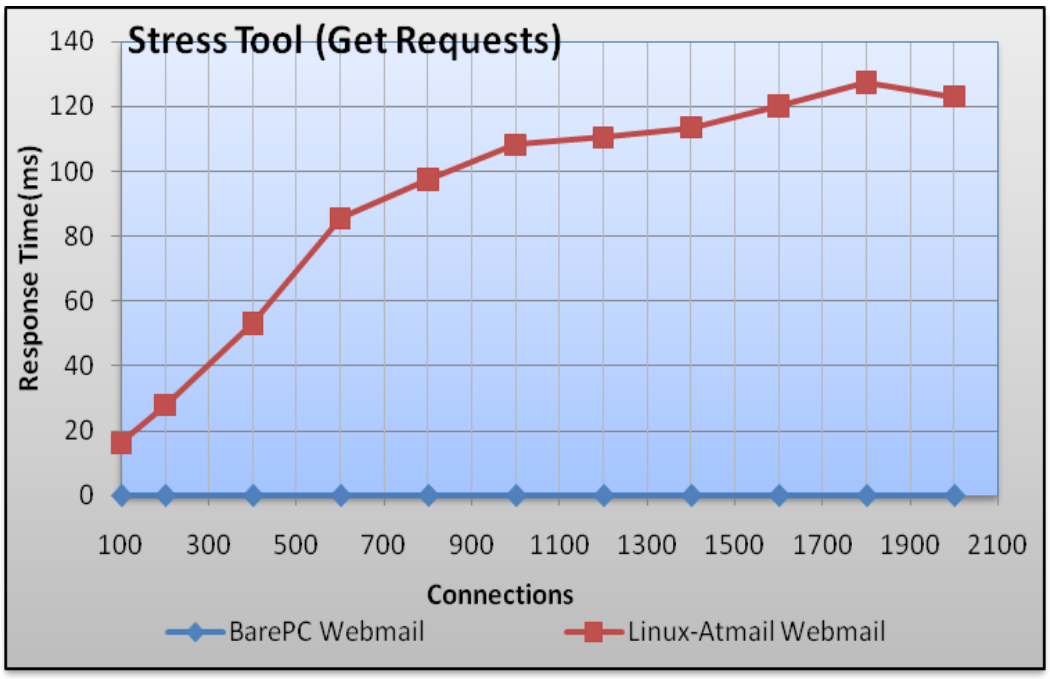

Figure 47. Connections against Response Time

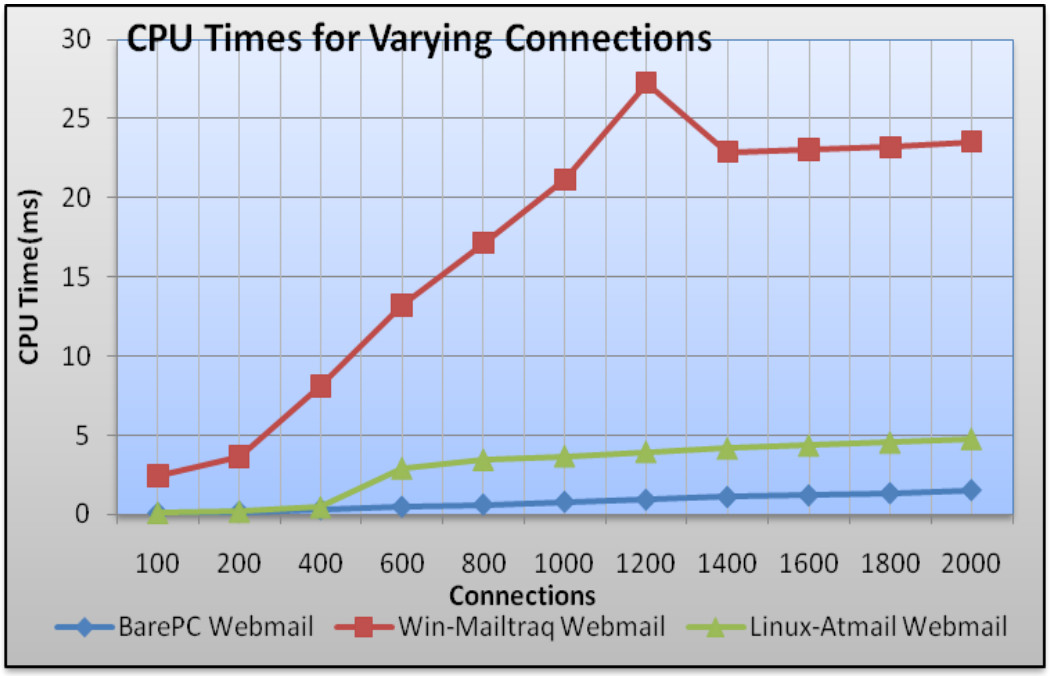

Figure 48. CPU Processing Time (Client Side)

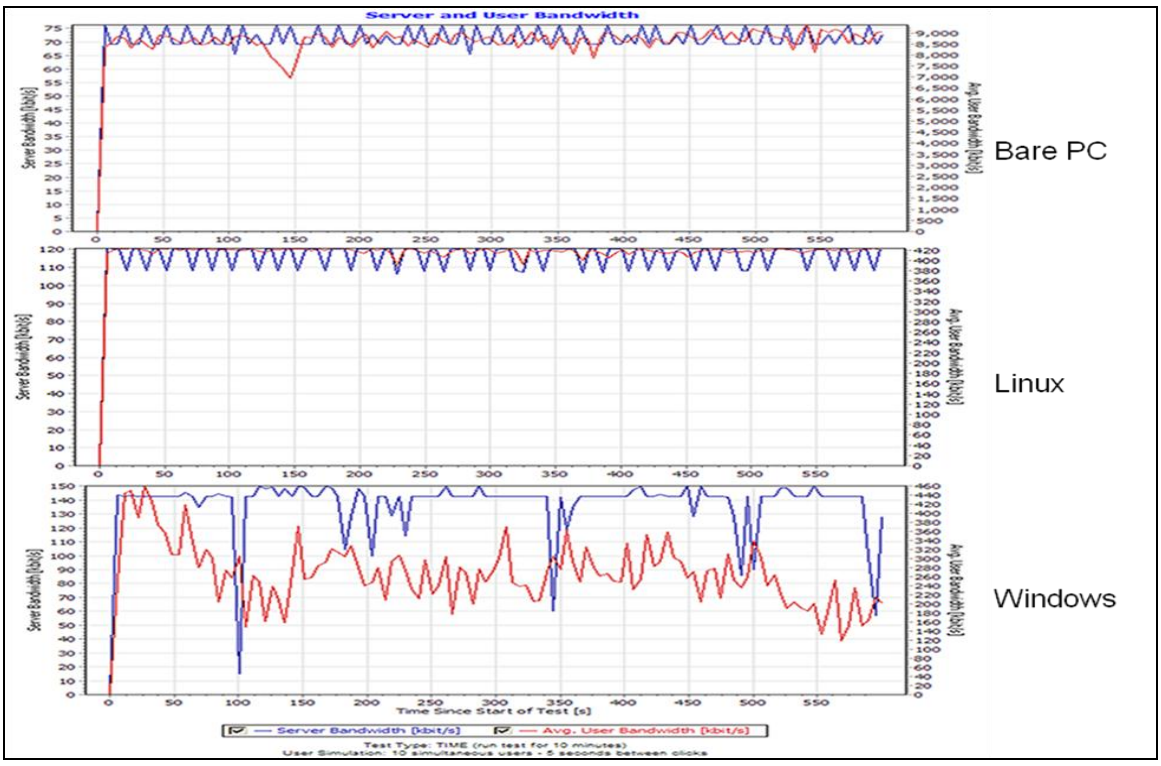

Figure 49. Stress Tool Graph for Server and User Bandwidth

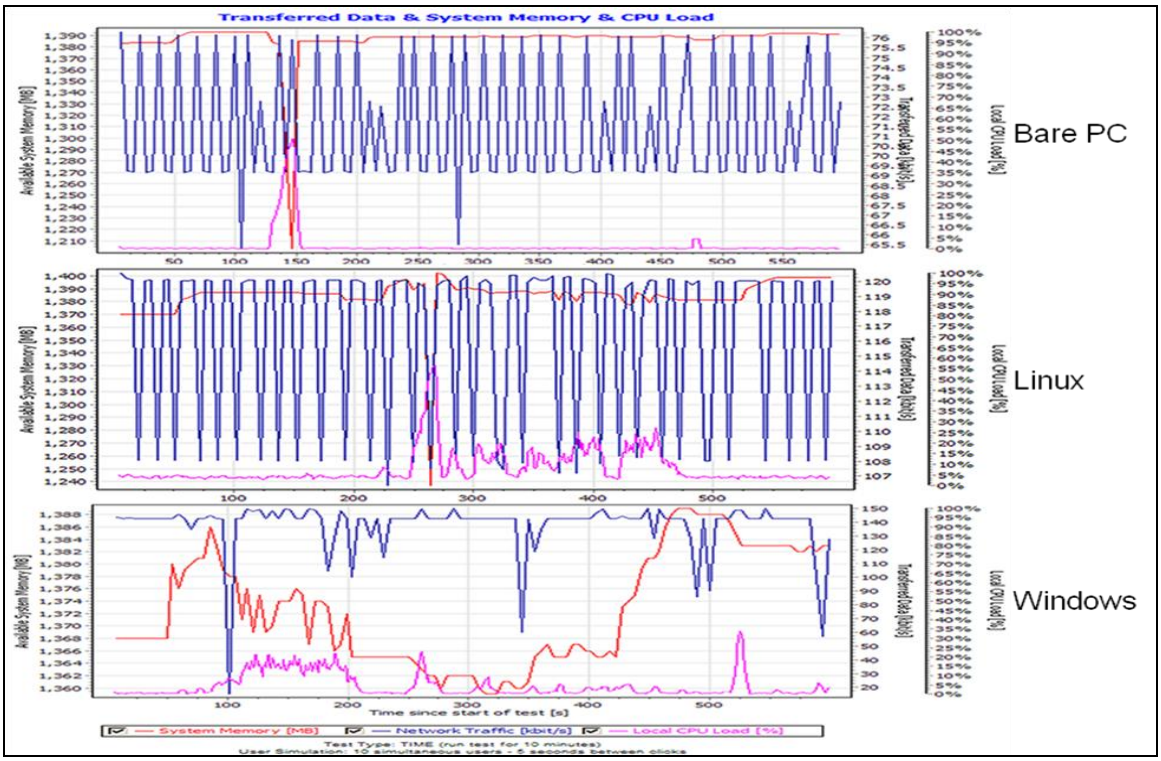

Figure 50. Stress Tool Graph for Server CPU Load

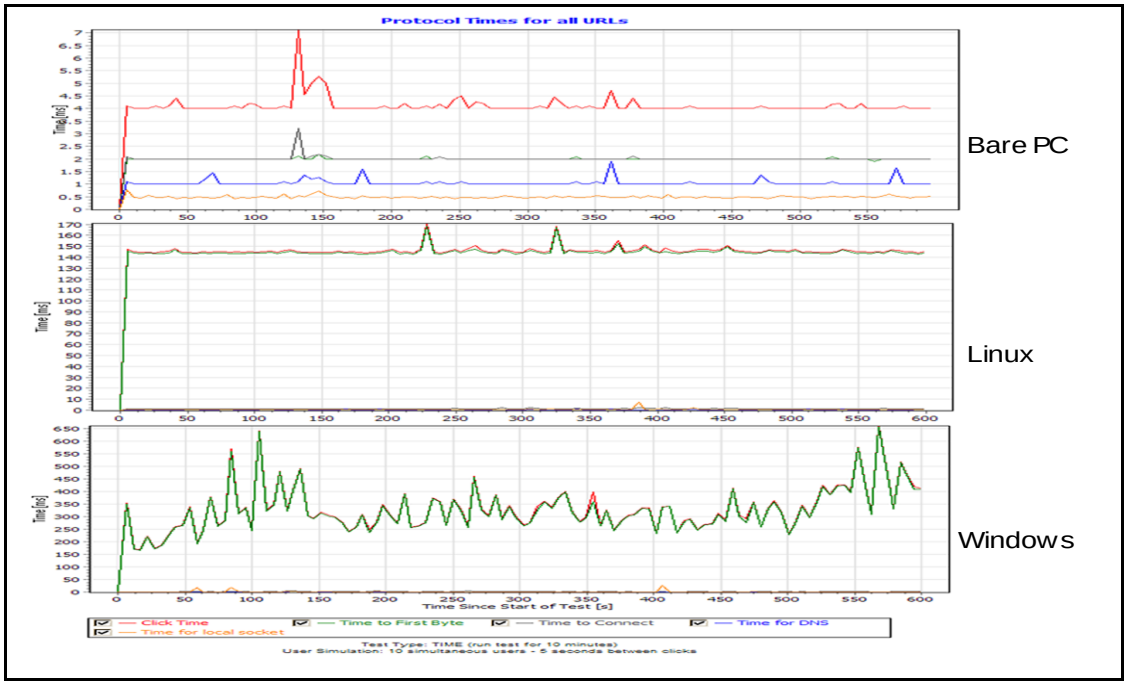

Figure 51. Stress Tool Graph for Protocol Time

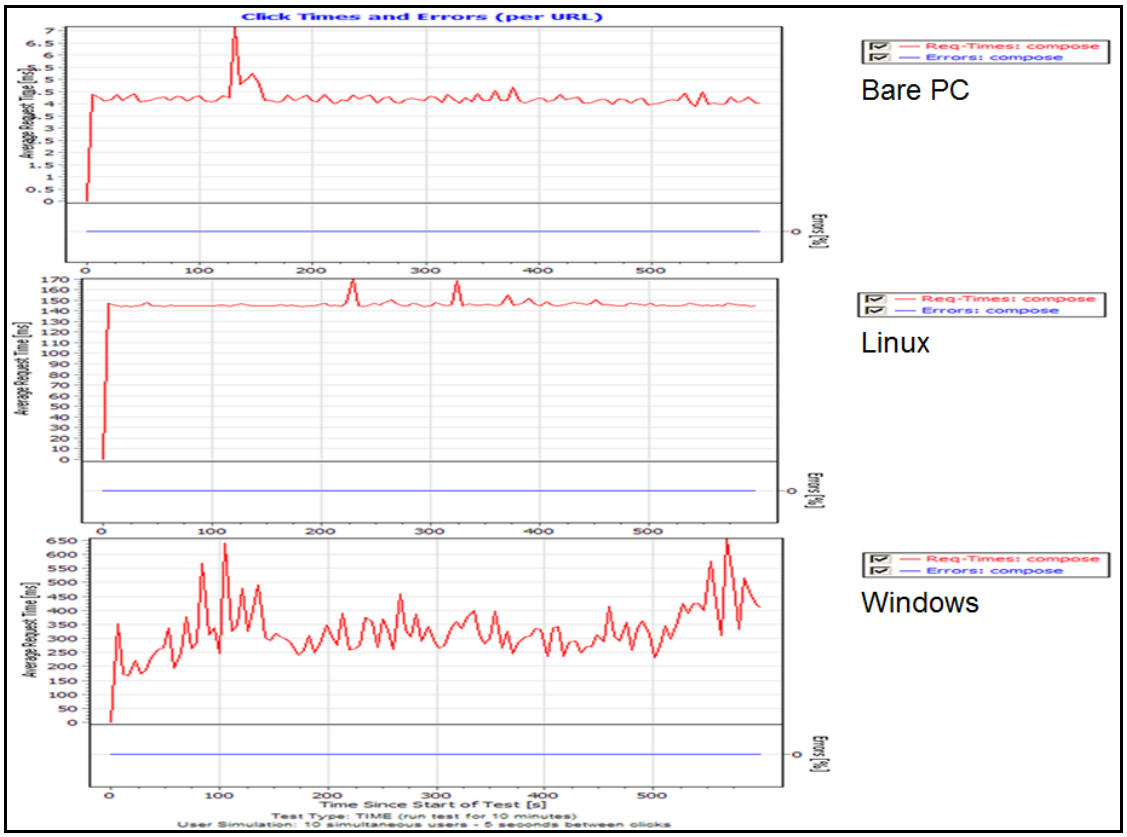

Figure 52. Stress Tool Graph for Click Time

Figures 49-52 show the stress tool graphs for the stress test conducted on the three servers and present in figures 44-48. These graphs validate the discussions and results presented. It can be seen in these graphs that the bare PC server has better click time (single request process time), CPU load distribution, protocol processing time, server and user bandwidth than the Windows and Linux based servers. The time variations are almost twice lower than the OS-based servers. This affirms bare PC's stand as a high performance platform.

