Analysis of Delivery of Web Contents for Kernel-mode and User-mode Web Servers

Syed Mutahar Aaqib
Research Scholar
Department of Computer Science & IT
IT University of Jammu

Lalitsen Sharma
Associate Professor
Department of Computer Science &
University of Jammu

ABSTRACT

In this paper, the architecture of kernel-mode and user-mode web-servers and the constraints that affect their performance are studied. A set of experiments have been performed to measure and analyze performance of kernel-mode and user-mode web servers on an open source Scientific Linux CERN platform. Web servers under study include kernel-mode TUX web server and user-mode Apache web server for varying static workload sizes. The results of the experiments revealed that the performance of kernel mode web servers greatly exceeds to that of user-space web servers.

General Terms: Performance analysis

Keywords: Web servers, Web performance analysis, kernel-mode, user-mode web servers

1. INTRODUCTION

World Wide Web (WWW), in addition of being an internet service is also being used as an interface to various underlying internet services. With the exponential growth of the number of internet users and increase in the number of various internet services, the load of a web server has increased tremendously. Depending on its architecture, a web server is implemented either in user-space or in the kernel of the operating system, respectively referred to as user-mode web servers and kernel-mode web servers. The user-mode web servers include process-driven, threaded and Event-driven. Process-driven and threaded web servers are the most common, with Apache being the most popular1.Web servers that use threads include JAWS [17] and Sun Java System Web Server [22]. Web servers like Flash [12], Zeus [10] are the examples of event-driven architecture web servers. In kernel-mode web servers such as kHTTPd [24], TUX [23] and AFPA [3], HTTP server is tightly integrated with the host's TCP/IP stack. The whole of the web server is implemented within the operating system-kernel itself, thus reducing the overhead associated with the expensive transitions within the user-space. In this paper, two web server architectures, user-mode and kernel mode web servers are briefly reviewed and their comparative performance analysis is presented.

2. KERNEL-MODE WEB SERVERS

The first kernel-mode web server, kHTTPd [24], developed by Arjan van de Ven for Linux in 1999, creates one thread for each CPU on the system. Requests are divided into a number of states with a queue for each state. As the state changes, requests are transferred from one queue to the next. These states include Wait for accept, Wait for HTTP-header, Send data to user mode web server and Cleanup state. The first state "wait for accept" is handled by the operating system itself while the other queues are handled by kHTTPd. As kHTTPd has access to the data structures and functions inside the kernel, it can directly transfer the incoming request to the user-mode web server if its intended for it. Another kernel-mode web server is TUX [23] [25], which stands for "Threaded linUX HTTP layer" designed and implemented by Ingo Molnar at Red Hat. Like kHTTPd it is also a kernel mode web server but it differs in design and features. TUX takes a different approach than kHTTPd while responding to network events, Instead of using a normal socket interface from kernel mode and using non-blocking calls to examine the connections for activity, TUX hooks in to the TCP/IP-stack by changing function pointers in the internal structure representing a socket to point to functions provided by TUX so that when a network event occur it will be called directly and can respond to it. Responses from both kHTTPd and TUX are derived in a thread or interrupt-context. kHTTPd uses socket interfaces in kernel-mode while TUX uses a threaded model [18].

3. USER-MODE WEB SERVERS

Apache web server being a user-mode web server, its architecture follows a multi-process programming paradigm. When an HTTP connection request arrives, that request is first parsed by the main root process of the Apache daemon which then spawns a new process to fetch or form the HTML file. If the requested file is a static file, it is loaded from the disk else the request is forwarded to one of its modular processes. The multi-processing module allows multiple processes to prepare HTML files concurrently. When too many processes are working simultaneously, the overhead of context switching causes the performance to degrade. This can be prevented by configuring and setting a limit to the maximum number of allowed processes in Apache.

4. RELATED LITERATURE

Contemporary web servers require special techniques to handle a large number of concurrent connections. To handle thousands of such requests concurrently without degrading the performance, Pai et al.[1, 2] proposed different server architectures to handle such problems. Joubert et. al [3] reported the techniques to improve the user-space web server performance by elimination of data copies and reads, reduction of scheduling and context switching overhead due to event notification, and reduction of overall communication overhead in the socket layer, TCP/IP stack, link layer, and network interface hardware. Eliminating copies and reads for a Web transaction offers significant performance improvements for large responses. However it is difficult to avoid it in user-mode web server where the data that is to be sent resides in the file system cache unless the data is already mapped into the user-mode address space. To further reduce the overhead of event notification, Pai et al. [1] developed

