2010 Fifth International Conference on Systems and Networks Communications

# A Study of Bare PC SIP Server Performance

A. Alexander, A. L. Wijesinha, and R. Karne
Department of Computer & Information Sciences
Towson University
Towson, MD 21252
USA

*Abstract*—**In bare computing, applications run directly on an ordinary PC without using an operating system (OS). Advantages of bare PC computing include elimination of overhead due to the OS, and immunity against attacks targeting OS vulnerabilities. We evaluate the performance of a bare PC SIP server by determining its throughput and latency in a dedicated test network using the open source SIPp workload generator to generate requests for registration (register, register update, and register logout operations), proxying (invite and invite-not-found operations) and redirection (invite redirect operation) with and without authentication. We also report internal timings for the server. The performance of the server is then compared with the OpenSER and Brekeke SIP servers running on Linux and Windows respectively. Our results show that the bare PC SIP server has low cost for internal SIP-related operations, higher throughput than the Windows server, and higher or equal throughput than the Linux server except in the case of redirection. Its latency is more than that of an OS-based server only in the case of invite with authentication and invite-not-found without authentication.**

*Keywords*—**SIP server, application object, bare PC, bare computing, performance.**

## I. Introduction

In a bare PC, applications run directly over the hardware of an ordinary PC such as a desktop or laptop without using any operating system (OS). Previous studies [1], [2] have shown that bare PC email servers and Web servers can significantly outperform their counterparts running on conventional OS-based systems. Bare PC applications can also be deployed in secure environments since they are immune to attacks that target specific OS vulnerabilities. Although many studies have investigated the design and performance of network and security protocols [3] in a bare PC, including those used for peer-to-peer communication among bare VoIP clients [4], none has included the Session Initiation Protocol (SIP). SIP has since become the most frequently used protocol today for initiating VoIP calls and for media session support with a variety of other applications including video streaming, instant messaging, gaming and IPTV.

In this paper, we evaluate the performance of a bare PC SIP server and compare it with two commonly used OS-based SIP servers: OpenSER (now OpenSIPS) running on Linux and Brekeke running on Windows. All studies are conducted using identical ordinary (non-server) machines.

SIP servers play an important role in supporting audio or multimedia sessions. For example, OpenSER uses SIP to provide voice, video, instant messaging, and presence services. In general, SIP servers locate and register clients, provide proxy services for forwarding SIP messages, or redirect SIP requests to other servers. An optimized SIP server can thus help improve the overall performance of audio or video applications although it is typically not directly involved in the actual transmission of audio or video. The throughput and latency of the SIP server when responding to requests from SIP user agent clients and other SIP servers are often used as measures in evaluating its performance.

We use the popular open source SIP workload generator SIPp to evaluate the performance of the bare PC SIP server by measuring its throughput and latency for registration, proxying, and redirection, with and without authentication, for increasing workloads. We then evaluate the performance of the OS-based (Linux and Windows) servers for the same workloads when running on compatible hardware. Our results show that the bare PC SIP server has higher or equal throughput to the Linux server and higher throughput than the Windows server, except in case of redirection, when its throughput is less than that of the Linux server. The latency performance of the bare PC server is also shown in general to be better than or equal to that of Linux server and better than that of the Windows server except for invite with authentication and invite-not-found without authentication. Our contributions in this paper include: 1) results characterizing the performance of a bare PC SIP server running on an ordinary desktop; 2) internal timings for SIP-related operations on a bare PC SIP server; and 3) detailed comparisons of the throughput and latency for a bare PC SIP server, Linux OpenSER, and Windows Brekeke servers running on identical machines.

The rest of this paper is organized as follows. In Section II, we describe the bare PC SIP server and relevant optimizations. In Section III, we briefly summarize related work. In Section IV, we give details of the experimental setup and discuss the results of the performance study. In Section V, we present the conclusion.