	Bare PC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
Login Processing Time(ms)	2.767	185.224	87.296
Avg. Compose Processing Time(ms)	101.162	587.085	1330.250
Avg. Imbox Processing Time(ms)	6.005	295.503	219.145
Avg. Read Message Processing Time(ms)	32.413	530.254	748.915
Avg. Response Time(ms)	0.200	8981.827	89.336
Avg. Throughput	16.315	2.306	0.454

Table 1. Initial Test Performance Summary

The overall summary of the performance measurements are shown in Table 1. Notice that the Bare PC Webmail server outperforms the other servers in response time, processing time, and throughput. The Bare PC Webmail server's response time is 44909 times better than the Windows Webmail server and 446 times better than the Linux Webmail server. The average processing time for the Bare PC Webmail server is 36 times better than the Windows server and 29 times better than the Linux server. The throughput for Bare PC is 8 times better than Windows and 32 times better than Linux.

B. Further Performance Comparison with other Servers

1) LAN Measurements

- *Experimental Setup*

Further LAN and WAN performance studies were conducted on the bare PC webmail server and other conventional based webmail servers. For the LAN studies, a dedicated test network consisting of five Ethernet switches (S1-S5) interconnected linearly by four Linux routers (R1-R4) was set up. The client (C) and Webmail server (WMS) were connected to the ends of the network so that messages between the client and Webmail server are routed along the following path:

C--S1--R1--S2--R2--S3--R3--S4--R4--S5—WMS, as shown in figure 53.

All switches were gigabit switches except for the 100Mbps switch (S1) used to connect the client to the network. The clients ran Windows XP and the OS-based Webmail servers ran Windows XP or Linux (CentOS). All machines were Dell OptiPlex GX520s. OS-based Webmail server details are as follows: Afterlogic MailSuite Pro (Linux), MailTraq Server (XP), Atmail Server 6.20.3 (Linux), Icewarp Server 10.2.1 (XP), and Hexamail Server 4.0.1.002 (XP).

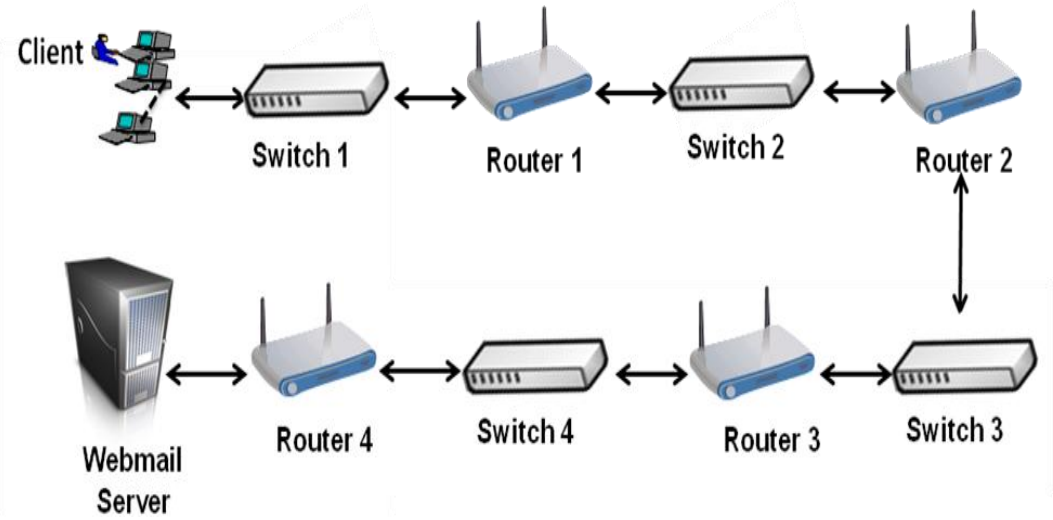

Figure 53. Experimental Setup for Routed Network Test

- LAN Results

Figure 54 is derived from the Wireshark timestamps for each message in the sequence of messages exchanged during a login GET request. The difference in timestamps for a pair of consecutive messages such as (GET, ACK) or (Data, 200_OK) gives the delay between the pair. As expected, the performance for all servers during the initial TCP handshake is the same. There is a rise between the client GET request and the server ACK due to the server delay in processing the request. All servers show little variation in processing time for subsequent message pairs.

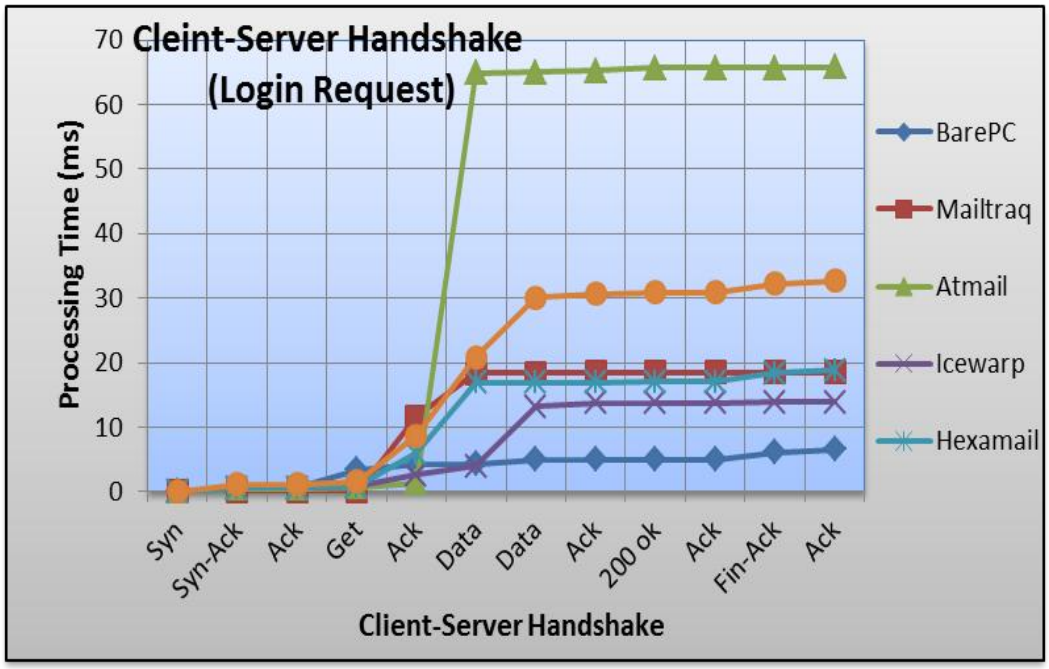

Figure 54. Login Get Request Message Times

Similarly, Figure 55 compares the processing time for a login POST request. The (POST, ACK) behavior for all servers except for Hexamail and MailTraq is the same. The (ACK, 302_Found) delay is visible for all servers except MailTraq. The Bare PC server processing times for both GET and POST requests are minimal. Since different servers may do the work to process the requests during different steps, only the overall processing time should be compared. Figure 56 shows the processing time for compose with varying message sizes. The varying behavior of the servers reflects the combination of TCP, HTTP and the mail server application. Hexamail has stable behavior for large message sizes, while Icewarp shows the most variation. The Bare PC server has the highest processing delay for a message of 10,000 bytes, but shows a general reduction for larger sizes except for a small rise at 20,000 bytes. Figure 57 shows the processing time

for receiving an inbox with 6 messages. While all servers complete processing in about 1.1 milliseconds on the average, the Bare PC server requires less than 0.1 ms.

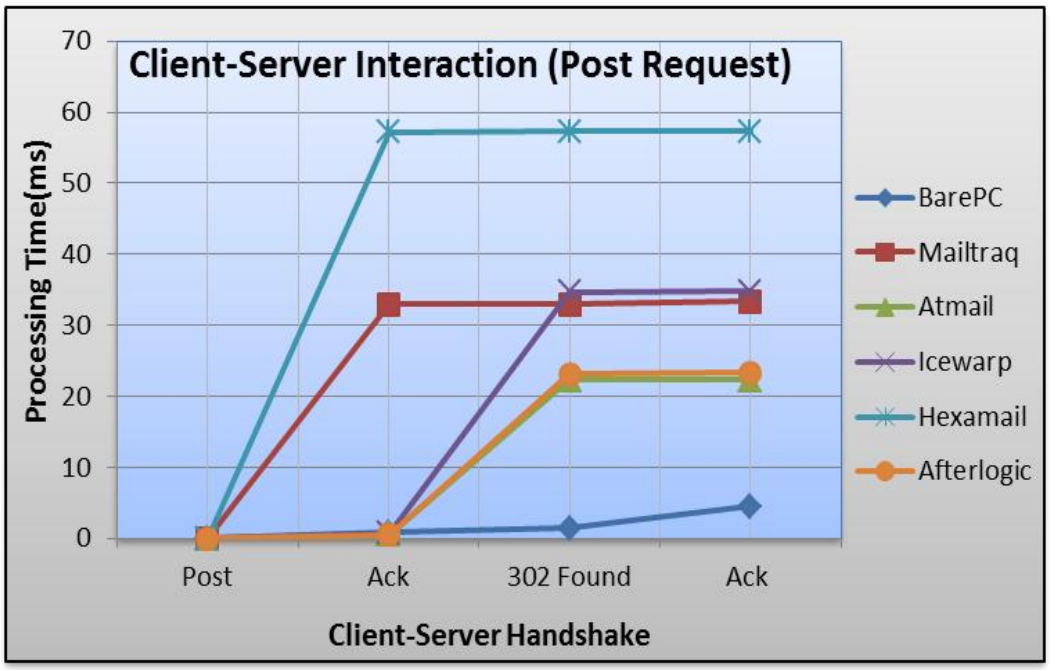

Figure 55. Login Post Request Message Times

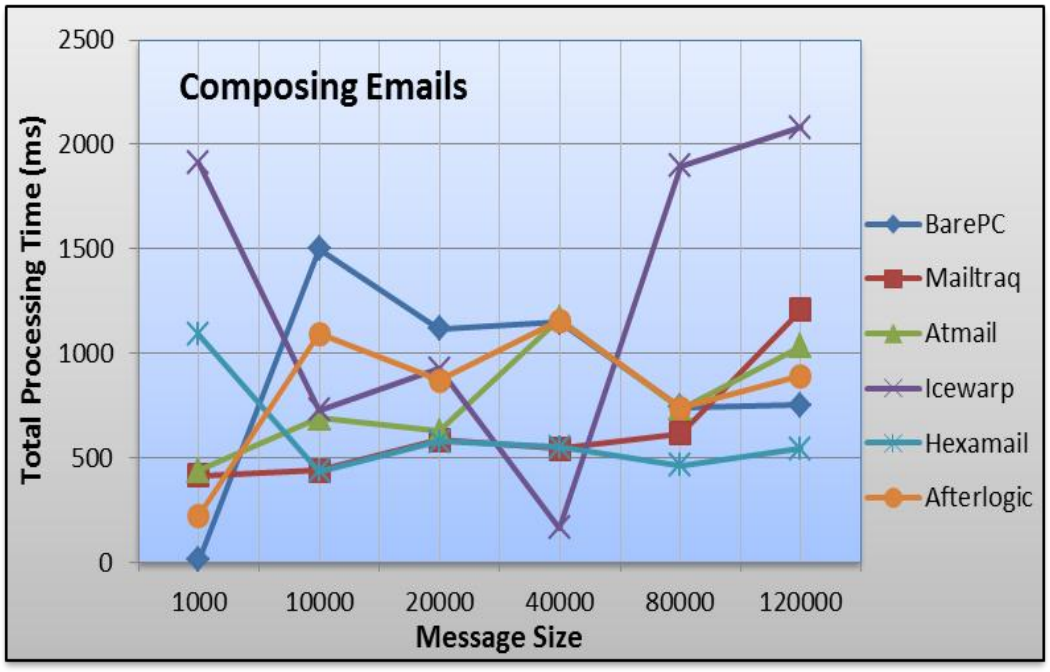

Figure 56. Processing Time for Compose (Varying Message Sizes)

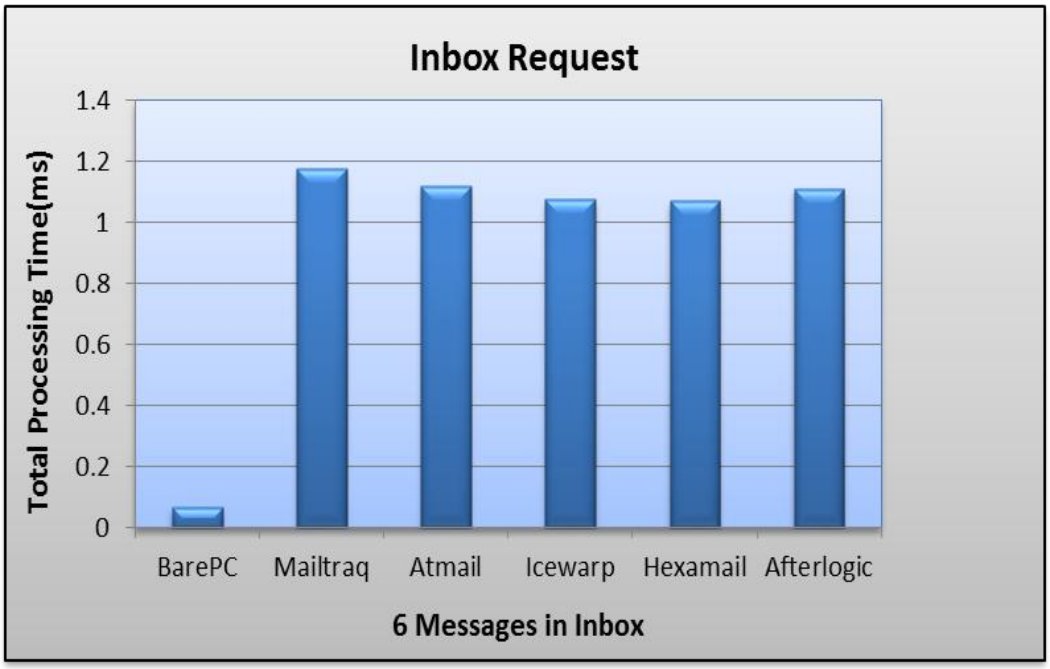

Figure 57. Processing Time for an Inbox Request (6 Messages)

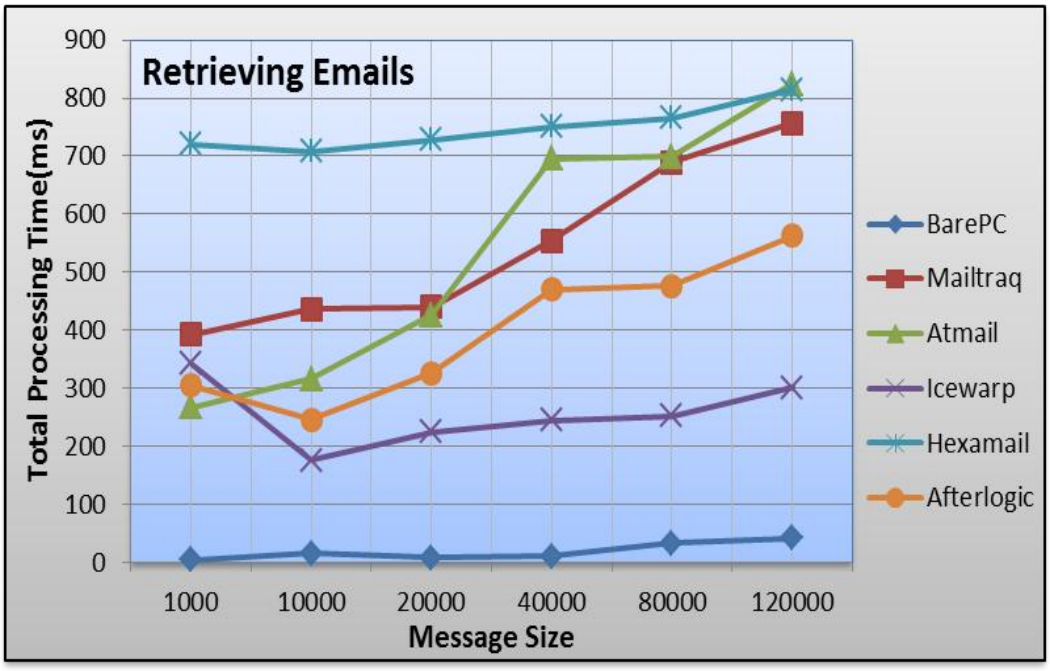

Figure 58. Processing Time for Read (Varying Message Sizes)

Figure 58 shows the processing time to retrieve messages of sizes 1000-120,000 bytes. Processing time for the Bare PC is minimal, and it has the smallest increase in

processing time followed by Icewarp and Hexamail show the smallest increase in processing time. If a message is already retrieved into an inbox, it takes less time to process and transmit the message to the client. Figure 59 shows the throughput measured during compose for increasing message sizes. The Bare PC server throughput is highest and approximately twice the throughput of the best OS-based server Afterlogic. The low throughput of Icewarp reflects its large processing time in Figure 56.