[1] URL:www.news.netcraft.com/archives/category/web-server-survey as accessed on 02 Nov 2010

a mapping between threads and requests as multiple process/thread (MP) or single process event driven (SPED). Web servers such as Zeus [10], IIS [11] use a SPED model. Flash also uses a SPED model for cached content, using only one thread for serving cache hits. The Windows 2000 APIs implementing zero copy data transfer and efficient event notification are described in [13]. Various researchers have also proposed modifications in operating system interfaces and mechanisms for efficient notification and delivery of network events to user-space servers [4, 5, 6, 7] thus reducing the amount of data copied between the kernel and the user-space [8], reducing the number of kernel boundary crossings and a combination of the above [9]. Some of the researchers have tailored existing socket API's and implemented new APIs with newer web servers in mind [14] while others [15] have redesigned and improved the TCP/IP stack for Web server workloads to efficiently manage short-lived connections. To reduce the event notification overhead, Banga et al. [16, 17] optimized the implementation of select () and poll () system calls thereby improving the already existing interfaces and their implementation. Banga et al. [19] implemented a scalable version of select() system call and the kernel routine that allocates a new file descriptor, They later proposed an entirely new event delivery mechanism for UNIX to overcome inherent limitations in the scalability of the select() system call [20]. More recent work by Brecht et al. [21] focuses on the scalability of the accept() system call.

5. EXPERIMENTAL TEST-BED

The test environment consists of two clients connected to a server via a 100Mbits/s Ethernet switch.

5.1. CLIENT & SERVER CONFIGURATION

The two client machines are running Scientific Linux CERN 5 (2.6.18). Each machine has a single 2.0 GHz Intel processor with 1 GB of RAM and uses "RAM-disk" (a virtual disk in Linux) of 128 MB for collecting statistics during testing using httperf [26] workload generator. The server machine in our test environment is an Intel Core 2 Duo 2.4 GHz machine also running Scientific Linux CERN 5 (2.6.18) with 2 GB of RAM. The hardware configuration is identical to that of the clients.

5.2. PERFORMANCE TUNING

The number of available file descriptors was increased (from 1024 to 32,678) and the limit of the local port range was also increased. TCP TIME_WAIT recycling was enabled to free up sockets in a TIME_WAIT state more quickly, thus allowing clients to generate and sustain high request rate. Also, all the non-essential processes and services on the server as well as client machines were disabled.

5.3. CLIENT BENCHMARK PROGRAM

The httperf [26] is an open source benchmark developed by David Mosberger at Hewlett-Packard Research Labs. The httperf benchmark is a flexible HTTP client that requests a file from a web server multiple times and for number of parallel threads and then prints out detailed statistics. Its source code is modified in order to print the server response rate information more frequently. Thus the output of the httperf provides information about TCP (TCP connection rate) and HTTP (request and reply rate) behaviors on second by second basis.

5.4. WEB SERVERS TESTED

The comparison of user-mode and kernel-mode HTTP servers was performed by examining some widely popular web servers in both categories. In the experiments Apache 2.2.3 was used as a user-mode web server running at port 8080 and TUX 3.0.0, being a kernel-mode Web server running at port 80. Both the web servers were restarted between each experiment.

5.5 EXPERIMENTAL DESIGN

Subject and Metrics

The tests were focused on several key metrics: Workload file size, TCP connection rate, HTTP request rate, HTTP reply rate, HTTP reply time, Throughput. Other factors i.e. network I/O and the CPU utilization was taken into account as well. Workloads were categorized into small, medium and large-file sizes. For requests involving small files (1KB), most of the time would be spent on setting up the TCP connection and processing the request. The number of requests per second thus gives good information on the performance. For larger file sizes most of the time would be spent on the task of reading file data from the file system and sending it to the TCP/IP-stack. So in this case throughput is chosen to be a metric to depict the performance differentiation between web servers. Another metric is to measure how a web server handles large number of concurrent connections; this will show whether the connections are handled in an efficient way or if processing time goes just in administrating and sustaining the connections. Table 1 lists the connection rates for varying workloads.

Web Server	Workload Size	Connection Rate
TUX	Small (1kb) Medium(320kb) Large (900kb)	500,1000,2000, …, 20000 200,400,600,…,2000 10,20,40,80,… 640
Apache	Small (1kb) Medium(320kb) Large (900kb)	500,1000,2000, …, 20000 200,400,600,…,2000 10,20,40,80,… 640

Table 1: Connection rates for varying workload sizes

6. RESULTS AND DISCUSSIONS

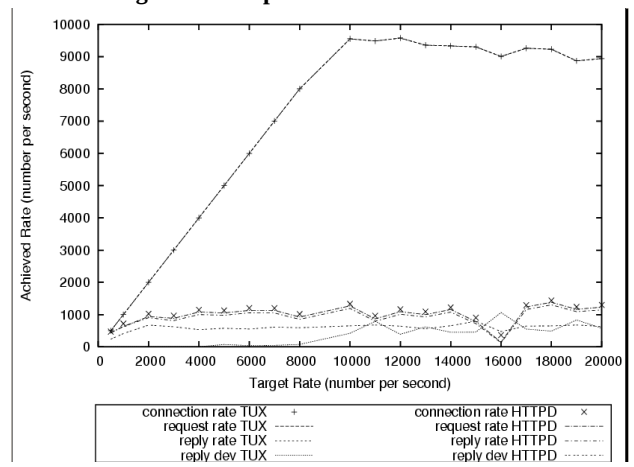
The following table summarizes the performance results for both kernel and user mode web server under varying static workload sizes when the web server attains its saturation level by listing the average number of TCP, HTTP requests, HTTP responses and throughput per second.