## II. Bare PC Sip Server Overview

Any bare PC application, including the bare PC SIP server, is encapsulated in an application object (AO) [5]. Since there is no OS, minimal code for the application to run on the PC hardware is contained in the AO. This means that the AO contains the code for the bootable self-executing application itself, any required network interface drivers, handlers for protocols used by the application, and memory and task management algorithms including modules to facilitate concurrency, scheduling, and security. Real memory is used since there is no hard disk, and unlike in a conventional OS-based protocol stack, application code is intertwined with protocol code to eliminate redundancy and improve efficiency as described for the case of a bare PC email server in [6].

The bare PC SIP server AO implements a lean version of SIP that provides essential functionality only. Additional features such as those needed to support load balancing and media stream security are not included. Although a bare PC SIP server that can operate over TCP or UDP has been implemented, this paper only considers SIP over UDP since the majority of SIP servers employed in practice use UDP.

A received UDP packet containing a SIP message is placed in the Ethernet buffer, where the bare PC SIP application can directly access it i.e., there is no distinction between real mode and user mode (or kernel-level operations) since there is no OS. The Ethernet handler processes the packet, determines that the packet is for IP, and the IP handler in turn processes the packet and invokes the UDP handler, which verifies the UDP checksum (if this feature is enabled) and the port number. In case of the SIP server, the port number is 5060 and the packet is processed by the SIP server application. If a response needs to be sent, the SIP application, the UDP, IP and Ethernet handlers are invoked in order to add the respective headers before the packet is transmitted by the network interface hardware. In a bare PC, data copying is minimized since headers are added to a single copy of the message.

At a minimum, only two CPU tasks are required by the SIP server AO: a receive (Rcv) task that processes a received packet all the way from its arrival in the Ethernet buffer until completion, and a Main task that runs whenever a Rcv task completes (and also when the system is booted or the system is idle). For example, for a register message, the Rcv task itself could manage the lookup and update operations and send the response to the client. However, a separate SIP task could be used for each request when concurrent processing of requests is needed. In case of the invite message for example, a new SIP task could be used to handle the request since there may be a delay in contacting the peer (callee) and receiving the response. In general, since a typical workload involves a mix of requests for different services, bare PC SIP server performance would be improved by the concurrent handling of requests. In the bare PC SIP server and in other bare PC applications, a CPU task is allowed to run to completion unless it is "suspended" due to waiting for an event such as a response. As soon as the event occurs, the task is "resumed" so that it can be executed. This simple task scheduling algorithm and the disabling of timer interrupts on a bare PC reduces the context switching overhead of an OS and enables the CPU to be utilized with maximum efficiency on behalf of an application.

## III. Related Work

There are many commercial and open source servers implementing SIP and its companion protocol SDP. While a SIP server usually runs over UDP and in some cases over TCP, the use of SCTP as a transport protocol for SIP has also been studied [7]. An early study on SIP server performance [8] found that the overhead on a Java SIP server due to security mechanisms such as authentication and TLS was negligible. However, the study in [9], which measured throughput and latency in a dedicated gigabit Ethernet for stateless and stateful proxies over UDP and TCP, showed that authentication, TCP, or the operation/server configuration can significantly change SIP server performance. Their experiments were conducted using a 3.06 GHz server class machine, and only the performance of a single SIP server (OpenSER on Linux) was evaluated. In [10], SIP server performance for several stateful SIP proxies over UDP was evaluated. The authors concluded that the overhead due to string processing operations and memory management could consume significant processing time and that performance varied considerably depending on the proxy. Recent work on SIP servers has dealt with performance under overload conditions [11], scalability issues [12], [13], load balancing [14], and the impact of transport protocols on performance [15]. The main difference between previous studies and this paper is that we study the performance of a bare PC SIP server and compare it with the performance of two OS-based SIP servers using ordinary desktop (non-server) machines. Also, in addition to evaluating performance for the usual register, invite, and redirect operations, we also evaluate SIP server performance with the register update, register logout, and invite-not-found operations likely to be encountered in practice. We only consider SIP over UDP with stateless proxying, which is the most common configuration when setting up VoIP calls.