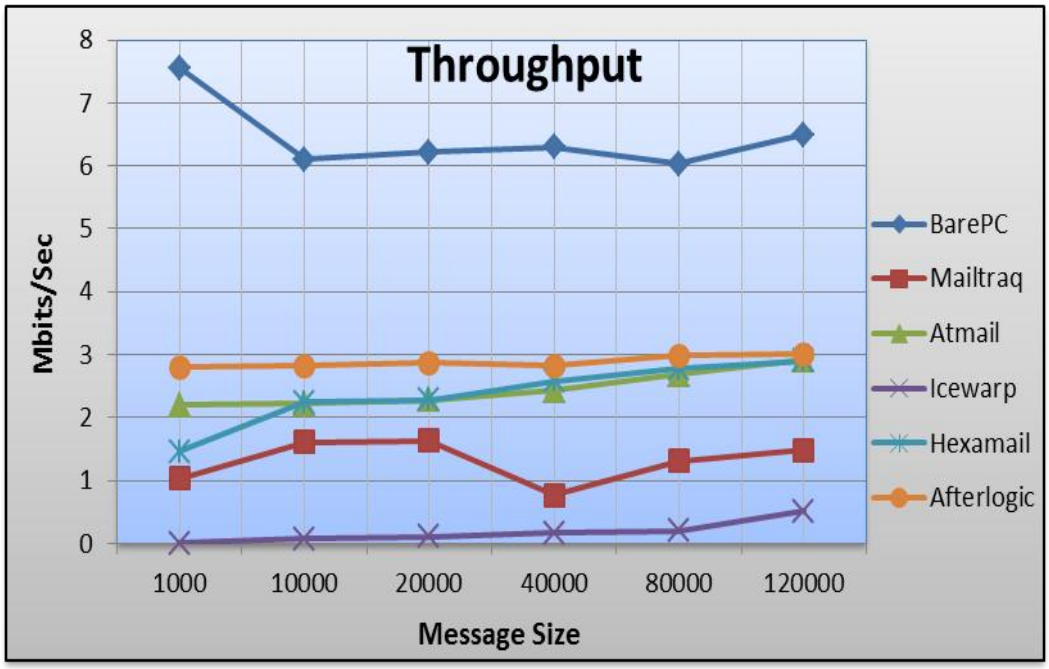

Figure 59. Throughput for Compose (Varying Message Sizes)

Further performance tests were conducted on the Webmail servers using the stress tool [56]. The tool was used to increase the number of users from 1 to 10 and determine the resulting impact on performance. Each test was run for 10 minutes and each user makes 100 requests/s. Figure 60 illustrates the variation of server CPU utilization over time for a maximum of 10 users. The average CPU utilization of the Linux-based Afterlogic and Atmail servers and the Bare PC server is less than 4%, while that of the

Windows-based Mailtraq, Icewarp and Hexamail servers is between 8-12%. It is evident that more CPU processing is required by the Windows-based servers when processing concurrent requests. The figure also indicates that the CPU utilization of the Bare PC server shows some slight initial variability compared to that of the Atmail server.

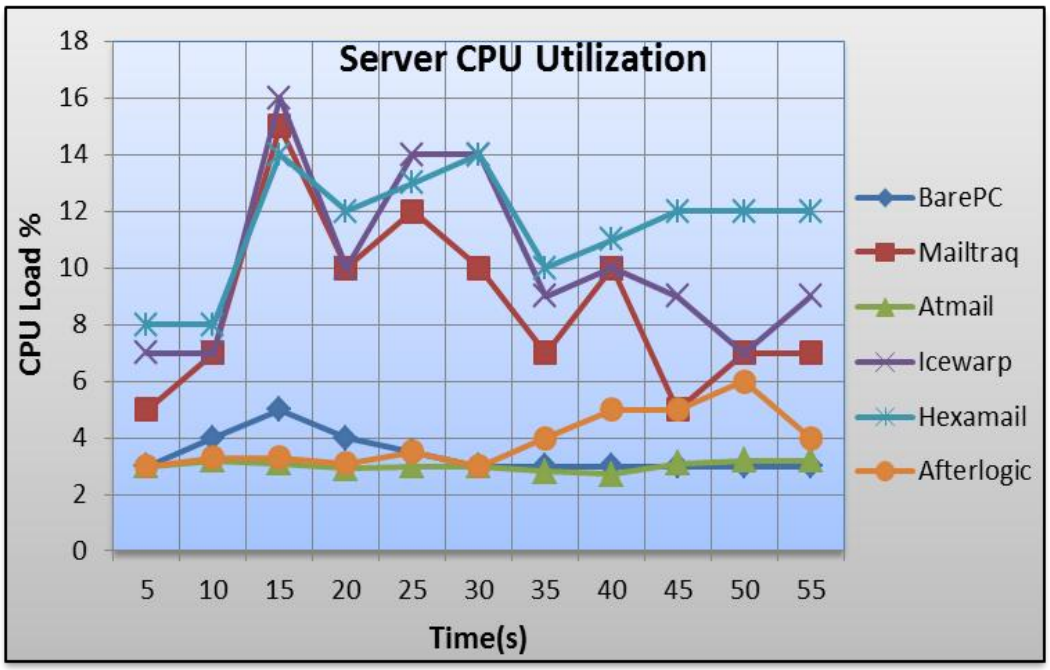

Figure 60. CPU Utilization for 10 Users

Figure 61 shows the variation of server bandwidth over time for 10 users. It can be seen that the bandwidth of all servers is relatively stable after the initial increase during the first 5 seconds. However, while there is little difference between the bandwidth of the OS-based servers (average < 60 kbps, maximum <700 kbps), the bandwidth of the Bare PC server is significantly higher (average and maximum exceed 100 kbps and 12 Mbps respectively).

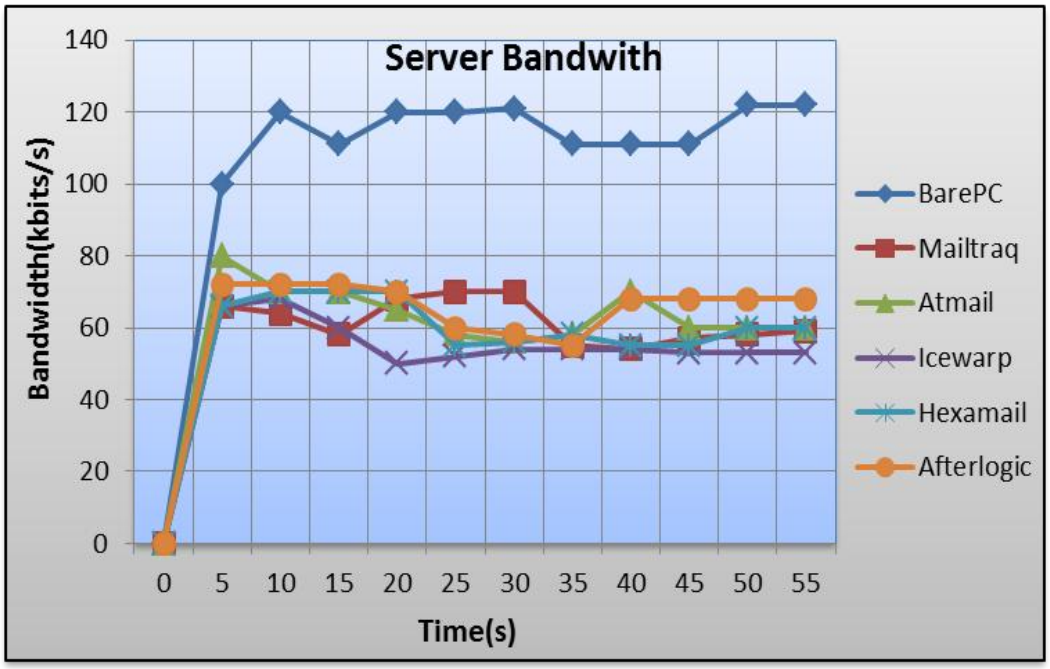

Figure 61. Bandwidth Variation for 10 Users

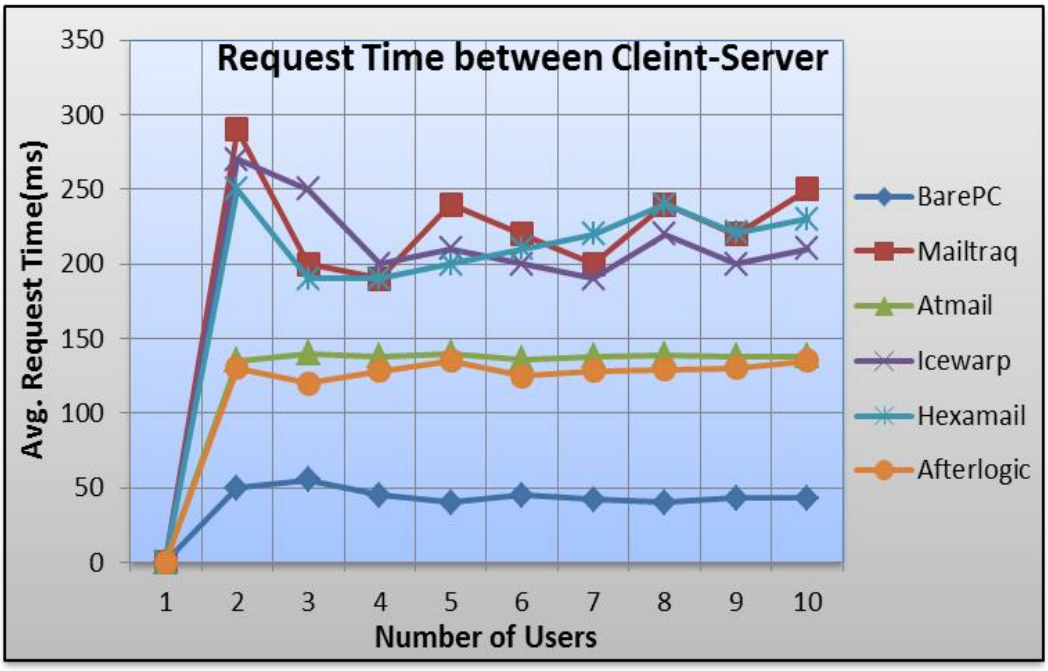

Figure 62. Post Request Completion Time

Figure 62 is the average time (delay) to complete a POST request for a varying number of users. Even two users cause the delay to increase significantly compared to one user, but the delay stabilizes for three or more users. The Bare PC has the least delay, while the Windows-based servers have the highest delay in this case. It can be seen that the delays for the Bare PC server and Linux-based servers differ by almost 100 ms.

Figure 63 show the amount of time a user waits for the server to establish a connection in the presence of multiple users. The Linux and Bare PC servers perform much better than the Windows servers, but the performance advantage of the Bare PC server compared to the Linux servers is reduced since the performance of the latter improves significantly when there are 6-10 users.

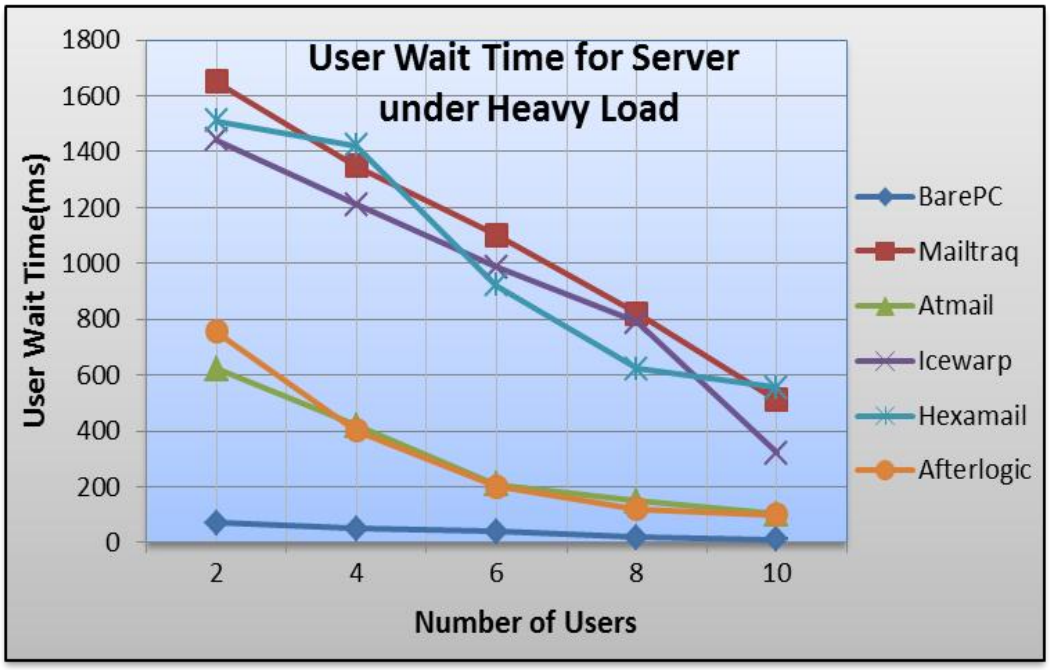

Figure 63. User Wait Time (Increasing Number of Users)

Figures 64 and 65 show respectively the Webmail server processing times for a read request and the throughput for a compose request with and without stress. To create

stress, the tool is used to generate100 concurrent requests/s from 10 users and an additional client is used to generate the read or compose request involving an email message of 120,000 bytes. Although performance degrades under stress for all servers as expected, the Bare PC server's performance with and without stress is significantly better than the performance of the OS-based servers in both cases. Moreover, regardless of whether they are under stress or not, OS-based servers with higher throughput have larger processing times and vice-versa.