Table 2: Performance measurements of kernel and user-mode web servers at saturation levels

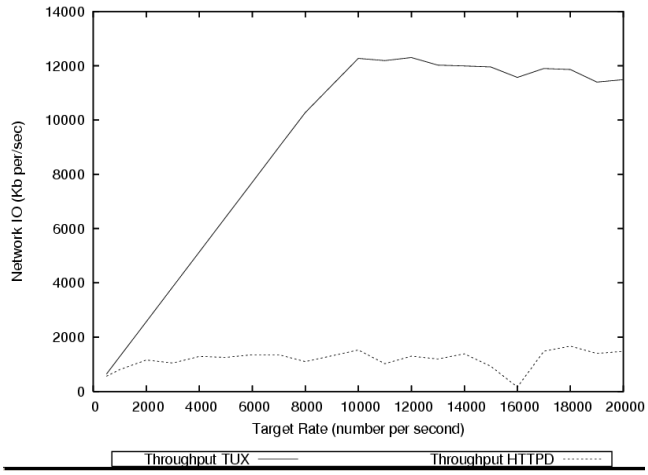
Web Server	Workload Size	TCP connections /sec	HTTP requests /sec	HTTP responses /sec	Throughput (kb/sec)
TUX 3.0.0	Small	10,100	9700	9700	400-12300
	Medium	1000	1000	1000	64058-320309
	Large	320	320	300	9551-305395
Apache 2.2.0	Small	1500	1381	1300	400-1670
	Medium	699	641	634	64058-202086
	Large	346	258	245	9551-233933

Figure 2 shows the comparative results for small-workload file using Apache and TUX web server. In figure 2(a), there are eight sets of data plotted. It presents the average number of TCP connections established per second, average HTTP request and response rates and also the reply deviation for both TUX and Apache httpd server. The time unit for each point on the graph is 2 minutes as it is the test duration for each test in the experiment.

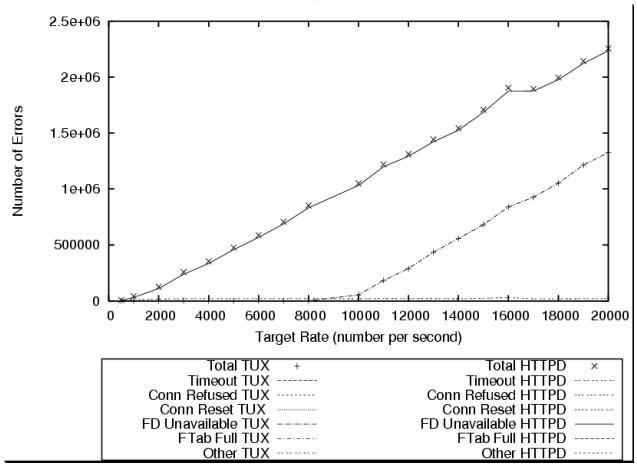
Figure 2: Comparison of results of 1KB



2(a)



2(b)



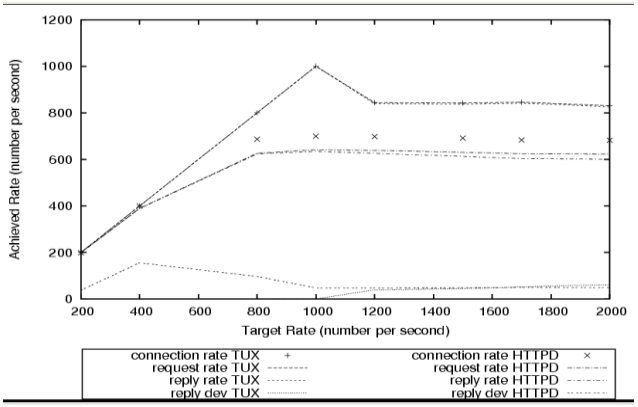
2(c)

Figure 2(b) shows the Network IO for the 1 KB file. It shows that the total throughput increased from 400 KB/sec to up to 12300KB/Sec for TUX and from 400-1670 KB/sec for the Apache web server after which it degrades.