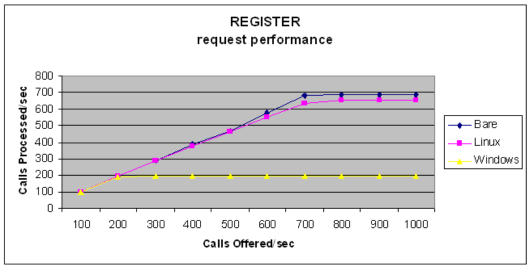
## IV. Results

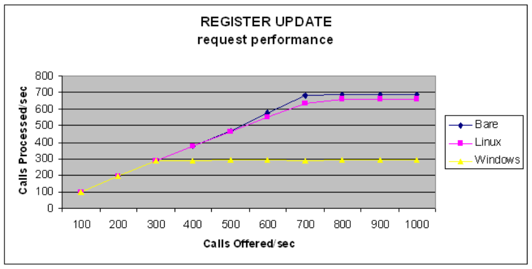
In this section, we present the results obtained from our experiments. We compare throughput and latency for the bare PC and OS-based SIP servers using register, register update, register logout, invite, invite-not-found, and redirect operations.

### A. *Experimental Set Up*

The test network consists of a 100 Mbps Ethernet to which each SIP server and the client machines running SIPp are connected. In addition to the bare PC SIP server, the details of the systems and software used is as follows: OS-based SIP servers: OpenSer SIP Server ver 1.3.2 –notls (Linux) OpenSer (KAMILIO/OpenSIPS) and Brekeke SIP Server ver 2.1.6.6 (Windows) utilizing the Jakarta Web Server and Java platform; machines: Dell GX260's with Intel Pentium 4 (2.4 GHz), 1.0 GB of RAM and 3COM Ethernet 10/100 PCI network cards; OSs: Microsoft Windows XP Professional ver. 2002 Service Pack 2 and Linux Ubuntu 8.04 Kernel 2.6.24-16; workload generator: SIPp.

For register updates, the SIP Server searches its user database for a match and then updates the corresponding user's location data and registration expiration time; and in the register logout operation, it removes the user from the database. The invite operation requires the server to lookup the callee's contact details in its database, forward the request to the callee, and send the response back to the caller. The invite-not-found operation is similar to invite except that the callee is not found in the database. For redirect, the server receives an invite message, but instead of forwarding the response to the callee, it forwards a temporarily moved message back to the caller.





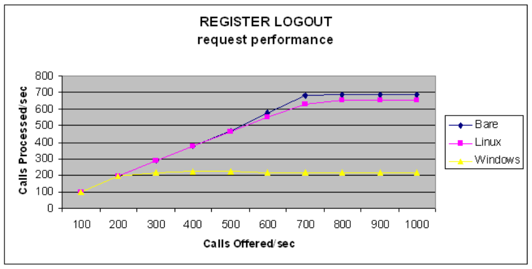
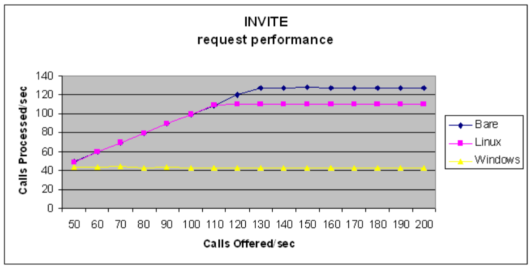
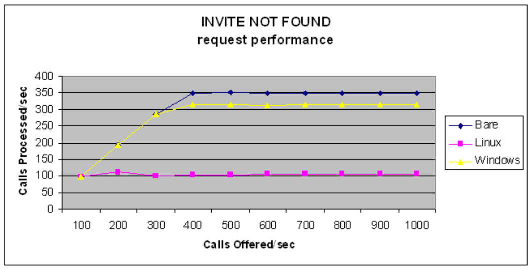