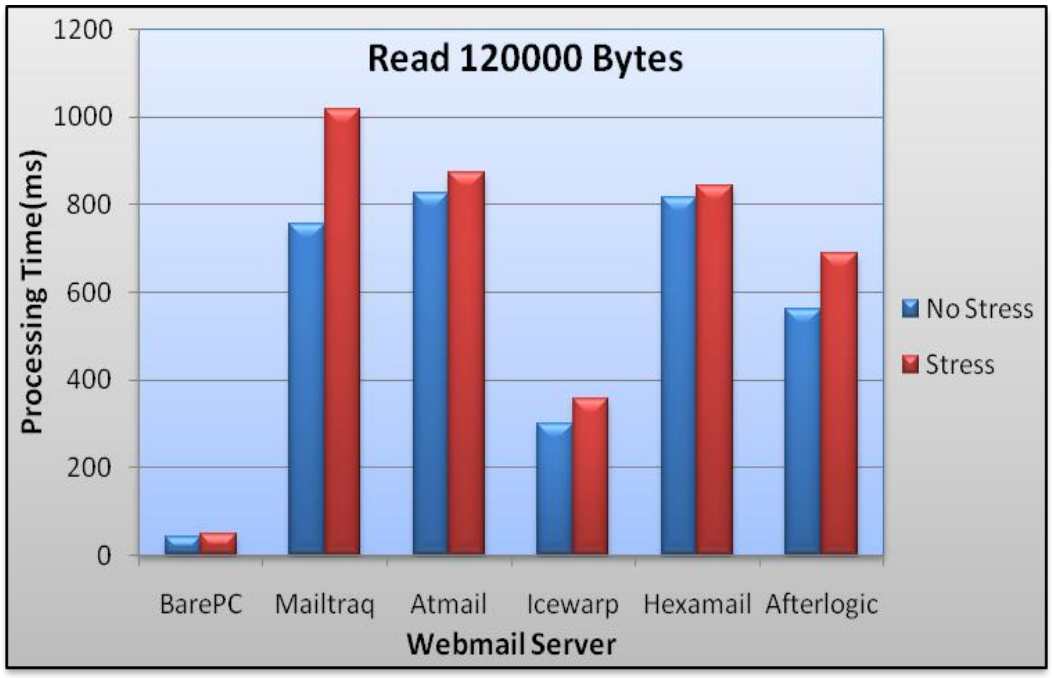

Figure 64. Message Read Time (120000-Byte Message)

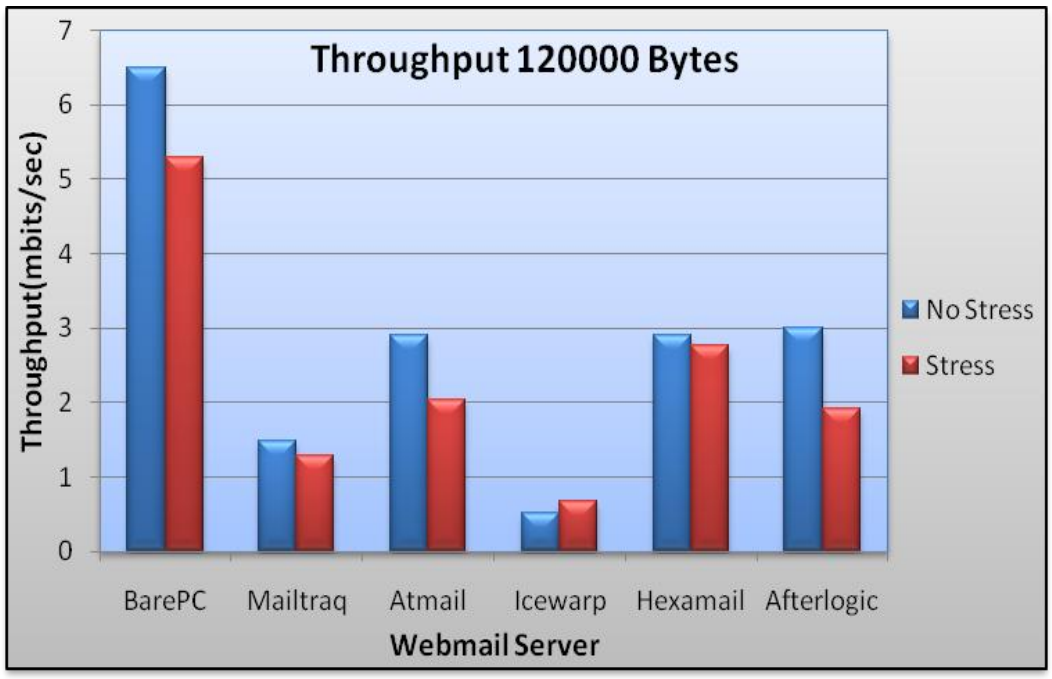

Figure 65. Throughput for Compose (120000-Byte Message)

2) WAN Measurements

- *Experimental Setup*

For the WAN tests, an Internet connection was established between each Webmail server and a client PC located approximately 50 miles away with about 30 hops between the two destinations. To ensure consistency, tests were performed during a contiguous time and repeated several times to ensure that the results were stable and independent of variable network conditions. As before, a Wireshark packet analyzer was used to capture the data. The machines used were the same as for the LAN studies.

- WAN Results

Figures 66 and 67 are derived from the Wireshark timestamps for each message in the sequence of messages exchanged over the WAN during a login GET or POST request (they correspond to Figures 54 and 55 for the LAN tests). As before, the difference

between cumulative processing times for a pair of consecutive messages such as GET-ACK for GET or POST-ACK for POST gives the delay between the pair. It can be seen that these delays for GET and POST requests are significantly less for the Bare PC server than for the OS-based servers.

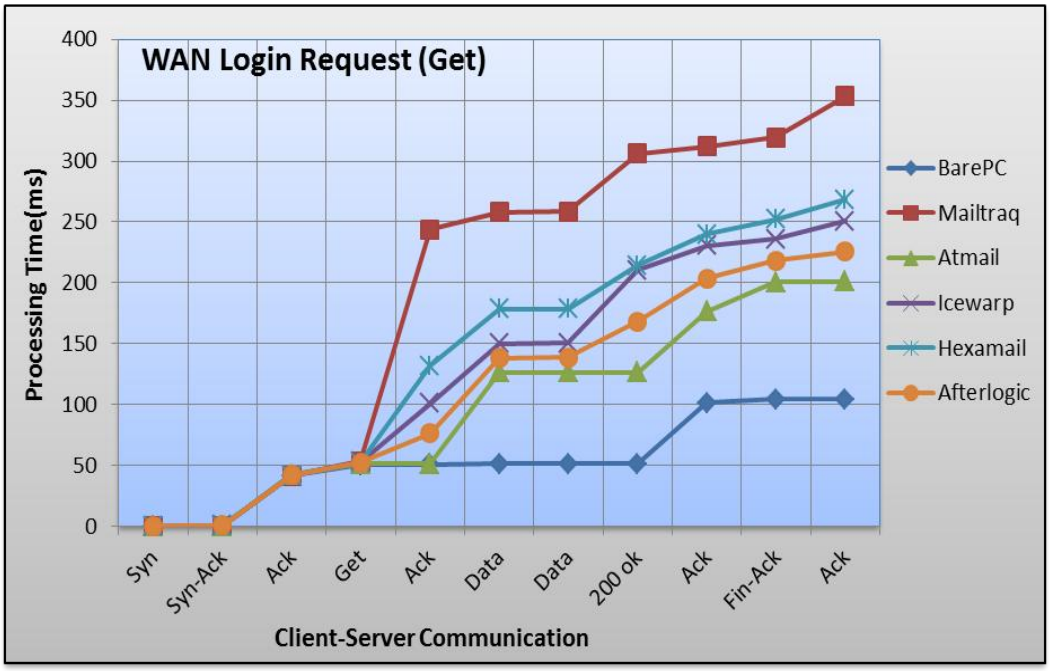

Figure 66. Processing Time for Login Get Request

A closer examination of Figures 66 and 67 reveals that the GET and POST delays for the OS-based servers vary considerably across message pairs. For example in Figure 66, MailTraq has the highest GET-to-ACK time, Atmail has the highest ACK-to-Data time, and Icewarp has the highest Data-to-200_OK time. However, Atmail has the lowest GET-to-ACK and Data-to-200_OK times after the Bare PC. Similarly, compared to the Windows servers, Afterlogic has lower GET-to-ACK and Data-to-200_OK times but a higher ACK-to-Data time. In case of a login POST request (Figure 67), it can be seen that MailTraq and Atmail have respectively the highest and lowest (next to the Bare PC)

POST-to-ACK time, whereas Atmail has the highest and MailTraq has the lowest (next to the Bare PC and Icewarp) ACK-to-302_Found time.

Figure 68 shows the processing time over the WAN for compose, with message sizes varying from 1000 to 120,000 bytes. Processing time increases in an approximately linear manner as the message size increases. This was not the case for the corresponding LAN result in Figure 56. However, it can be seen that the processing time for all servers increases at a higher rate for message sizes from 40,000-120,000 bytes.

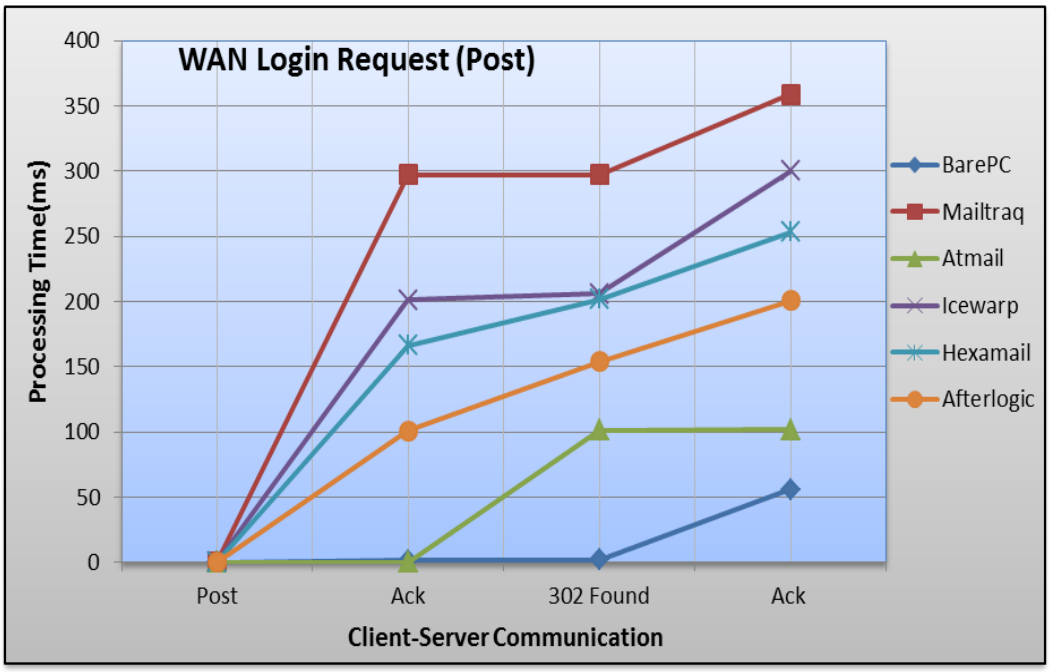

Figure 67. Processing Time for Login Post Request

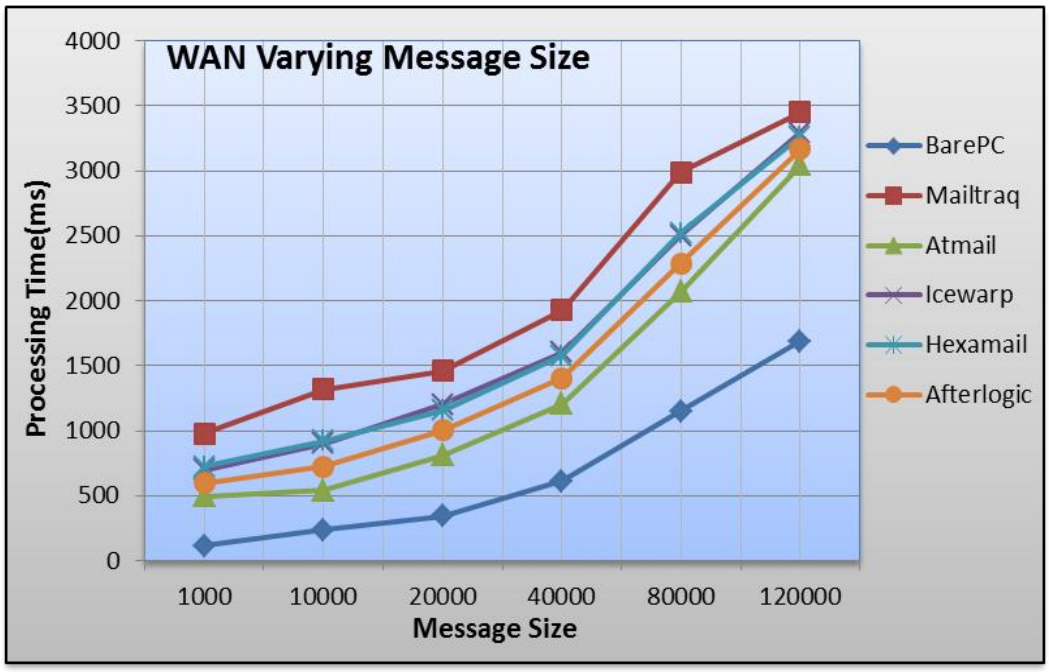

Figure 68. Processing Time for Compose (Varying Message Sizes)

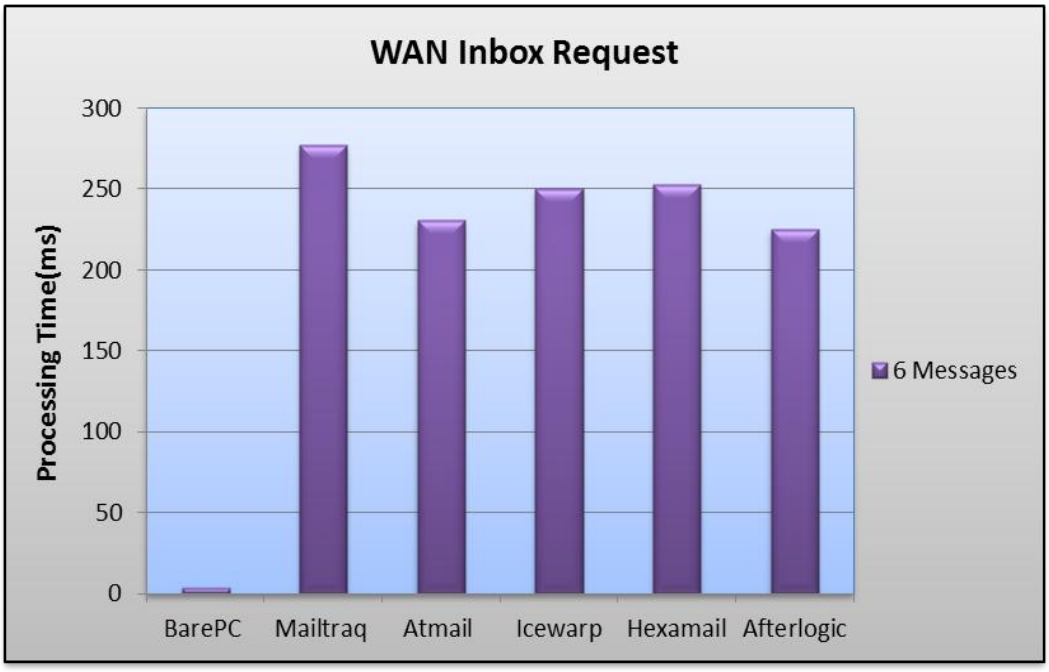

Figure 69. Processing Time for an Inbox Request (6 Messages)

Figure 69 shows the processing time over the WAN for receiving an inbox containing 6 messages. The Bare PC receives the inbox in 0.392 milliseconds, while the other

servers require an average time of about 230 milliseconds. Figure 70 shows the processing time on the WAN for reading individual emails of varying message sizes. The processing time for the Bare PC server is stable up to 40,000 bytes and increases slowly thereafter for larger messages. The processing times on the other servers are stable up to 20,000 bytes, but rise sharply to 1400 milliseconds.

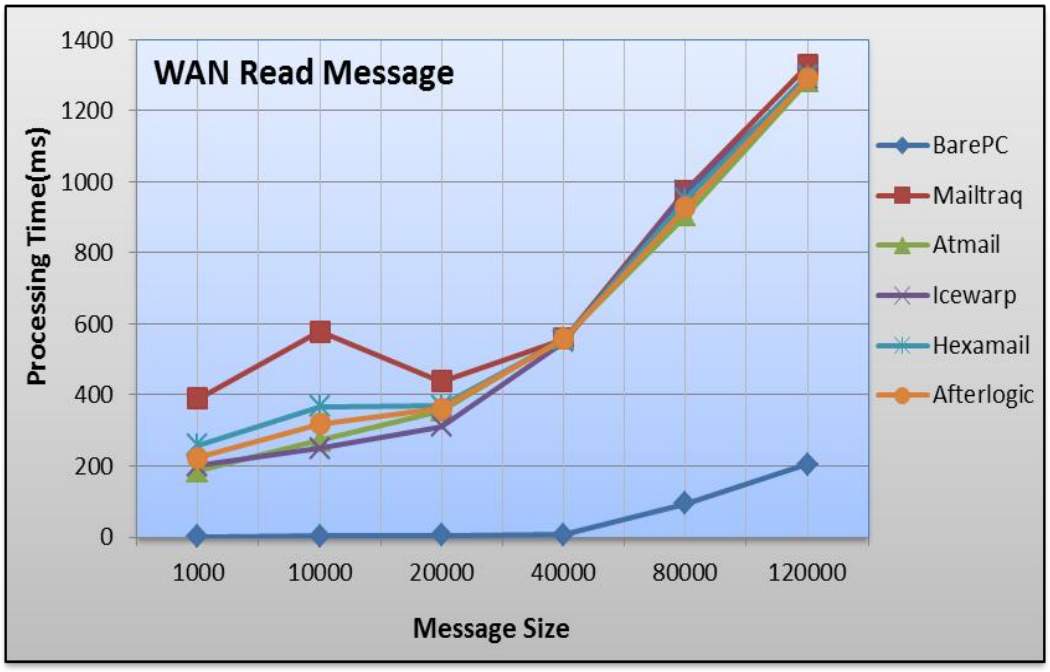

Figure 70. Processing Time for Read (Varying Message Sizes)

The throughput for varying message sizes was also captured during the Internet test and compared in Figure 71. The throughput for all servers is on the average 1.28 times better than the throughput of the Atmail server, whose performance in general on the previous tests was better than the other OS-based servers.