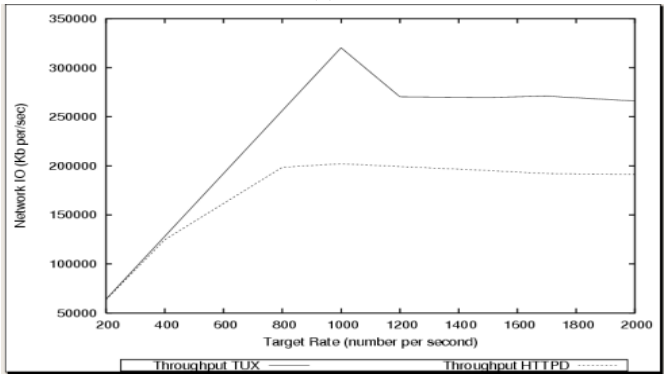
Figure 2(c) shows the error rate for both Apache and TUX for 1KB workload. As it is obvious, the error rate for Apache is more than that of the kernel mode TUX. Also the target rate after which the error rate increases for TUX is 10,100 and for Apache it is 1500. Similar graphs are shown in Figure 3 and 4 for the medium and large workload file results for both TUX and Apache web server.

Comparison results of kernel mode TUX and user mode httpd for medium and large workload file.

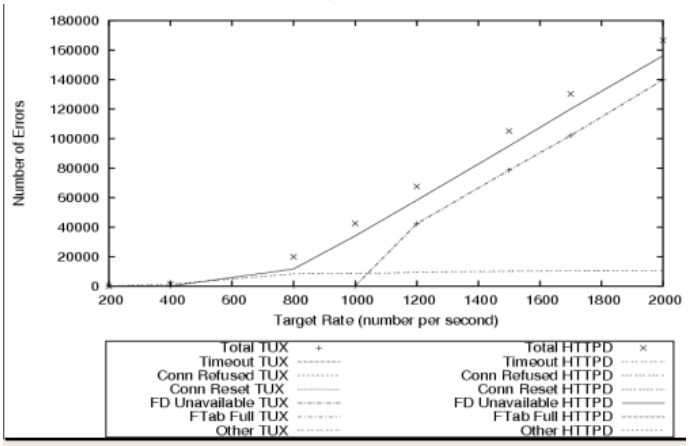
Figure 3: Comparison of results for 320 KB file.



3(a)



3(b)



3(c)

Figure 5 shows the connection lifetime and the connection establishment times and reply rate/sec for TUX and Apache for different workloads.

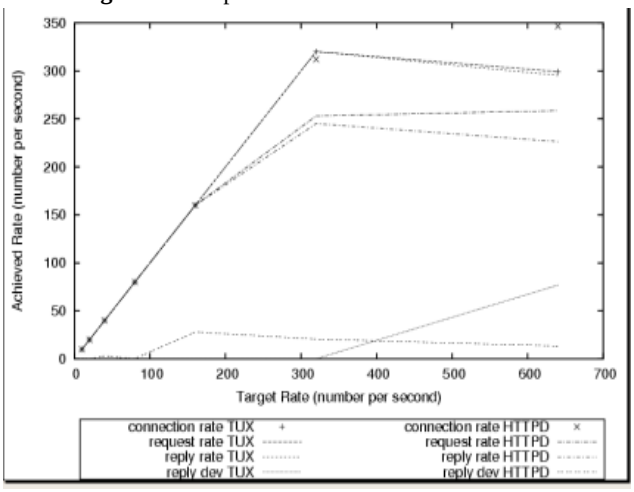
Figure 5(a) shows that the connection establishment time for 1KB workload TUX web server is negligible up to the target request rate of 8000 reqs/sec after which it increases slightly up to 10 ms and the connection lifetime which is maximum up to 255 ms before the saturation level then increases more than 3000 ms. This occurs because the server is overloaded at that time and has attained its saturation level.

Figure 5(b) plots the reply rate per second for time taken in the 1kb file experiment. It shows that there is a uniform growth of the reply rates up to the saturation level; the intervals are separated by the idle time period during which the graph has no value. In this graph it also shows the saturation point is up to 10,000 requests

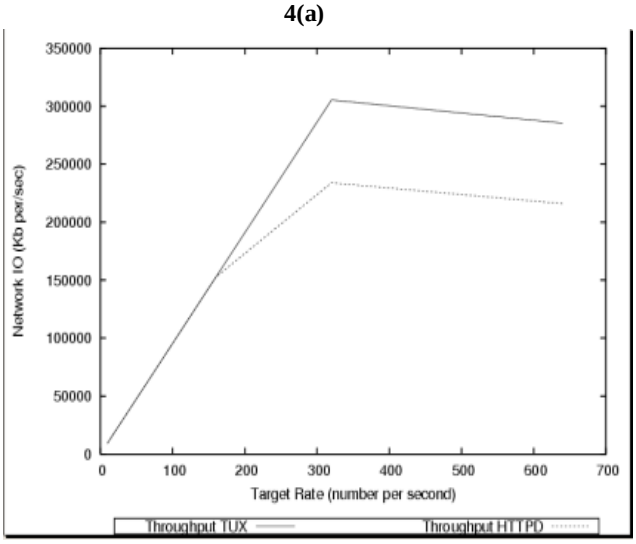
per second which took up to 1500 seconds. Hence the time it took for the server to saturate for a 1kb file in a kernel based server is 1500 seconds.