Figure 1. Throughput for register without authentication





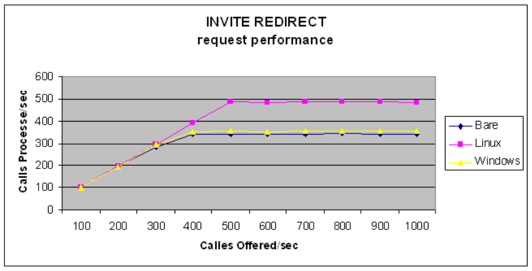
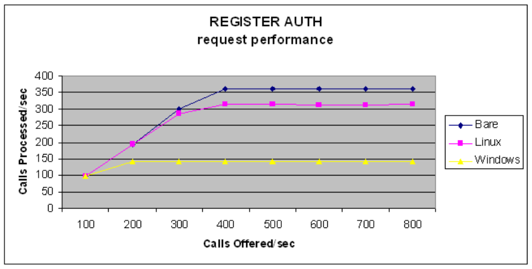
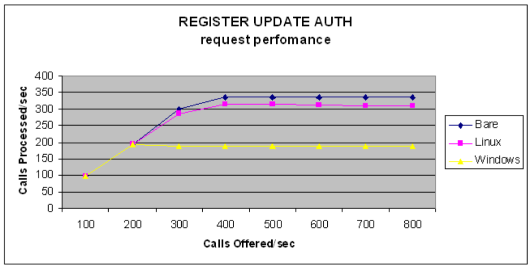


Figure 2. Throughput for invite without authentication





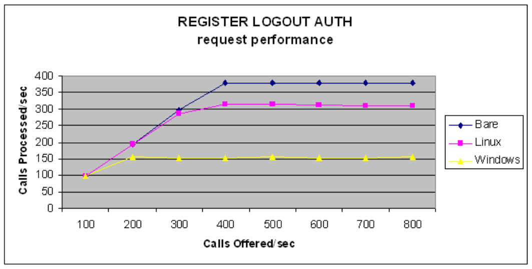


Figure 3. Throughput for register with authentication

For the register, register update, and register logout operations, latency measures the delay at the user agent between sending the register message and receiving the "200 OK" message. Latency for the invite operation measures the sum of two delays: the time between the invite message and "200 OK" messages; and the time between the "bye" and "200 OK" messages. Each of these operations was also tested with authentication enabled, which adds processing overhead due to verifying the MD5 hash, and extra message overhead due to the "unauthorized" message for registration and "407 proxy authentication" message for invite (and their responses). Latency for registration with authentication measures the sum of two delays: the time between the register request and the "unauthorized message"; and the time between the new register