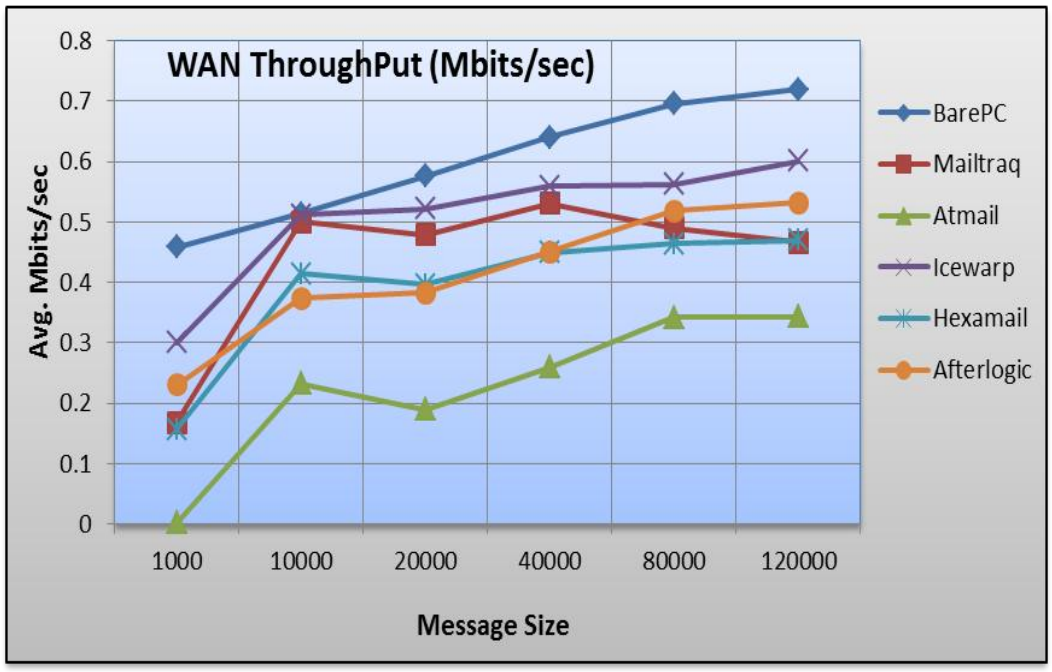

Figure 71. Average Throughput (Varying Message Sizes)

C. TLS Measurements

- *Experimental setup*

For the experiments, a 100-Mbps Ethernet test LAN with servers and a client running on ordinary Dell OptiPlex GX520 PCs (3.2GHz Intel Pentium 4 Processor with 1GB RAM) was set up. In addition to the TLS and non-TLS bare PC Webmail servers, the OS-based TLS Webmail servers were Atmail on Linux, and Icewarp on Windows XP. The client was a Firefox browser on Windows XP.

- *Experimental Results*

This section discusses the results of experiments conducted using the test LAN. The goal of the experiments was to compare the basic performance of the bare PC and OS-based TLS Webmail servers, and to determine overhead due to TLS on the bare PC Webmail server. TLS processing includes decrypting incoming requests, encrypting

outgoing responses, and verifying/computing the message authentication code, but excludes the one-time TLS handshake. Each experiment was repeated several times to ensure the results were consistent. Figures 72 and 73 respectively show the processing time for a single Get and Post (Application Data) request sent to the bare PC and Atmail servers (processing times for the Windows server are omitted since they were much larger). In Figure 72, the processing time for the Atmail server spikes between the client's Application data (GET) request and the server's ACK. This spike corresponds to the processing of the request, which includes decryption of data by the server. The processing time for the bare PC Webmail server does not spike and is approximately linear.

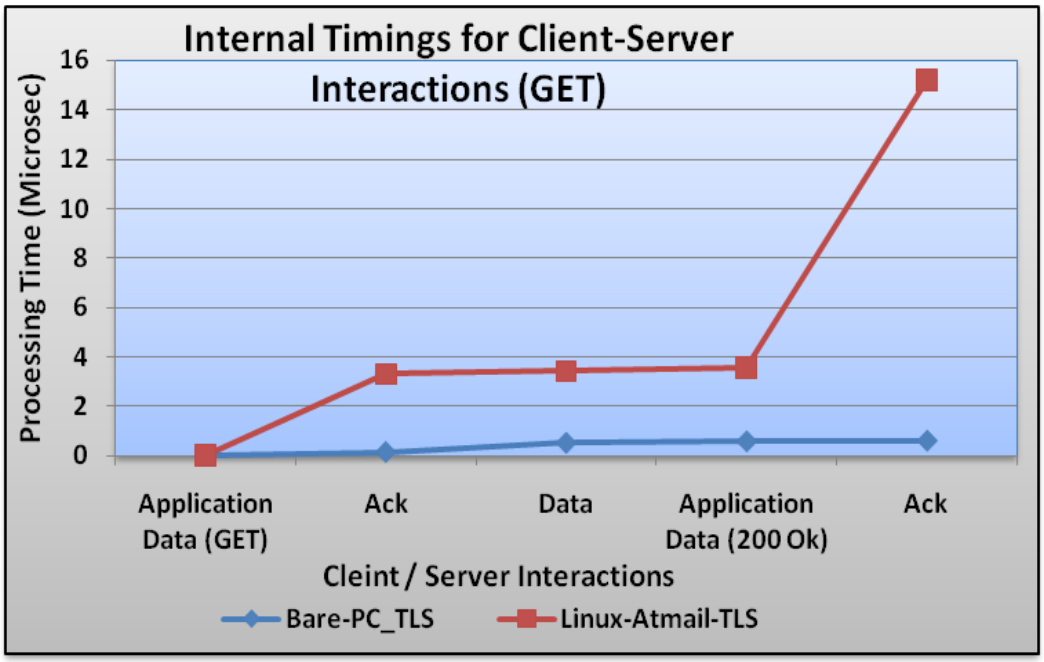

Figure 72. GET Request

In Figure 73, the processing time for the POST request is shown. The processing time for the Atmail server now spikes between servers' Application Data (302 found) and

ACK response, but that for the bare PC Webmail server shows only a small increase. The better performance of the latter reflects reduced overhead due to OS absence and protocol intertwining.

Figure 74 shows the total time (in milliseconds) required when composing an email message with message sizes varying from 1KB to 16 KB. 16K was used to conduct the test due to TLS limitation on bare PC. The current bare TLS architecture allows encryption and decryption of file sizes that are 16K or less. The processing time for the bare PC Webmail server is small and approximately the same for all message sizes. In contrast, the processing times for the Atmail (Linux) and Icewarp (Windows) servers are larger with a maximum for 6KB messages.

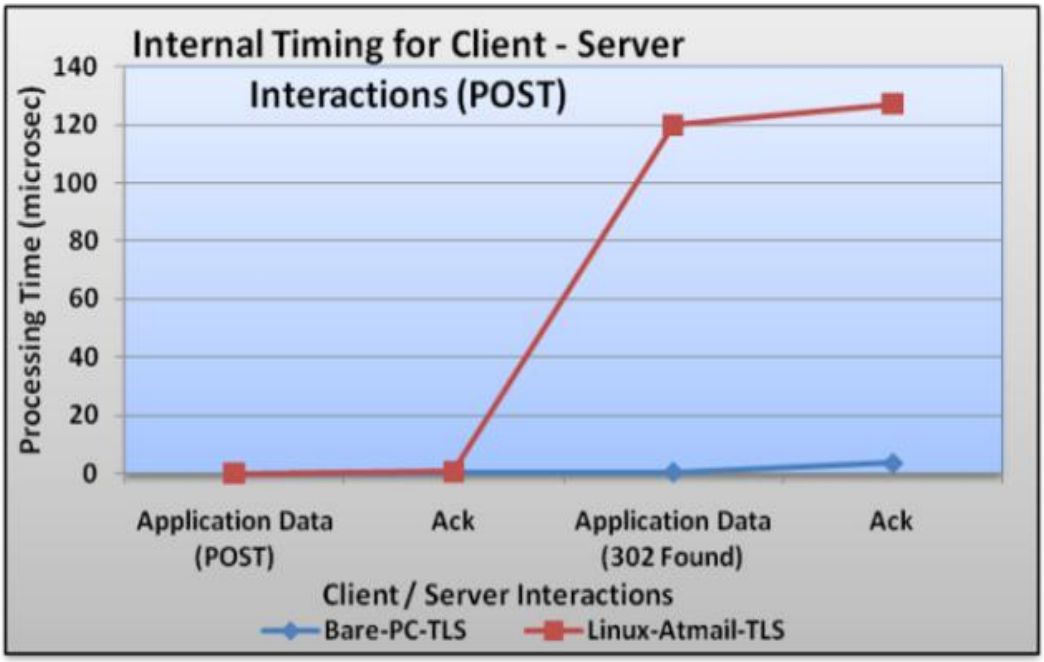

Figure 73. POST Request

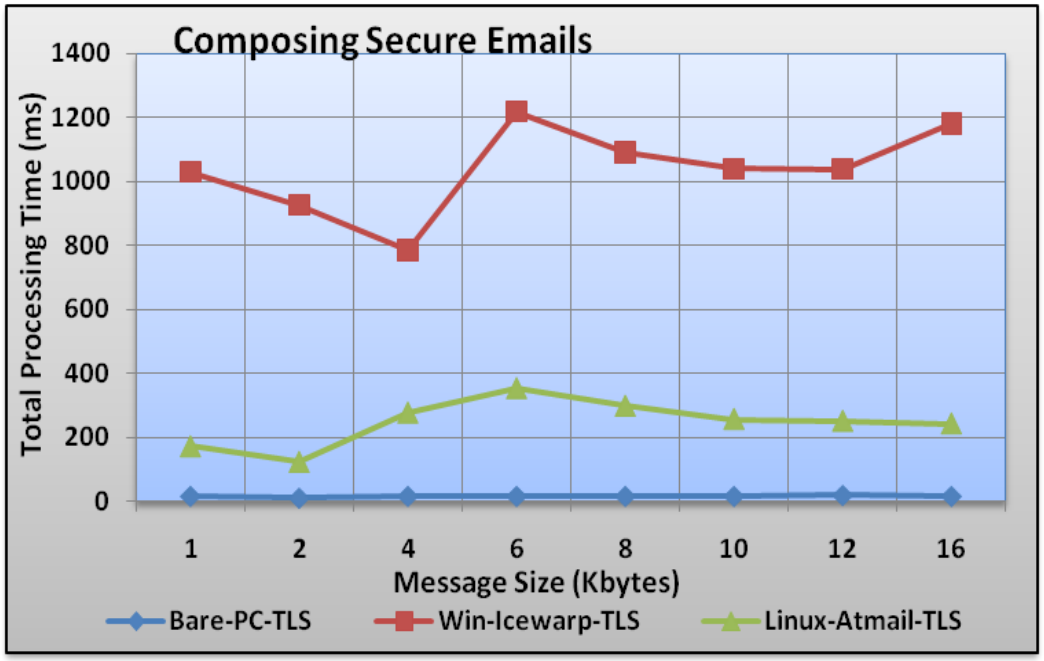

Figure 74. Compose Email Messages

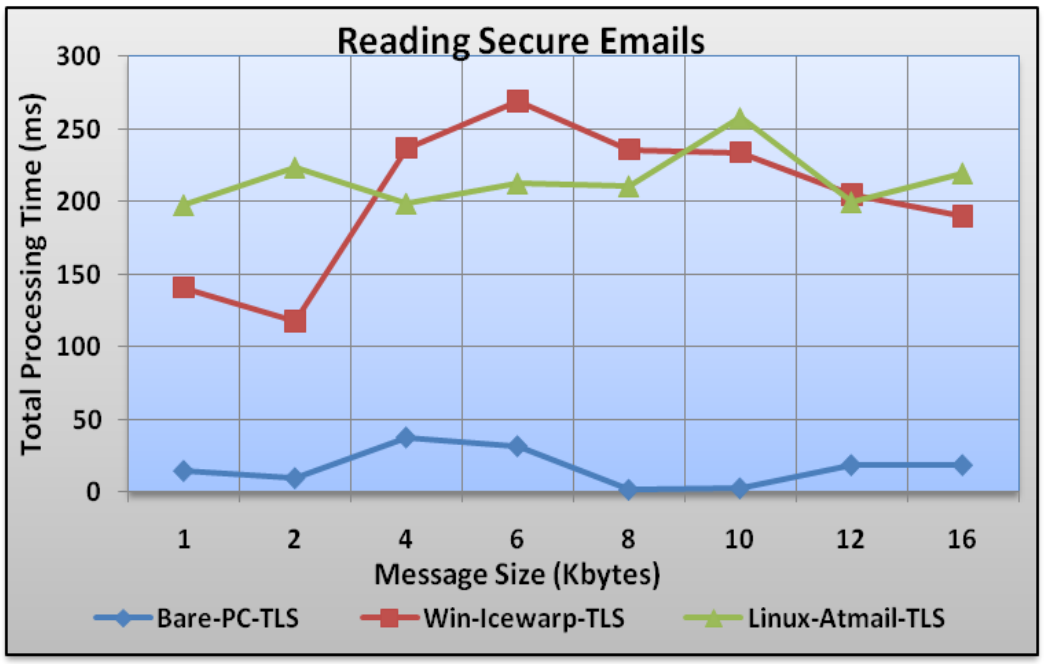

Figure 75. Retrieval Email Messages

Figure 75 shows the total processing time when reading email messages with message sizes varying from 1KB to 16KB. The processing time in this case includes the time to

retrieve a message from the user inbox and display it on the screen. In this case, all servers show an increase in processing time for certain message sizes. However, the processing time for the bare PC Webmail server never exceeds 50 ms, whereas those for the OS-based servers can exceed 200 ms. Figure 76 shows the total time each server spent to process an inbox request that involves 10 email messages. The bare PC Webmail server processes the request in a total time of 42 milliseconds; this is approximately 7 times faster than Icewarp (total time 291 milliseconds), and approximately 6 times faster than Atmail (total time 238 milliseconds).

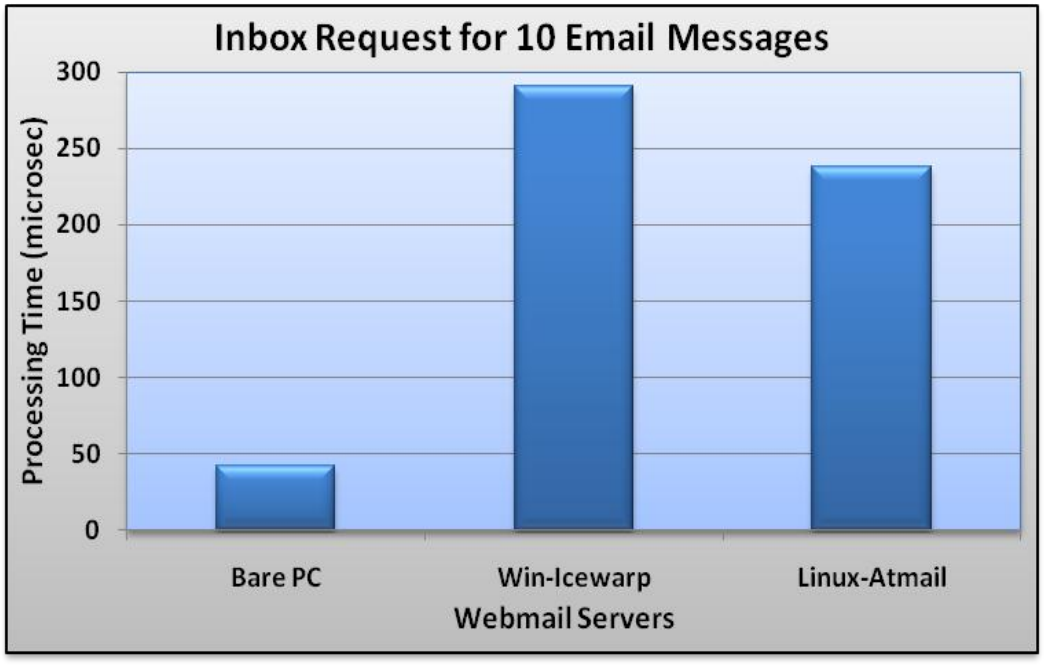

Figure 76. Inbox Request for 10 Email Messages

The performance of the TLS-capable Webmail server with the non-TLS bare PC Webmail server were compared by measuring the processing times for various email transactions. Since the TLS-capable server is an adaptation of the non-TLS server, the

performance differences reflect the overhead due to TLS processing of Webmail messages.