Similarly, Figures (6,7) show the connection times and reply rate graphs for medium and large workload files using TUX web server, Whereas graphs in figures (8,9,10) show these results for small, medium and large workload files using Apache web servers..

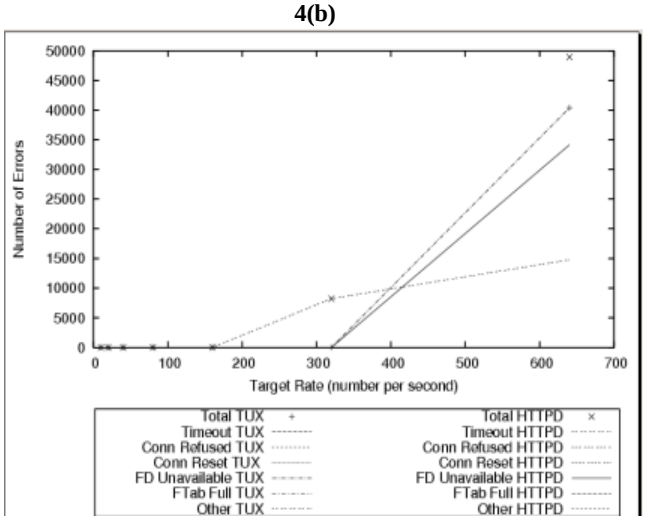
Figure 4: Comparison of results for 900 KB file.



4(a)

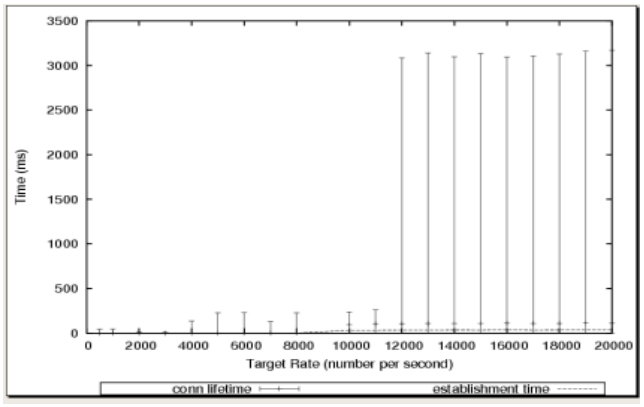


4(b)

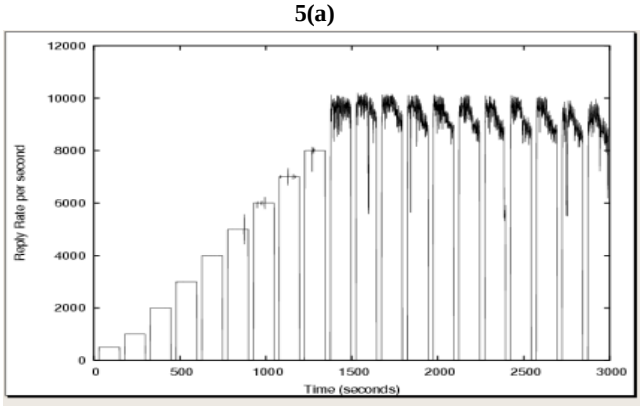


4(c)

Figure 5: Connection times and reply rate for 1KB workload file for TUX Web server.

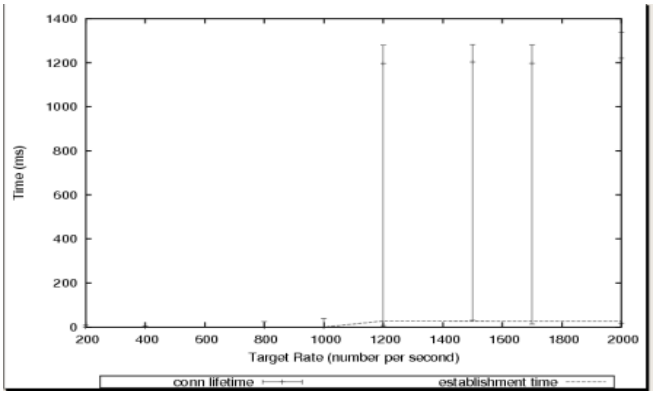


5(a)