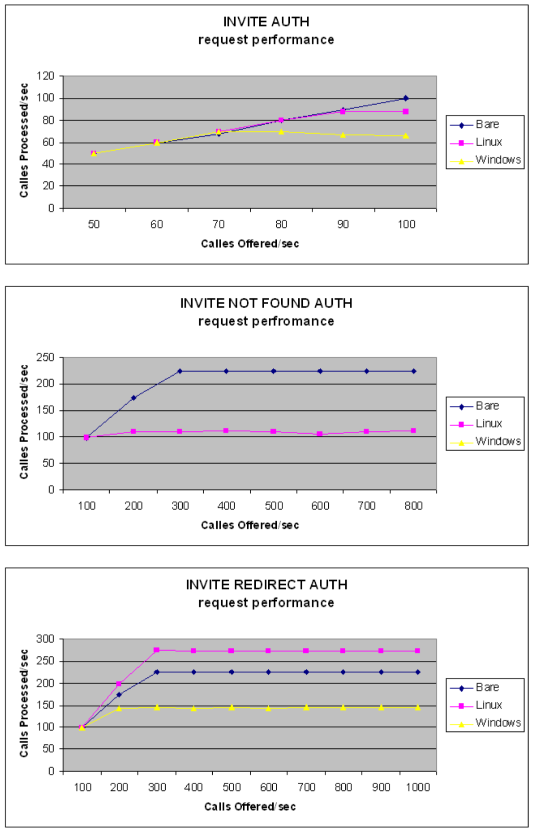


Figure 4. Throughput for invite with authentication

message with authentication credentials and the “200 OK” message. Latency for invite with authentication measures the sum of three delays: the time between the invite and “407 proxy authentication” messages; the time between the “invite with authentication” message and the “200 OK” messages; and the time between the “bye” and “200 OK” messages. For invite-not-found and redirect operations, the latency is similarly measured using the “404 not found” and “302 moved temporarily” messages.

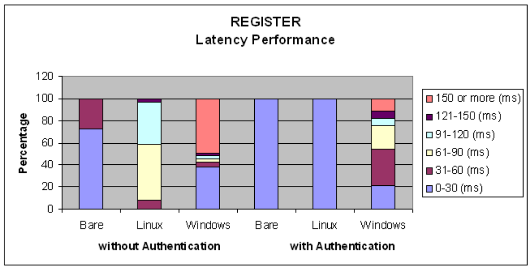
We measured the throughput and latency of a server associated with each SIP call flow. The latency for a given operation is computed by adding the respective delays between sending the relevant messages to the server and receiving their responses as described above. The throughput is the number of calls per second successfully handled with respect to the offered load, which is the number of calls per second that are generated and sent to the server. The peak throughput is the highest throughput achieved under overload while the server remains stable (and produces consistent results). To conduct the experiments, the servers were configured to operate in three configuration modes with and without authentication: registrar, proxy, and redirector. In addition, internal timings were measured by inserting timing points within the bare SIP server. Each SIP server was pre-loaded with 10,000 unique SIP username and password pairs. The call flows for register, invite-not-found, and redirect were run for a maximum of 10000 unique users, measuring the performance of each call flow with rates varying from 10 to 1000 calls/sec. The invite test call flows were run for a maximum of 5000 users with rates varying from 50 to 100 calls/sec. Each experiment was repeated a minimum of three times to ensure that the results were consistent.

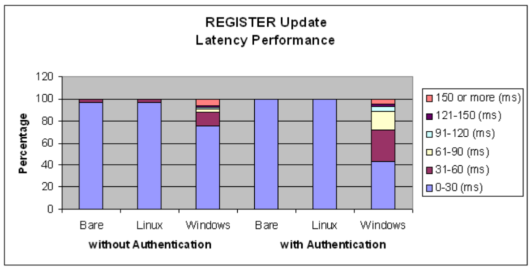
## B. *Throughput*

The throughput for the register and invite operations respectively without authentication is shown in Figs. 1 and 2. It can be seen that the peak throughput of the bare PC SIP server is always higher than that of the OS-based servers except in the case of invite redirect. The peak throughput of the bare PC server typically exceeds that of the Linux server by 50-125 calls/sec depending on the operation, although it is only 10 calls/sec more for invite and 150 calls/sec less than that of the Linux server for invite redirect. For example, the bare PC SIP server has a peak throughput of 700 calls/sec for register operations (without authentication), which is better than the peak throughput of Linux (650 calls/sec); the Windows server has a much lower peak throughput (around 200 calls/sec).