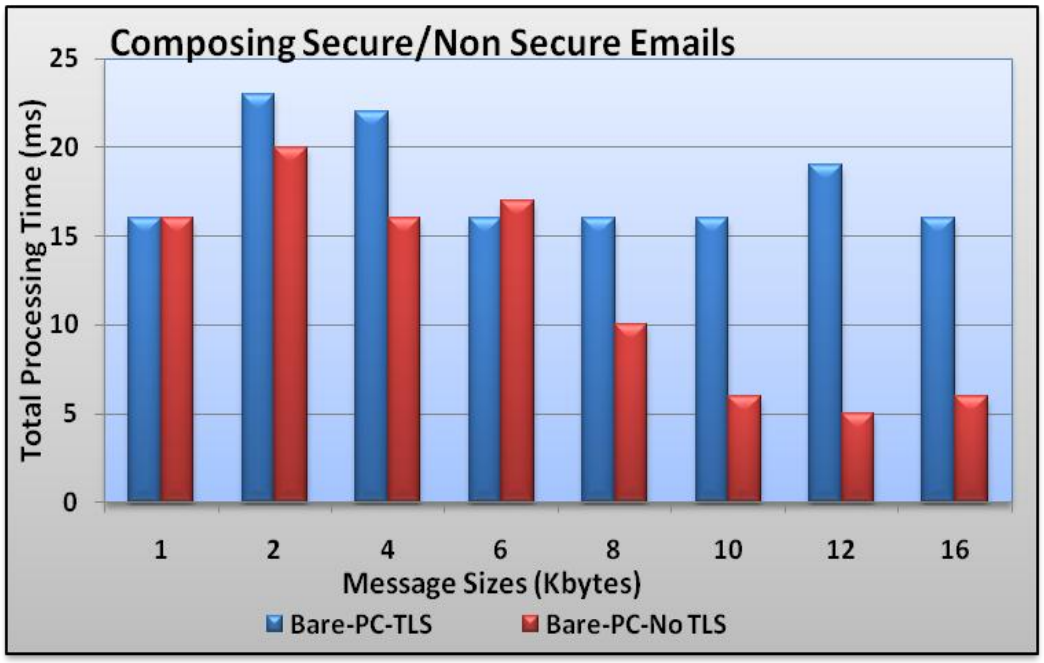

Figure 77. Compose with and without TLS

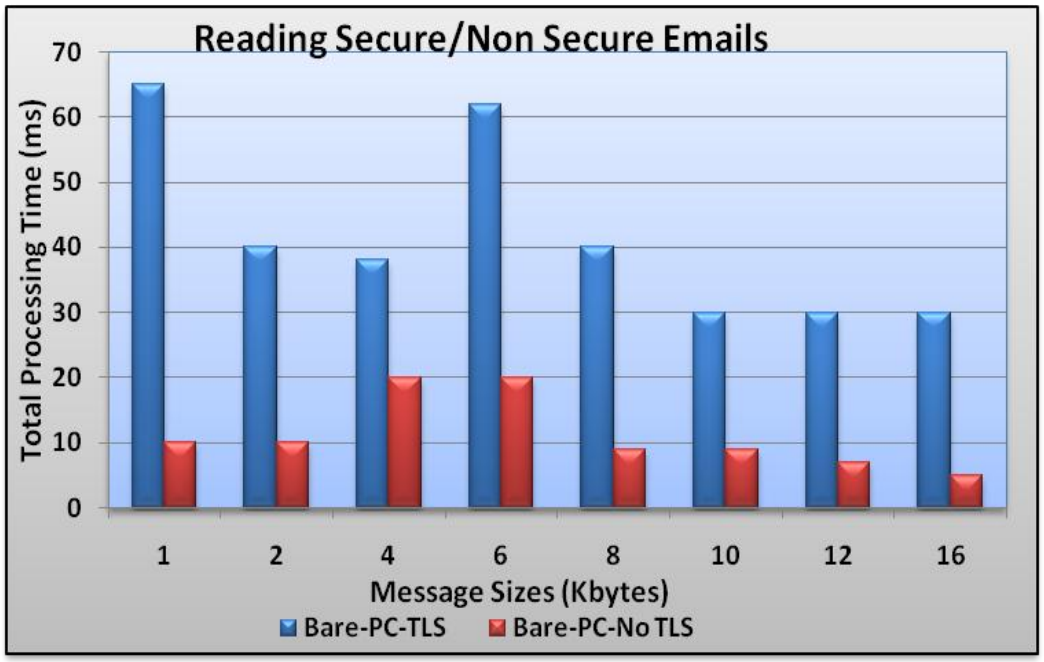

Figure 78. Retrieval with and without TLS

Figure 77 shows the total processing time when composing email messages of various message sizes. TLS increases the processing time by at least 5 ms for most message sizes, with a doubling of processing time for message sizes that are 10 KB or larger. For a 6-KB message, the processing time for the non-TLS server is slightly larger. More detailed timing studies are needed to explain this behavior.

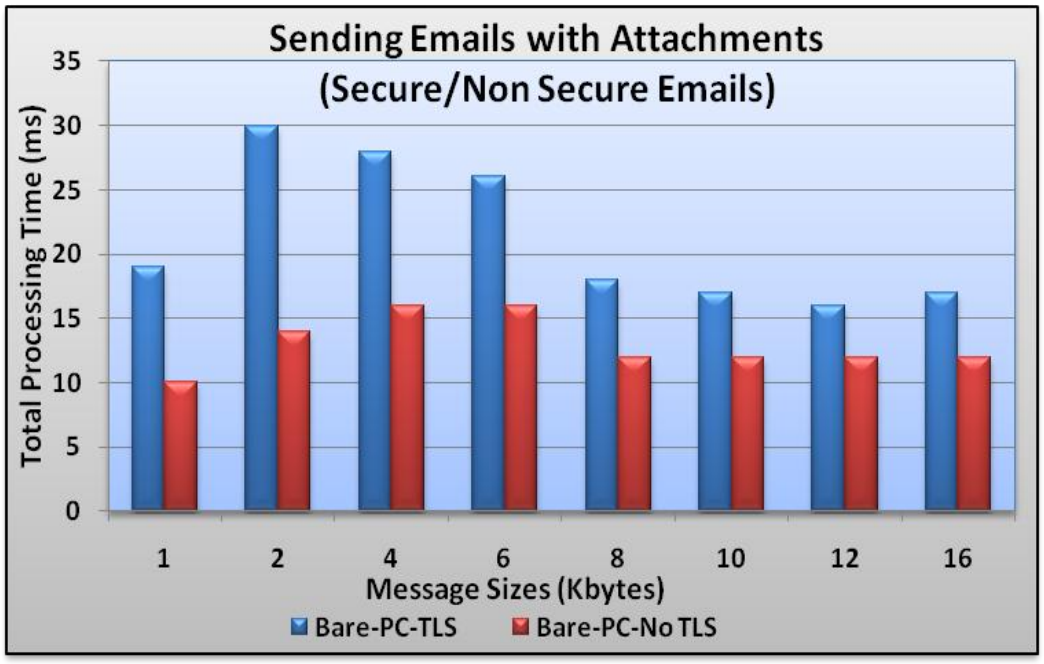

Figure 79. Sending Attachments with and without TLS

Figure 78 shows the processing time when retrieving email messages with varying message sizes. There is at least a three-fold increase in processing time due to TLS for most message sizes. Figure 79 shows the processing times when sending email messages of varying sizes with an attachment. The increase in processing time due to TLS is smaller for message sizes that are 8 KB or larger.

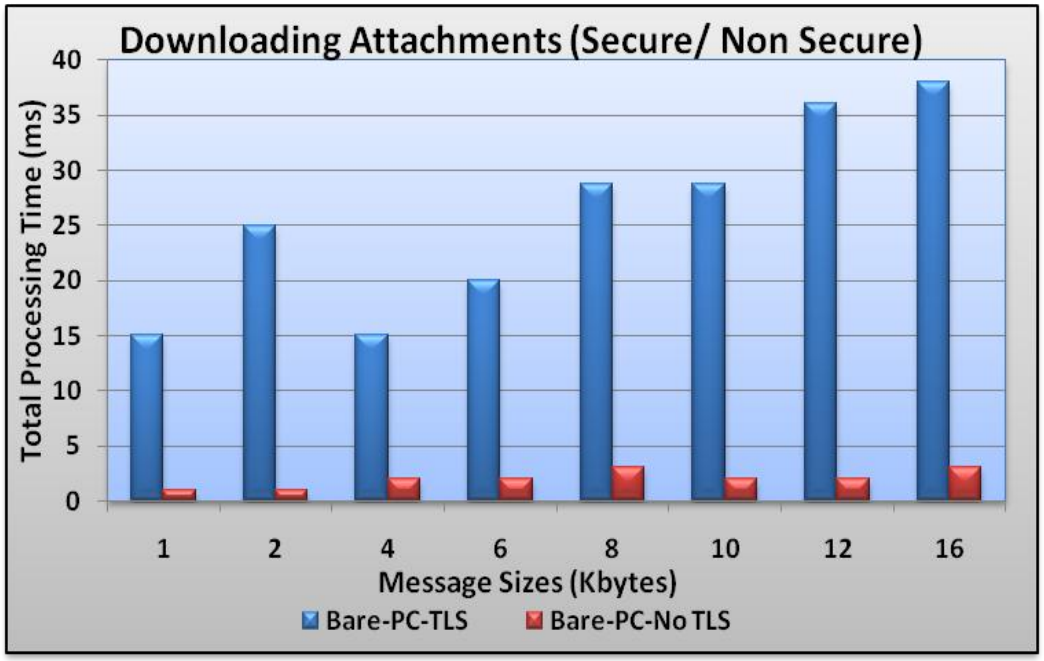

Figure 80. Retrieving Attachments with and without TLS

Figure 80 shows the processing time when retrieving email messages of varying sizes with an attachment. The increase in processing time due to TLS is now larger for all message sizes. Collectively, these results indicate that both TLS and OS overhead can significantly impact Webmail server performance.

VIII. NOVELTY AND SIGNIFICANT CONTRIBUTIONS

Significant contributions that this study will provide will be insight into the ability to create complex application objects, which uses many system resources and multiple concurrent user accesses, for efficient and timely throughput processing using the bare PC architecture. Lessons learned from the development and integration process, and analysis of results from test measurements and comparisons will provide new insights on how to develop future application objects of increasing complexity. This research will help identify the strengths and weaknesses of applying bare PC constructs as solutions for those applications which have a large set of functionalities, data sets, and user bases which they perform.

This research will also enhance the tool set of available application objects which are available for use in a suite of communication functions based on the bare PC platform. It serves as the milestone in building high-performance and pervasive systems based on the development of the Webmail Server which has a known, minimal, and efficient design and implementation architecture. This server will be available for future enhancements by other researchers who might want to perform further research in Webmail Servers and integrate new cutting edge features to the existing Webmail systems worldwide.

One novelty that this server presents is the integration of all the Webmail components (i.e. Email Server, Web Server, and Webmail Daemon or parser) into one single executable application. Thus making the AO more intelligent, faster, easy to run and use. One area that will be of interest is to investigate how to implement cloud computing on the current bare PC Webmail Server. This will allow multiple Webmail Servers to load balance their request and also to be able to handle multiple concurrent requests at the

same time. This will be an experiment in an attempt to implement chain authentication where one Webmail Server authenticates the user and passes the user credentials to other servers to serve the Email request. That is reducing SPAM emails or anonymous email injections into Webmail servers and reducing security threats in current Webmail Systems. If successful, this will be documented as a proposed change to the existing RFCs for Webmail standards.

The secure webmail server is an integration of the web server, email server, TLS services and a lean PHP parser and intertwining of the necessary protocols into a single monolithic server. Building such complex systems requires thorough understanding of the various server behaviors and operational functionalities. Lessons learned from building such a complex application could be the bases for building generic servers with multifunctional capabilities.

IX. CONCLUSION AND DISCUSSIONS

A. Conclusion

This thesis focused on the study of creating lean secure webmail servers on bare PC environments. The novel architectural features, design, implementation, design challenges and some performance results were discussed. Designing and implementing a complex and intriguing systems such as the lean secure webmail server on a bare PC environment pose some form of herculean task. This complexity could be aggravated if the server is made TLS-capable. Design characteristics such as tasking and intertwining of TLS, TCP and HTTP were leveraged to implement the bare server. An operational and functionality test performed on LAN/WAN environments were critically studied. The results of the test were matched with results of test performed on optimized OS-based servers that are Linux and Windows OS platforms. Comparison of the results purges the bare webmail server above the OS-based servers by an order of magnitude exceeding the other servers.

Bare PC applications are known to be high performing systems by their design principles. This fact, have being confirmed by the numerous high performing applications developed on bare and the validation of results gathered from this experiment. To further validate the novelty of the bare server, further performance analyses were done on six webmail servers including the bare server and other OS-based servers. Server processing times, CPU utilization and throughput in LAN and WAN environments and under stress conditions were compared with respect to common email transactions. The results of these tests showed that the performances of the OS-based servers were variable and no single server performed consistently better than the others on all tests. There appeared to

be no simple relationship between LAN and WAN results even for the case of a single server. With a few rare exceptions, only a small drop in performance was seen under stress conditions. However, a detailed study under real workloads and conditions were needed to determine the ability of servers to handle stress. As expected, the Bare PC server performed significantly better on all tests with a few minor exceptions.

A proof of concept performance study of the server was also conducted after making the server TLS-capable. This test compared the processing times for common Webmail transactions with those for two TLS Webmail servers running on Linux and Windows respectively, and a bare PC Webmail server without TLS. The bare PC Webmail server outperformed the OS-based Webmail servers indicating that reduction or elimination of OS overhead should be considered when building high performance Webmail servers. However, the performance reduction due to TLS can be significant even on a bare PC Webmail server with no OS-related overhead. This suggests that the performance of both bare PC and OS-based Webmail servers can be improved by streamlining TLS operations and implementing them more efficiently. The novel design features of bare PC applications could serve as a baseline for the designing secure high-performance and optimal servers and generic applications based on bare PC concepts in the future.

B. Discussions

Significant contributions and novelties have being discussed above. Future recommendations were identified and elaborated as well. A few ideas could also open the door for innovations and be ground breaking for research and the direction for future computing trends. One area of interest is splitting the TLS protocol on different servers into connection server, certificate exchange and key manipulation server and data server.

A similar approach is being investigated on the HTTP protocol and very interesting outcomes have evolved. By splitting the TLS protocol on different servers will make the protocol algorithm more complicated and secure to thwart hacker attacks and intrusions. Another area of interest could be developing rule based algorithms used for developing intrusion detection and SPAM preventions applications and integrating these rules to the architectural algorithms of the bare PC webmail server. By adding these algorithms to the server design will make the single monolithic server more intelligent and secure to detect hacker behaviors.

A significant area that could be investigated and implemented will be developing a file system for the bare server. There is no file management system that currently supports mail storage and inbox folders. Email messages are currently stored and managed in memory which poses its own design issues. Due to lack of file system for bare PC, the IMAP4 protocol for retrieving emails was not considered for this design. It will be helpful to study and implement this protocol on the webmail server to replace the POP3 protocol currently being used, once the file system is implemented in the bare PC environment.

The current design of TLS is limited to a maximum file size of 16K. It can only encrypt and decrypt files that are 16K or less. However conventional webmail servers can send emails that are larger than 16K. The current BIGINT algorithm that is being used in the RSA, HMAC, MD5, SHA-1 and AES calculations makes the bare TLS application 1 millisecond slower than the Linux version. Further studies should be conducted to find solutions to these issues.

The current webmail server does not include the relay forwarding feature as it is in conventional webmail servers. This feature was implemented in the SMTP email server [21, 22, and 23] but was not done in this server, since this was only a lean implementation of webmail server systems on bare PC. Further studies are needed to implement this feature, so that emails bound to different domain can be forwarded to the appropriate SMTP servers.