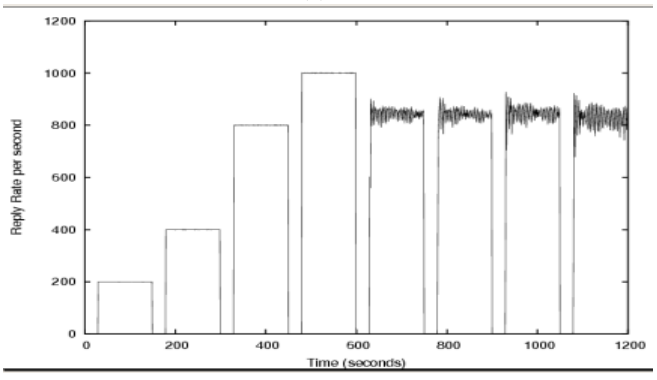


5(b)

Figure 6: Connection times and reply rate for 320 KB workload file for TUX Web server.

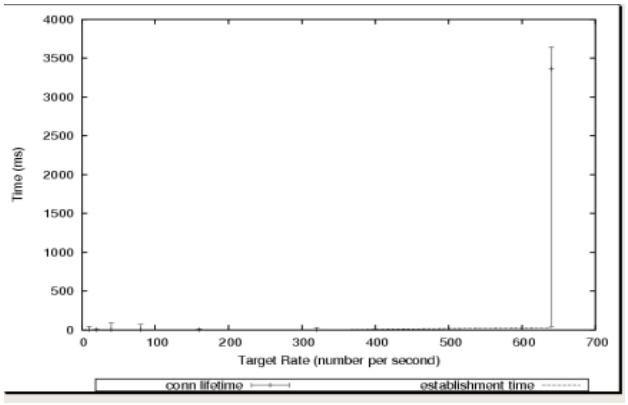


6(a)

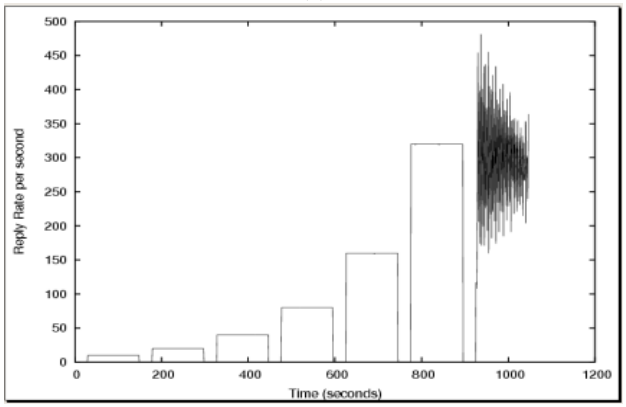


6(b)

Figure 7: Connection times and reply rate for 900KB workload file for TUX Web server.

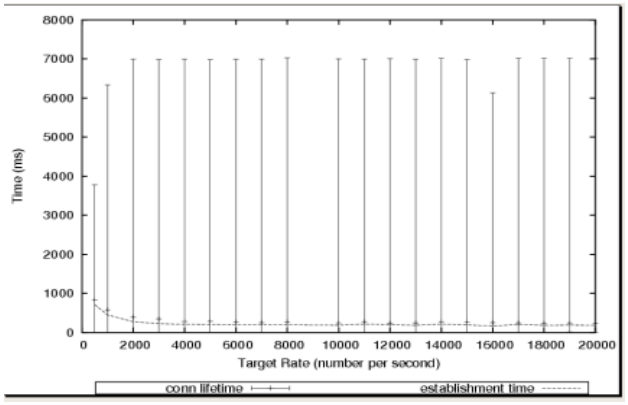


7(a)

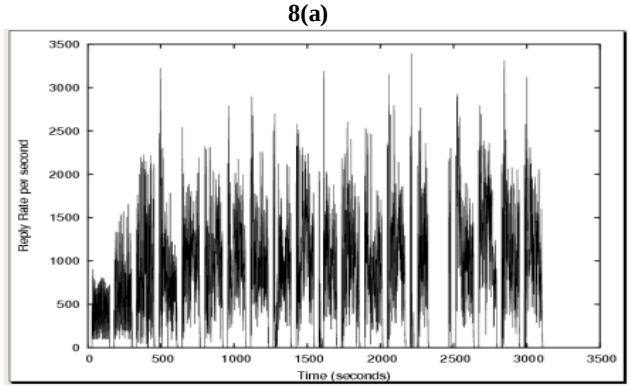


7(b)

Figure 8: Connection times and reply rate for 1KB workload file for Apache web server.