The peak throughput performance of the bare PC SIP server should be better than that of the OS-based servers, due to its simple design and the elimination of OS overhead. However, this performance advantage may be reduced or lost in certain cases due to inefficient algorithms or the lack of concurrency. The latter situation arises with the invite operation. The peak throughput of the bare PC server is only marginally higher than Linux in this case, but introducing a separate SIP task to handle an invite operation will improve performance. The apparent drop in performance of the bare PC server for invite redirect is due to a significant improvement in the performance of the Linux server in this case. Implementing Linux’s search algorithm on the bare PC SIP server should improve its performance. A more efficient search algorithm should also improve the performance for the invite-not-found operation. The peak throughput of a given server does not vary much across the three register operations since the work performed in each case is essentially the same. The increase in the peak throughput of the Windows server for register update compared to that for the other two register operations is possibly due to caching.

The results in Figs. 3 and 4 show that peak throughput of all servers is reduced as expected for both register and invite operations when authentication is added. This reduction in performance is due to the extra message overhead noted previously, and the overhead of computing and verifying the additional information needed for authentication with a message digest [8]. The negative impact of authentication on performance was also noted in [9]. There are no throughput values for the Windows server for invite-not-found with authentication since its message flow in this case could not be compared with that of the other two servers. It is evident that the peak throughput of the bare PC server with authentication shows a greater reduction versus its peak throughput without authentication compared to the OS-based servers. Adapting the approach used for authentication by Linux for the bare PC server could improve its performance.





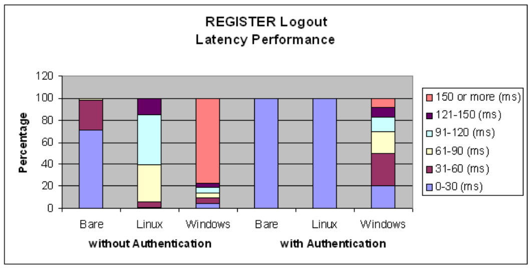
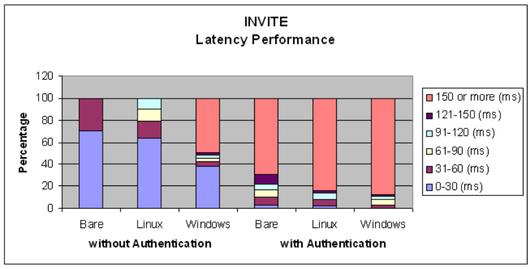
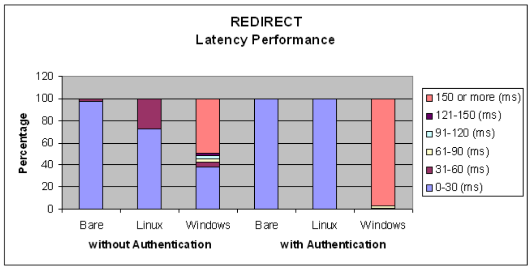


Figure 5. Latency for register with and without authentication





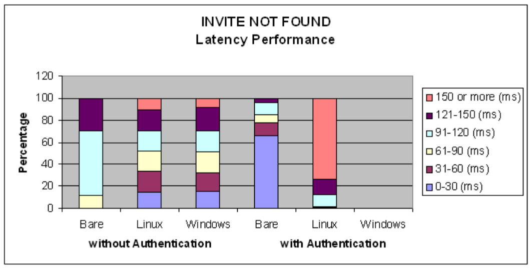


Figure 6. Latency for invite with and without authentication

## C. *Latency*

Figs. 5 and 6 compares the latencies for bare PC and OS-based SIP servers for the register and invite operations respectively with and without authentication. In most cases, the bare PC server performs better than the OS-based servers. As seen in the figure, the highest percentage of latencies for the bare PC server are usually in the 0-30 ms range, and it rarely has latencies that exceed 150 ms. The invite operation is an exception and latency performance in this case could be improved by enabling concurrency in the server as noted earlier. For all register operations and invite redirect with authentication, the latency performance of the bare PC and Linux servers is the same. Further studies are needed to determine if the approach used to implement authentication in the Linux server will improve the latency performance of the bare PC server in these cases.