Appendices

Appendix A. Standard RFCs

The following Request for Comments (RFC) standards, maintained by the Internet Engineering Task Force (IETF) were used in creating bare PC Webmail Server. Compliance to these documented standards allows the webmail to interact with various clients in secure and non secure environments.

RFC 821. SMTP – Simple Mail Transfer Protocol.

RFC 822. Standard for ARPA Internet Text Messages.

RFC 937. Post Office Protocol V2.

RFC 974 Mail routing and the domain system.

RFC 1035 Domain names - implementation and specification.

RFC 1082. POP V3 Extended Services.

RFC 1101 DNS encoding of network names and other types.

RFC 1123 Requirements for Internet Hosts - Application and Support.

RFC 1225. POP V3.

RFC 1426 SMTP Service Extension for 8bit-MIMEtransport.

RFC 1460. POP V3.

RFC 1521 Mechanisms for Specifying and Describing the Format of Internet Message Bodies

RFC 1664 Using the Internet DNS to Distribute RFC1327 Mail Address Mapping Tables.

RFC 1725. POP V3.

RFC 1734. POP3 Authentication Command.

RFC 1847 Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted

RFC 1912 Common DNS Operational and Configuration Errors.

RFC 1939. POP Version 3.

RFC 1957. Some observations on implementations of POP3.

RFC 2045 MIME Part One: Format of Internet Message Bodies.

RFC 2046 MIME Part Two: Media Types. N. Freed, Nathaniel Borenstein.

RFC 2047 MIME Part Three: Message Header Extensions for Non-ASCII Text.

RFC 2049 MIME Part Five: Conformance Criteria and Examples.

RFC 2181 Clarifications to the DNS Specification.

RFC 2183 Communicating Presentation Information in Internet Messages: The Content-Disposition Header.

RFC 2231 MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations. N

RFC 2384. POP URL Scheme.

RFC 2387 The MIME Multipart/Related Content-type

RFC 2449. POP3 Extension Mechanism.

RFC 2595. Using TLS with IMAP, POP3, and ACAP.

RFC 2782 A DNS RR for specifying the location of services (DNS SRV).

RFC 2821. SMTP. April 2001.

RFC 2822. Internet Message Format.

RFC 3156 MIME Security with OpenPGP

RFC 3206. The SYS and AUTH POP Response Codes.

RFC 3501. Internet Message Access Protocol – V4r1.

RFC 4288. MIME Part Four: Media Type Specifications and Registration Procedures.

RFC 4289. MIME Part Four: Registration Procedures.

RFC 5246. The Transport Layer Security (TLS) Protocol Version 1.2.

RFC 2246. The TLS Protocol Version 1.0.

RFC 4326. The Transport Layer Security (TLS) Protocol. Version 1.1.

RFC 5746. Transport Layer Security (TLS) Renegotiation Indication Extension.

RFC 2818. HTTP Over TLS.

RFC 2616. Hypertext Transfer Protocol. HTTP/1.1

RFC 959. File Transfer Protocol (FTP).

RFC 1123. Requirements for Internet Hosts - Application and Support.

RFC 1579. Firewall-Friendly FTP.

RFC 2151. A Primer On Internet and TCP/IP Tools and Utilities.

RFC 2228. FTP Security Extensions (FTPSECEXT).

RFC 3310. Hypertext Transfer Protocol (HTTP) Digest Authentication Using Authentication and Key Agreement (AKA)

RFC 2817. Upgrading to TLS Within HTTP/1.1

Appendix B. Data Tables

Login Summary	Bare Webmail	Mailtraq Webmail	Atmail Webmail
Time(sec)	0.004	0.472	0.291
packets	10	9	12
Avg. Packets/sec	2576.67	10.603	41.271
Avg. Packet size	170.2	297.6	192.333
Bytes	1702	1488	2308
Avg. Bytes/sec	438549.294	3155.356	7937.869
Avg. Mbit/sec	3.508	0.025	0.064

Table 2. Login Timings (Get-LAN)

	BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
Syn	0	0	0
Syn-Ack	0.13	0.131	0.1
Ack	0.152	0.155	0.116
Get	0.352	0.417	0.243
Ack	0.505	115.841	0.604
Data	1.11	183.938	85.809
Data	1.235	184.265	85.809
Ack	1.262	185.028	85.905
200 ok	1.275	185.197	85.905
Ack	1.307	185.224	87.043
Fin-Ack	2.68	185.224	87.043
Ack	2.767	185.224	87.296

Table 3. Login Internal Timings (Get-LAN)

	BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
Post	0	0	0
Ack	1.654	329.524	0.469
302 Found	2.113	329.677	223.147
Ack (Next Get)	8.022	333.077	223.374

Table 4. Login Internal Timings (Post-LAN)

Avg. Bytes/sec	Attachment File Sizes		
	BarePC Webma	Win-Mailtraq Webmail	Linux-Atmail Webmail
1000	1067783.911	129221.688	21179.93
2000	1096372.471	574429.955	26620.943
4000	1400403.682	135299.272	33236.174
10000	2001638.363	149404.109	53134.876
20000	3863263.558	172588.926	61284.768
30000	3536919.207	197408.138	113939.9

Table 5. Timings for Compose with Attachments (LAN)

	BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
100	0.08	2.45	0.12
200	0.16	3.64	0.21
400	0.32	8.12	0.47
600	0.48	13.22	2.93
800	0.62	17.15	3.47
1000	0.79	21.15	3.69
1200	0.93	27.27	3.97
1400	1.1	22.91	4.17
1600	1.21	23.07	4.36
1800	1.35	23.19	4.55
2000	1.54	23.56	4.78

Table 6. Timings for Compose with varying Messages(LAN)

	Bare	Windows			Linux		
	Time (Millisec)	Total Time(Millisec)	Get-Ack-Delay	Ack-1st-Data-Delay	Total Time(Millisec)	Get-Ack-Delay	Ack-1st-Data-Delay
1	0.838	276.377	172.438	101.348	188.643	1.484	184.606
2	4.085	278.266	238.995	37.315	171.848	0.219	168.717
4	4.281	311.468	145.874	163.538	198.262	0.272	194.599
6	3.81	321.088	151.139	161.418	225.621	0.397	222.555
8	4.528	314.003	119.461	191.952	245.349	0.279	241.57
10	4.684	289.055	168.881	116.501	298.508	0.262	295.089

Table 7. Timings for Inbox(LAN)

	Bare	Windows			Linux		
		Total	Get-Ack-Delay	Ack-1st-Data-Delay	Total	Get-Ack-Delay	Ack-1st-Data-Delay
1000	1.451	392.702	142.618	247.915	266.508	1.086	263.488
2000	1.619	426.829	183.959	240.614	308.421	2.357	304.705
4000	1.763	430.014	151.575	276.079	328.808	2.471	324.258
10000	1.91	436.267	133.084	300.206	315.286	2.41	311.305
20000	3.073	439.383	214.063	215.245	426.752	0.28	423.393

Table 8. Timings for Read with varying Message Sizes (LAN)

	Bare	Windows			Linux		
	Time(Millisec)	Total Time(Millisec)	Get-Ack-Delay	Ack-1st-Data-Delay	Total Time(Millisec)	Get-Ack-Delay	Ack-1st-Data-Delay
1000	1.144	175.876	13.046	162.632	69.151	0.653	67.423
2000	1.531	13.77	13.1	0.651	81.03	0.267	69.108
4000	1.474	13.986	13.507	0.259	68.791	0.302	67.015
10000	2.044	14.903	13.833	0.794	68.068	1.986	60.069
20000	2.623	21.359	19.379	0.345	70.719	0.637	62.055
30000	3.9	17.562	13.942	0.323	68.374	0.516	61.134

Table 9. Timings for Read Message with Attachments (LAN)

Avg. Packets/sec Attachment File Sizes			
	BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
1000	11.981	1.028	0.203
2000	10.319	4.615	0.226
4000	11.421	4.629	0.274
10000	13.736	0.771	0.426
20000	19.385	1.314	0.683
30000	31.05	1.483	0.912

Table 10. Server Throughputs (LAN)

	BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
Syn	0	0	0
Syn-Ack	0.037	0.084	0.063
Ack	41.607	41.353	42.247
Get	50.437	53.068	51.154
Ack	50.521	243.859	51.297
Data	51.116	258.1	126.489
Data	51.252	258.478	126.49
200 ok	51.284	306.436	126.586
Ack	101.243	312.249	176.527
Fin-Ack	104.265	319.31	200.686
Ack	104.338	353.395	200.939

Table 11. Login Internal Timing (Get-WAN)

	BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
Post	0	0	0
Ack	1.532	296.932	0.2078
302 Found	1.99	297.016	101.5048
Ack (Next	55.954	358.729	101.5978

Table 12. Login Internal Timing (Post-WAN)

	BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
1000	117.047	980.109	497.589
10000	236.338	1319.486	542.75
20000	342.967	1460.869	808.039
40000	612.378	1924.657	1206.85
80000	1149.209	2988.137	2071.27
120000	1689.487	3448.492	3044.687

Table 13. Timings for Compose with Varying Message Sizes (WAN)

	BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
1000	1.316	388.931	186.542
10000	2.613	578.236	271.544
20000	4.633	438.994	356.364
40000	7.492	558.31	559.311
80000	95.36	973.708	904.479
120000	204.024	1326.838	1282.33

Table 14. Timing for Read Message with Varying Message Size (WAN)

	BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
1000	0.459	0.168	0.002
10000	0.514	0.513	0.233
20000	0.576	0.479	0.019
40000	0.641	0.531	0.126
80000	0.696	0.41	0.342
120000	0.72	0.466	0.343

Table 15. Average Mbit/sec Throughput (WAN)

threads	Squirrelmail-Linux Fedora			Atmail-Linux Cent OS			Icewarp-Windows XP			Bare		
	Total time	Avg. click time	Avg. Kbit/sec	Total time	Avg. click time	Avg. Kbit/sec	Total time	Avg. click time	Avg. Kbit/sec	Total time	Avg. click time	Avg. Kbit/sec
1	17860	155	85.97	70948	678	89.28	19244	169	81.23	28808	257	88.92
2	28889	160	63.68	113809	677	89.3	30748	169	81.22	46475	260	88.26
3	41768	168	80.09	156935	679	89.04	42562	170	80.52	68873	281	81.65
4	50940	160	83.49	199907	680	88.97	53767	169	81.1	84378	270	84.89
5	61109	159	83.91	246467	694	86.79	65148	169	81.18	102517	270	84.77
6	72924	160	83.65	283996	678	89.27	77802	171	80.59	120618	270	84.75
7	90105	173	79.92	325573	678	89.22	96563	186	74	136973	268	85.16
8	89180	151	88.4	367820	679	89.15	112879	172	71.84	156844	271	84.59
9	101401	154	86.8	404818	677	89.37	110834	169	81.29	173098	270	84.83
10	110169	152	88.16	447596	677	89.33	122518	169	81.26	193286	270	84.57

Table 16. Stress Tool Timings (Web Stress Tool)

	Clicks or Post	BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
		BarePC Webmail	Win-Mailtraq Webmail	Linux-Atmail Webmail
time	12	53	3335	1641
1 user	59	270	14463	8476
	118	540	28552	17160
5 users	57	258	11031	8062
	293	1273	77786	41666
	587	2467	160536	83343
10 users	112	472	25700	15532
	586	2477	174790	82678
	1172	4966	362338	167322

Table 17. Stress Tool Timings (Web Stress Tool)

Type	Rate	Fetches/Sec	Bytes/Sec	msecs/connections			msecs/first-response		
BarePC Webmail				Mean	Max	Min	Mean	Max	Min
	10	9.99444	27314.8	0.163553	0.31	0.133	0.219111	0.559	0.176
	100	99.9944	273285	0.155307	0.742	0.125	0.186128	1.597	0.164
	200	199.994	546585	0.150718	1.511	0.122	0.182068	3.104	0.159
	400	499.994	1.37E+06	0.160207	5.251	0.123	0.179013	6.684	0.159
	600	950.433	2.60E+06	0.149529	8.948	0.106	0.182825	6.861	0.146
	800	999.994	2.73E+06	0.148165	8.644	0.105	0.182362	4.088	0.146
	1000	999.994	2.73E+06	0.147324	16.35	0.108	0.181066	6.384	0.146

Table 18. Stress Tool Timings (HTTPLOAD)

Compose		
	Bare-PC-TLS	Bare-PC-No TLS
1	19	10
2	30	14
4	28	16
6	26	16
8	18	12
10	17	12
12	16	12
16	17	12
Read		
	Bare-PC-TLS	Bare-PC-No TLS
1	15	1
2	25	1
4	15	2
6	20	2
8	28.8	3
10	28.8	2
12	36	2
16	38	3
Throughput-Compose		
	Bare-PC-TLS	Bare-PC-No TLS
1	3.903	4.073
2	4.195	3.118
4	5.416	3.494
6	6.467	6.787
8	7.401	6.138
10	8.072	8.806
12	9.312	9.791
16	11.241	12.031
Throughput-Read		
	Bare-PC-TLS	Bare-PC-No TLS
1	25.305	24.47
2	3.158	26.293
4	35.982	34.032
6	37.765	39.427
8	2.27	32.845
10	2.327	45.79
12	3.156	51.612
16	3.823	49.028

Table 19. TLS / NON TLS Timings

Compose				
	Bare-PC-TLS	Bare-PC-No TLS	Win-Icewarp	Linux-Atmail
1	16	16	1030	172
2	23	20	927	123
4	22	16	786	277
6	16	17	1219	353
8	16	10	1090	298
10	16	6	1040	257
12	19	5	1039	251
16	16	6	1180	243
Inbox				
	Bare PC	Bare PC-No TLS	Win-Icewarp	Linux-Atmail
10	0.042		0.291	0.238
Read				
	Bare-PC-TLS	Bare-PC-No TLS	Win-Icewarp	Linux-Atmail
1	65	10	141	198
2	40	10	118	224
4	38	20	237	199
6	62	20	269	213
8	40	9	236	211
10	30	9	234	278
12	30	7	305	142
16	30	5	125	165
Throughput-Compose				
	Bare-PC-TLS	Bare-PC-No TLS	Win-Icewarp	Linux-Atmail
1	3.891	4.184	0.014	0.216
2	7.382	3.229	0.041	0.105
4	5.463	5.68	0.072	0.029
6	6.506	6.311	0.072	0.321
8	7.395	7.034	0.09	1.203
10	8.336	5.752	0.114	1.176
12	8.087	9.1	0.028	1.031
16	10.304	9.375	0.149	1.147
Throughput-Read				
	Bare-PC-TLS	Bare-PC-No TLS	Win-Icewarp	Linux-Atmail
1	1.312	30.133	0.192	0.065
2	5.777	34.88	0.233	0.148
4	1.435	40.444	0.12	0.075
6	1.472	33.978	0.024	0.072
8	2.551	48.045	0.12	0.072
10	2.671	39.119	0.127	0.189
12	1.458	51.673	0.102	0.258
16	1.441	54.667	0.031	0.323

Table 20. TLS Timings (Compose and Read)

	Bare-PC_TLS	Linux-Atmail-TLS	Win-Icewarp
Application Data (GET)	0	0	0
Ack	0.157	3.308	0.903
Data	0.53	3.43	215.128
Application Data (200 Ok)	0.589	3.558	215.304
Ack	0.615	15.2	215.335
Application Data (POST)	3721.587	3083.517	5066.058
Ack	3721.801	3084.104	5067.587
Application Data (302 Found)	3721.997	3203.372	5069.749
Application Data (GET-Next Ack)	3725.154	3220.659	5071.364
Ack	3735.323	3221.005	5071.398
Data	3735.86	3221.711	5071.485
Data	3735.98	3222.045	5071.638
Application Data (200 Ok)	3736.134	3222.157	5071.754
Ack	3736.155	3222.539	5071.904

Table 21. TLS Internal Timings

REFERENCES

[1] A. Ghafoor, S. Muftic, and G. Schmölzer, "CryptoNET: Design and implementation of the Secure Email System", Proceedings of the 1st International Workshop on Security and Communication Networks (IWSCN), 2009, pp. 1 – 6.

[2] Afterlogic Webmail Server, www.afterlogic.com Retrived on May 20th, 2010

[3] Appiah-Kubi, P., Karne, R.K. and Wijesinha, A. L. "The design and Performance of a Bare PC Webmail Server," 12th IEEE International Symposium of Advances on High Performance Computing and Networking. (AHPCN-HPCC) 2010, Melbourne Australia. September 2010.