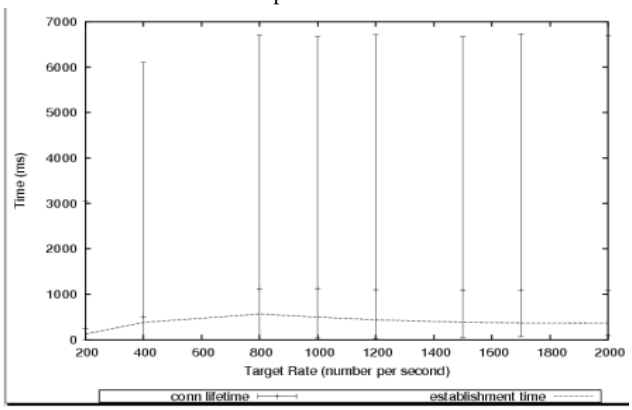


8(a)

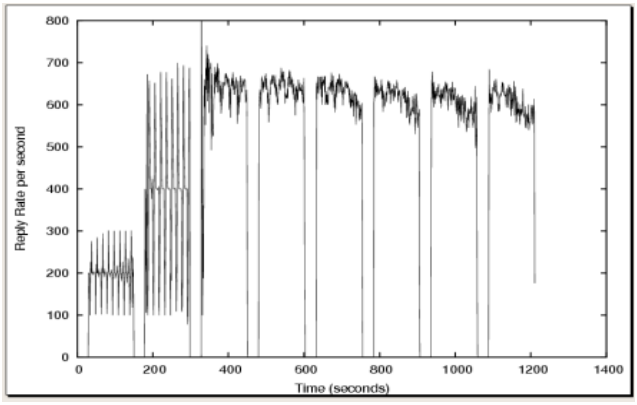


8(b)

Figure 9: Connection times and reply rate for 320 KB workload file for Apache web server.

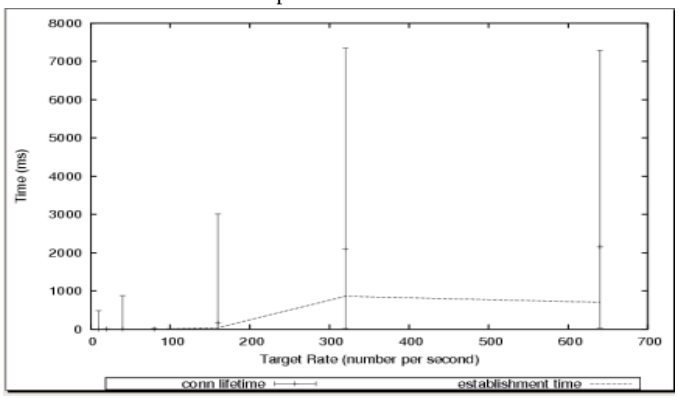


9(a)

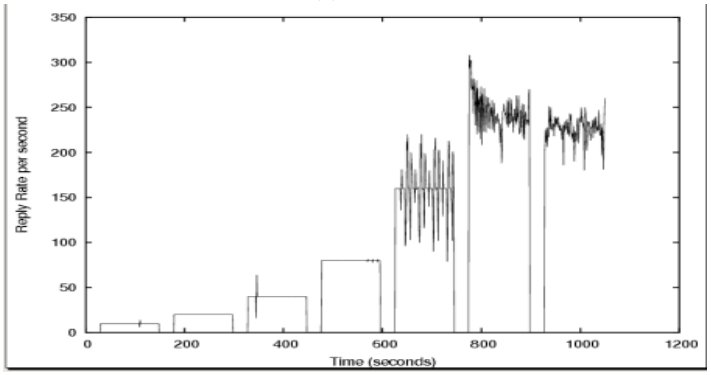


9(b)

Figure 10: Connection times and reply rate for 900 KB workload file for Apache web server.



10(a)



10(b)

7. CONCLUSION

The measurement results revealed that the kernel-mode web servers have a significant performance improvement over user-mode web servers on Linux platform. Here the metric that improved most in kernel mode is the throughput and achieved rate which means that a larger number of requests per second can be handled; this was found for smaller files where the kernel mode web server on Linux was up to ten times faster than user mode web server. As the size of the workload file increases this difference decreases up to 32%. The reason for the big improvement in achieved rate and throughput is because a kernel mode web server avoids process scheduling. Throughput is improved in the kernel mode web servers but the difference decreases as we use larger workload files. The main reason behind the higher throughput from the kernel mode web server TUX is that that it avoids data copies and reads, scheduling overhead, context switching and also reduction of overall communication overhead in socket layer. Thus, it results in handling large number of requests per second and also helps in increasing the throughput. By comparing absolute performance between the two types of web servers we have made this observation that kernel mode web server is faster than user-mode web servers for static workloads. Thus, kernel mode web servers should be worth considering from a performance point of view even though kernel mode web servers do not provide security and reliability benefits as in user-mode web servers.