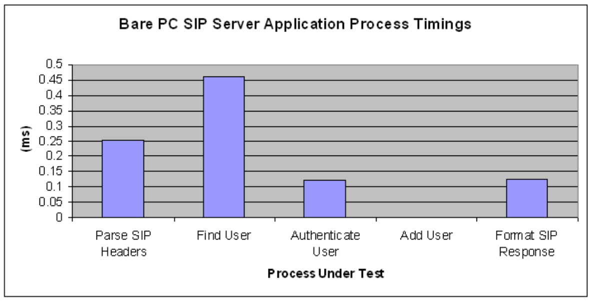


Figure 7. Internal timings for the bare PC SIP server

## D. *Internal Timings*

Fig. 7 compares average values of internal timings for the bare PC SIP server collected during the register operation under maximum load conditions. It is seen that FindUser, which searches for a given user, and ParseSIPHeaders, which processes the SIP header are the most expensive operations, although the former is twice as expensive as the latter. The least expensive operation is AddUser, which simply adds the information for a new user, and thus takes an insignificant amount of time as would be expected. The AuthenticateUser and FormatSIPResponse operations have approximately the same cost, which is about half that of ParseSIPHeaders. We conducted tests on the OpenSER server using OProfile 0.9.5 [16], which showed that the timings for the AddUser and ParseSIPHeaders operations exceed the corresponding timings on the bare PC by factors of 4 and 7 respectively.

### E. Analysis

Further insight into the results on throughput may be obtained by considering sustainable throughput, which is defined as the maximum rate of calls for which the processed call rate matches the offered call rate. Sustainable throughput reflects the extent to which a server can cope with the offered load, and it can be determined from the preceding Figs. 1-4. For example, the sustainable throughput of the bare PC server for the register, register update, and register logout operations without authentication is respectively 400, 600, and 700 calls/sec (the peak throughput for all three register operations without authentication is 700 calls/sec). It can be seen that the sustainable throughput of the bare PC server exceeds that of the Linux server for all operations without authentication except for invite-not-found when it is the same. In contrast, the sustainable throughput for the two servers for all operations with authentication is the same (or differs by a small amount). As noted earlier, in the case of peak throughput with and without authentication, the bare PC server's values are higher than those for the Linux server except for invite redirect. Thus, both sustainable and peak throughput values should be used to estimate server capacity with and without authentication.

The latency performance shown in the preceding Figs. 5 and 6 may be better understood by computing a latency coefficient $p_1*w_1+p_2*w_2+p_3*w_3+p_4*w_4+p_5*w_5-p_6$, where $p_1, \ldots, p_6$ are the latency percentages of the groups 0-30 ms, … , 121-150 ms, and > 150 ms respectively; and $w_1, \ldots, w_5$ are the weights of the first 5 groups with $0<=w_i<=1$ and $w_1+\ldots+w_5=1$. The last term with a negative sign reflects the undesirability of latencies > 150 ms. The weights $w_1, \ldots, w_5$ can be assigned based on the relative importance of the lower latency groups. For example, suppose we assign $w_1=0.55$, $w_2=0.445$, $w_3=0.004$, $w_4=0.0007$, and $w_5=0.0003$. Then the latency coefficients for register logout without authentication for the bare PC, Linux, and Windows servers are 0.496, 0.185, and -0.7. These values show that the latency performance of the bare PC server in this case is much better than that of the Linux server, whereas the performance of the Windows server is far worse than both of them. It can also be verified that the latency coefficient of the bare PC server is greater than or equal to that of the Linux server except in the case of invite with authentication and invite-not-found without authentication. As noted above, concurrency and use of a more efficient search algorithm may help to improve bare PC server performance in these cases.