[4] Atmail-Linux Webmail Server, www.atmail.com Retrived on October 5th, 2009.

[5] Axigen Email Sevrer, www.axigen.com Retrieved on January 23th, 2009.

[6] B. Ramsdell and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[7] Bawanz, J., Paulson, D. and Struss, J. "Spam software evaluation, training and support: fighting back to reclaim the email inbox". Proceedings of the 32nd Annual ACM SIGUCCS Fall Conference. (SIGUCCS), 2004. pp. 385-387.

[8] Cannata, G. " Phymail Box: An Information appliance that checks and prints only important emails". Personal and Ubiquitus Computing Journal. Vol. 10. Issue 2-3. January 2006. pp. 170-172.

[9] Chrobok, N., Trotman, A. and O'Keefe, R. "Advantages and vulnerabilities of pull-based email-delivery". Proceedings of the 8th Australasian Conference on Information Security. (AISC), 2010. pp. 22-31.

[10] Cristea, M. and Groza, B. "Augmenting a Webmail Application with Cryptographic Puzzles to Deflect Spam". Proceedings of the International Conference on New Technologies, Mobility and Security. (NTMS), 2011.

[11] Cisco WebEx Mail: High availability design for your most mission critical application. www.ciscowebexmail.com. Retrieved on May 20th, 2010.

[12] Complexity Is Killing IT, Say Analysts IDG News Service (04/13/07) Krill, Paul, http://www.techworld.com/opsys/news/index.cfm?newsID=8525&pagtype=all.

[13] Elprin, N. and Parno, B. "An Analysis of Database-Driven Mail Servers". Proceedings of the 17th USENIX Conference on System Administration. (LISA), 2003. pp. 15-22.

[14] Emailman Webmail Server, www.emailman.com Retrieved on May 21th, 2010.

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

[16] Emdadi, A., Karne, R. K. and Wijesinha, A. L. "TLS Timings on a Bare PC", WORLDCOMP/SAM 2010

[17] Engler, D. R. and Kaashoek, M.F. "Exterminate all operating system abstractions," 5th Workshop on Hot Topics in Operating Systems, USENIX, Orcas Island, WA, May 1995, p. 78.

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

[19] Ford, B., Hibler, M., Lepreau, J., McGrath, R. and Tullman, P. "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.

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

[21] Ford, G., Karne, R.K., Wijesinha, A. L. and Appiah-Kubi, P. "The design and implementation of a Bare PC email server". 33rd IEEE International Computer and Applications Conference, July 2009.COMPSAC09

[22] Ford, G., Karne, R.K., Wijesinha, A. L. and Appaiah-Kubi, P. "The performance of a Bare Machine email server". 21st International Symposium on Computing Architecture and High Performance Computing, IEEE October 2009.SBAC-PAD

[23] Ford, G. An Email Server on a Bare PC, Doctoral Dissertation, May 2010

[24] He, L., Karne, R.K. and Wijesinha, A.L. "The design and performance of a Bare PC Web server", International Journal of Computers and Their Applications, vol. 15, June 2008, pp. 100-112.

[25] He, L. A Bare Machine Web Server, Doctoral Dissertation, May 2008.

[26] Hexamail Webmail Server, www.hexamail.com Retrived on June 10th, 2010.

[27] IceWarp Mail Server, www.icewarp.com Retrived on May 20th, 2010

[28] Kageyama, E., Maziero, C. and Santin, A. "A pull-based email architecture," Proceedings of the 2008 ACM symposium on Applied computing, ACM, Fortaleza, Ceara, Brazil, 2008, pp. 468-472.

[29] Karne, R. K., Jaganathan, K. V. and Ahmed, T. "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.

[30] Karne, R. K., Jaganathan, K. V. and Ahmed, T. "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.

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

[32] Kaushik, S., Wijesekera, D. And Ammann, P. "BPEL Orchestration of Secure Webmail", 3rd ACM Workshop of Secure Web Services, Alexandra VA, ACM 2006, pp 85 – 94.

[33] Khaksari, G. H. A VoIP on a Bare PC, Doctoral Dissertation, May 2007.

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

[35] Kim, J. And Hong, J. W. "Design and Implementation of a Web-based Internet/Intranet Mail Server Management System", IEEE International Conference on Communication, 1999, Vol.1, pp. 641-645

[36] Lieberman, E. and Miller, R.C. "Facemail: showing faces of recipients to prevent misdirected email", Proceedings of the 3rd symposium on Usable privacy and security, Pittsburgh, Pennsylvania.

[37] Mailtraq email server-the complete email server, www.mailtraq.com Retrieved on June 10th, 2010.

[38] Mazieres, D. and Kaashoek, M. F. "The design, implementation and operation of an email pseuodonym server". 5th Conference on Computer and Communications Security, 1998, pp. 27-36.

[39] McDowell, L., Etzioni, O., Halevy, A. and Levy, H. "Semantic email". 13th International Conference on World Wide Web, New York, NY, USA, 2004, pp. 244 – 254.

[40] Noiumkar, P. And Chomsiri, T. "Top 10 Free Webmail Security Test Using Session Hijacking", 2008, http://doi.ieeecomputersociety.org/10.1109/iccit.2008.324

[41] Nusser, S., Cerruti, J. and Wicox, E. "Enabling Efficient Orienting Behavior in Webmail Clients", 20th Annual ACM Symposium on User Interface Software and Technology, Newport Rhode Island 2007.

[42] Pai, V. S., Druschel, P. 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.

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

[44] Pathak, A., Jafri, S. A. R. and Hu, Y. C. 'The case for high-performance spam-Aware mail server architecture". 29th IEEE International Conference on Distributed Computing Systems, pp. 155-164, July 2009.

[45] Petmail Webmail Server, www.petmail.lothar.com Retrieved on June 12th, 2010

[46] RoundCube Webmail Server, www.roundcube.com Retrieved on November 30th, 2010

[47] SquirrelMail Webmail, www.squirrelmail.com Retrieved on November 30th, 2010

[48] Schatzmann, D., Muhlbauer, W., Spyropoulos, T. and Dimitropoulus, X. " Digging into HTTPS: flow-based classification of Webmail traffic". Proceedings of the 10th Annual Conference on Internet Measurement. (IMC), 2010. pp. 322-327.

[49] Singaraju, G. and Kang, B.B. "RepuScore: collaborative reputation management framework for email infrastructure". Proceedings of the 21st USENIX Conference on Large Installation System Administration. (LISA), 2007.

[50] T. Dierks, and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

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

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

[53] Tomar, A.A., Chaudhari, S., Patil, J. and Rawat, A. "Implementation and Security Analysis of a CallerId Augmented 2FA Setup for Secure Web-mail Access". Proceedings of the 2010 IEEE International Conference on Advances in Communication Network and Computing. (CNC), 2010. pp. 346-348.

[54] Tran, T., Rowe, J. and Wu, S. F. "Social email: a framework and application for more socially-aware communications". Proceedings of the 2nd International Conference on Social Informatics. (SocInfo), 2010. pp. 203-215.

[55] Tung, C. and Tsai, S. "Tuning Webmail performance-The design and implementation of Open Webmail". National Cheng Kung University, www.openWebmail.org Retrieved on May 20th, 2010.

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

[57] Web Stress Tool 7, www.paessler.com Retrieved on May 20th, 2010

[58] WinWebmail Server, www.winWebmail.net Retrieved on October 20th, 2010

[59] Yeh, C. and Mao, C. "Adaptive e-mail intention finding mechanism based on e-mail words social networks", Proceedings of the 2007 workshop on Large scale attack defense, Kyoto, Japan

Curriculum Vitae

Patrick Appiah-Kubi
3582 Powder Mill Rd, #103
Beltsville, Maryland 20705
appiahkubi@towson.edu

OBJECTIVE

Looking for a Position in Academia that will provide the opportunity for advancement and to impact the knowledge and expertise I have gained to younger generations.

EDUCATION

Towson University, Towson, Maryland
Doctor of Science in Information Technology
December 2011
Dissertation: Lean Secure Webmail Server on a bare PC

Indiana State University, Terre Haute, Indiana
Master of Science in Electronics and Computer Technology
May 2005
Thesis: Preparation, Development and Evaluation of a Duplo Automated Assembly System.

Kwame Nkrumah University of Science and Technology, Kumasi, Ghana
Bachelor of Science in Computer Science
July 2002
Project: Automation of Nutritional Composition of Foods and their Medical Values

EXPERIENCE

TEACHING EXPERIENCE

Lecturer
Computer and Information Science Department, Towson University, Towson Maryland
August 2010 - Present

Courses Taught

- Computers and Creativity
- Information and technology for business
- Database Management Systems

Extra Curriculum Activities

- Research
- IS and IT committees

Adjunct Instructor
Computer and Information Science Department, Towson University, Towson Maryland
January 2009 – May 2010

Courses Taught

- Information and technology for business.

Extra-Curricular Activity

- Research

Teaching Assistant
Computer and Information Science Department, Towson University, Towson Maryland
August 2007 – May 2009

Courses Taught

- Information and technology for business.

Extra-Curricular Activities

- Research

Graduate Assistant
Electronics and Computer Technology Department, College of Technology, Indiana State University, Terre Haute, Indiana September 2003 - May 2005

Courses Taught

- Local Area Networks
- Project Management

Extra-Curricular activities

- Prepared lesson plans for various professors
- Assisted in research activities for the department

INDUSTRY EXPERIENCE
Systems Engineer/IT Consultant
MVS Consulting, Washington DC January 2007 – May 2010

- Prepared proposals for contract bidding.
- Lead the installation of network printers for the metropolitan police in Washington DC.
- Liaised with the District of Columbia Government to prepare a new IT policy for employees in the district.
- Member of the team that sanitized old computers for the department of human services in Washington DC by wiping off the hard drive with WipeDrive 3.0
- Lead contracting engineer for Prince George's County systems upgrade.

- Created PC system images using Norton ghost and transfer it to other systems
- Installed and configure Windows and Linux Servers
- Supervised the DC Mental health department system upgrade.
- PC troubleshooting and repairs.
- Supervised the DC Public Schools systems upgrade.
- Prepared reports and contract bills.
- Lead consultant for the Baltimore county government network upgrade.
- Prepared progress report on every contract the company executed for its clients.
- Project member on the DC fire department network project. The project included feasibility study and upgrading the fire department's network infrastructure and making it more proactive to respond to 911 calls promptly.
- Managed VPN and Watchguard Firewall
- Administered and Configured the Novell Netware on workstations
- Installed and managed Cisco Routers/Switches
- Wrote monthly reports on the performance of the network and made recommendations for new technologies
- Project Leader for the DC Department of Health Wired/Wireless Network Setup and Support team during the 2009 Immunization project.

Manufacturing Technician

Kelly Engineering Services

Contracted to Intel Corporation, Hudson, Massachusetts January 2006 - December 2006

- Initialized and verified fresh silicon scribes on the OCR tools using computer-imaging systems. Verified silicon wafers that were needed to be processed into the production line using work stream
- Performed daily preventive maintenance on wafer handling tools in work stream
- Entered results on particle check in SPC++ Statistical Software and analyzed the graphical output of the particle check.
- Coordinated the flow of test wafers within the factory
- Packaged and shipped test and production wafers to other Intel sites for further analysis
- Received and processed production wafers into the production line
- Check for distortions on wafer box using AUGUST imaging system and entered the results in SPC++ for statistical analysis
- Performed troubleshooting of computers that control the production tools.

Network Administrator
Bsystems Limited, Accra, Ghana November 2002 - July 2003

- Supported and Monitored a Windows 2000 Network with 3 servers and 50 workstations
- Managed VPN and Watchguard Firewall
- Administered and Configured the Novell Netware on all workstations
- Installed new Software, Hardware and Cisco Routers/Switches to enhance network performance
- Wrote monthly reports on the performance of the network and made recommendations new technologies
- Installed and added new printers and scanners to the network

Exchange Student
IMA Laboratory, Hamburg University, Germany August 2002 - October 2002

- Participated in research work on Robotics and Computer Vision using C++
- Familiarized myself with UNIX and LINUX

Systems Administrator
Cutting Edge Solutions, Accra, Ghana May 1999 - July 2002

- Performed Data Recovery and Retrieval for clients
- Setup, configured and Supported Windows NT/2000 Server networks
- Maintained the company's database.
- Setup and repaired network Printers
- Performed PC hardware repairs, troubleshooting and testing
- Assisted end users with Windows XP issues and login issues
- Trained end users in the use of Windows 2000/ XP and Microsoft Office suits (Word, Excel, PowerPoint, Outlook and Access)
- Installed new Software packages on workstations. Trained end users on the new software packages

RESEARCH INTEREST

- Bare Machine Computing Webmail and email
- Communications Network Performance and Management
- IT Business Strategy and Project Management
- Network Security and Wireless Networks
- High Performance Computing
- Cloud Computing

PUBLICATIONS

- P. Appiah-kubi, R.K. Karne and A.L. Wijesinha. "A Bare PC TLS Webmail Server". International Conference on Computing, Networking and Communication, January 2012. ICNC- CNC (Accepted)
- P. Appiah-kubi, R.K. Karne and A.L. Wijesinha. "A Performance Study of Conventional and Bare PC Webmail Servers". 7th International Conference on Networking and Services, May 2011. ICNS
- P. Appiah-Kubi, R.K. Karne, and A.L. Wijesinha. " The Design and Performance of a bare PC Webmail Server". 12th IEEE International Symposium on Advances of High Performance Computing and Networking, September 2010. AHPCN-HPCC.
- G. Ford, R.K. Karne, A.L. Wijesinha, and P. Appiah-Kubi. "The performance of a Bare Machine email server". 21st International Symposium on Computing Architecture and High Performance Computing, IEEE October 2009.SBAC-PAD
- G. Ford, R.K. Karne, A.L. Wijesinha, and P. Appiah-Kubi. "The design and implementation of a Bare PC email server". 33rd IEEE International Computer and Applications Conference, July 2009.COMPSAC09

TECHNICAL SKILLS

Operating Systems: Windows 7, Windows Vista, Windows XP, Windows 2000 Server/Professional, Windows Server 2003, Windows NT 4.0 Server/Workstation, Windows98, Windows95, Windows 3.x, Mac OS X, DOS, UNIX, LINUX

Software Packages: AutoCAD, RSNetworx for DeviceNet Network, RSLinx, RSlogix 5000, Automation Studio, RSview, Microsoft Excel, Microsoft PowerPoint, Microsoft Word, Minitab, Microsoft Visio, ConceptDraw pro, Adobe Photoshop, Remedy, AMAG Enterprise, WipeDrive 3.0, Norton Ghost 2003, Netboot IV.

Database: JDBC, Oracle 9i, Oracle 11g, SQL 2005 Server, LP/SQL, Crystal Reports, Microsoft Access, Hibernate.

Programming Applications: Visual Basic, Visual C++, C#, FoxPro, and Java.

Web Applications: Macromedia Dreamweaver 8, Macromedia Fireworks 8, FrontPage 2000/2003, Flash 8, HTML and DHTML.

Network Applications: Active Directory Server, Microsoft Exchange 2003 Server, File/Print Server, Web Server, Application Server, BackupExec, Lotus Notes, Citrix, SSH

- Experience includes Novell Netware, computer-based/ aided instructional system development, hardware/software configuration and troubleshooting
- Multiple domain/server Windows NT/2000/2003 network administration including multihomed servers with NAT, VPN configuration using Internet tunneling, Firewall configuration, DNS, DHCP, WINS, IIS.

Hardware: Compaq DL580, ML 380 Server, Dell PowerEdge, Terminal server, 3COM Remote office Bridge/Router, DSU/CSU, Modem, APC UPS, SCSI, Ultra wide SCSI, ID Card Reader, IDE/EIDE hard drives, CD ROM, IBM compatible PC's/Laptops, Cisco Routers/Switches.

AWARDS AND HONORS

Graduate Student Research Fellowship, Towson University August 2009 – May 2010
Graduate Student Scholarship, Towson University August 2007 – May 2009
Graduate Student Scholarship, Indiana State University August 2003 – May 2005

PROFESSIONAL AFFILIATIONS

National Society of Black Engineers
Association of Computing Machinery
Institute of Electrical and Electronic Engineers
Project Management Institute

REFERENCES

Dr. Ramesh K. Karne rkarne@towson.edu
Dr. Alexander L. Wijesinha awijesinha@towson.edu
Dr. Chao Lu clu@towson.edu