8. ACKNOWLEDGMENTS

The authors are thankful to Prof. Devanand, Head, Department of Computer Science and IT, University of Jammu, for his kind support.

9. REFERENCES

[1] Vivek S. Pai, Peter Druschel, and Willy Zwaenepoel. Flash: An efficient and portable web server. In Proceedings of the USENIX 1999 Annual Technical Conference, June 1999.

[2] M. Welsh, D. Culler, and E. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In Proceedings of the Eighteenth Symposium on Operating Systems Principles, October 2001.

[3] Philippe Joubert, Robert B. King, Richard Neves, Mark Russinovich, and John M.Tracey. High-performance memory-based web servers: Kernel and user-space performance. In USENIX Annual Technical Conference, General Track, pages 175–187, 2001.

[4] N. Provos and C. Lever. Scalable network I/O in Linux. In Proceedings of USENIX Annual Technical Conference, FREENIX Track, June 2000.

[5] Abhishek Chandra and David Mosberger. Scalability of Linux event-dispatch mechanisms. In Proceedings of the 2001 USENIX Annual Technical Conference, pages 231–244, 2001.

[6] G. Banga, J.C. Mogul, and P. Druschel. A scalable and explicit event delivery mechanism for UNIX. In Proceedings of the 1999 USENIX Annual Technical Conference, June 1999.

[7] Louay Gammo, Tim Brecht, Amol Shukla, and David Pariag. Comparing and evaluating epoll, select, and poll event mechanisms. In Proceedings of 6th Annual Linux Symposium, july 2004.

[8] Vivek Pai, Peter Druschel, and Willy Zwaenepoel. IO-Lite: A unified I/O buffering and caching system. ACM Transactions on Computer Systems, Vol. 18:37–66, 2000.

[9] Erich Nahum, Tsipora Barzilai, and Dilip Kandlur. Performance issues in WWW servers. IEEE/ACM Transactions on Networking, Vol. 10, February 2002.

[10] Zeus Technology Ltd. Zeus web server. URL: http://www.zeus. com.

[11] Microsoft Corporation. IIS, Internet information services features.http://www.microsoft.com/windows2000/guide/server/features web.asp.

[12] Flash web server URL: http://www.cs.princeton.edu/ ~vivek/ flash/.

[13]James C. Hu, Irfan Pyarali, and Douglas C. Schmidt. High performance Web servers on Windows NT: Design and performance. In USENIX, editor, The USENIX Windows NT Workshop 1997, August 11–13, 1997. Seattle, Washington, pages 149–149, Berkeley, CA, USA, August 1997. USENIX.

[14] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Better operating system features for faster network servers. In Proceedings of the Workshop on Internet Server Performance (held in conjunction with ACM SIGMETRICS '98), Madison, WI, June 1998.

[15] Jeffrey C. Mogul. Operating systems support for busy internet servers. Technical Report Technical Note TN- 49, Digital Western Research Laboratory, Palo Alto, CA., May 1995.

[16] Gaurav Banga, Jeffrey C. Mogul, and Peter Druschel. A scalable and explicit event delivery mechanism for UNIX. In Usenix Annual Technical Conference, pages 253–265, 1999.

[17] The JAWS adaptive web server, URL: http://www.dre.vanderbilt.edu/JAWS

[18] Vivek S. Pai, Peter Druschel, and Willy Zwaenepoel. IOlite: A unified I/O buffering and caching system. In Operating Systems Design and Implementation (OSDI '99), pages 15–28, 1999.

[19] Gaurav Banga and Jeffrey C. Mogul. Scalable kernel performance for Internet servers under realistic loads. In Proceedings of the 1998 USENIX Annual Technical Conference, New Orleans, LA, 1998.

[20] Gaurav Banga, Jeff Mogul, and Peter Druschel. A scalable and explicit event delivery mechanism for unix. In Proceedings of the USENIX Annual Technical Conference, Monterey, CA, June 1999.

[21] Tim Brecht, David Pariag, and Louay Gammo. accept()able Strategies for Improving Web Server Performance. In Proceedings of the 2004 USENIX Annual Technical Conference, Boston, MA, 2004.

[22] Sun Java System web server, URL: http://docs.sun.com/app/docs/coll/1308.3

[23] Tux reference manual, URL: http://www.redhat.com/docs/manuals/ tux/TUX-2.1-Manual/.

[24] khttpd web server, URL: http://www.fenrus.demon.nl/

[25] Ingo Molnar. TUX: Threaded linUX http layer. URL: http://people.redhat.com/mingo/ TUX-patches/.

[26] David Mosberger. httperf: A Tool for Measuring Web Server Performance. URL: http://www.hpl.hp.com/personal/David_Mosberger /httperf.html