## V. Conclusion

We studied the performance of a bare PC SIP server by measuring its throughput and latency for registration, proxying, and redirection with and without authentication. We also obtained internal timings for the server and compared its performance with the OpenSER server running on Linux and the Brekeke server running on Windows. The results show that the bare PC server has better performance than the Windows server and better or equal performance to the Linux server in most cases. The exceptions are throughput performance for the invite redirect operation, and latency performance for the invite operation with authentication and the invite-not-found operation without authentication, for which the Linux server is better. It is expected that the performance of the bare PC server can be improved in the latter cases by enabling concurrent processing and using more efficient algorithms. The bare PC SIP server implementation can also be modified based on internal timings to reduce the cost of the most expensive operations. Our results serve as a baseline to assess the minimal overhead associated with basic SIP server operations for both OS-based and bare PC servers, and to help improve the performance of bare PC SIP servers. These results also indicate the feasibility of deploying bare PC SIP servers in secure environments where OS-based vulnerabilities are a concern.

## References

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

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

[3] A. L. Alexander, A. L. Wijesinha, and R. Karne, "An Evaluation of Secure Real-Time Transport Protocol (SRTP) Performance for VoIP," Third International Conference on Network and System Security (NSS), pp. 95-101, 2009.

[4] G. H. Khaksari, A. L. Wijesinha, R. K. Karne, L. He, and S. Girumala, "A Peer-to-Peer bare PC VoIP Application," Proceedings of the IEEE Consumer and Communications and Networking Conference (CCNC), pp. 803-807, IEEE Press, Las Vegas, NV, 2007.

[5] R. K. Karne, K. V. Jaganathan, N. Rosa, and T. Ahmed, "DOSC: Dispersed Operating System Computing", OOPSLA '05, 20th Annual ACM Conference on Object Oriented Programming, Systems, Languages, and Applications, Onward Track, ACM, San Diego, CA, October 2005, pp. 55-62.

[6] G. Ford, R. Karne, A. L. Wijesinha, and P. Appiah-Kubi, The Design and Implementation of a Bare PC Email Server, with G. Ford et. al, 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC), 2009.

[7] K. Ono and H. Schulzrinne, The Impact of SCTP on SIP Server Scalability and Performance, GLOBECOM, pp. 1421-1425, 2008.

[8] S. Salsano, L. Veltri, and D. Papalilo, SIP security issues: The SIP authentication procedure and its processing load, *IEEE Network*, pp. 38-44, 2002.

[9] E. M. Nahum, J. M. Tracey, and C. P. Wright, Evaluating SIP server performance, in: 17th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Urbana-Champaign, Illinois, June 2007.

[10] M. Cortes, J. R. Ensor, and J. O. Esteban, On SIP Performance. *Bell Labs Technical Journal*, 9(3), pp. 155-173, 2004.

[11] C. Shen, H. Schulzrinne, and E. M. Nahum, Session Initiation Protocol (SIP) Server Overload Control: Design and Evaluation, IPTComm, pp. 149-173, 2008.

[12] V. A. Balasubramaniyan, A. Acharya, M. Ahamad, M. Srivatsa, I. Dacosta, and C. P. Wright, SERvartuka: Dynamic Distribution of State to Improve SIP Server Scalability, ICDCS, pp. 562-572, IEEE Computer Society, 2008.

[13] K. Ono and H. Schulzrinne, One Server Per City: Using TCP for Very Large SIP Servers, IPTComm, pp. 133-148, 2008.

[14] H. Jiang, A. Iyengar, E. M. Nahum, W. Segmuller, A. Tantawi, and C. P. Wright, Load Balancing for SIP Server Clusters, INFOCOM 2009.

[15] K. K. Ram, I. C. Fedeli, A. L. Cox, and S. Rixner, Explaining the Impact of Network Transport Protocols on SIP Proxy Performance, ISPASS, pp. 75-84, 2008.

[16] Oprofile-A System Profiler for Linux, July 31, 2009. [Online]. Available: http://oprofile.sourceforge.net/news/. Accessed: May 28, 2010.