2009 11th IEEE International Conference on High Performance Computing and Communications

A Study of Bare PC Web Server Performance for Workloads with Dynamic and Static Content

Long He, Ramesh K. Karne, Alexander L. Wijesinha, and Ali Emdadi
Department of Computer & Information Sciences
Towson University
Towson, MD 21225 USA

Abstract—**Bare PC applications do not use an operating system or kernel. The bare PC architecture avoids buffer copying, minimizes interrupts, uses a single thread of execution for processing network packets, and incorporates novel scheduling to minimize CPU utilization. We design a bare PC Web server that can serve both dynamic and static content. Measurements of response time, connection time and throughput for workloads containing requests for dynamic and static content indicate that the server has better performance than the Apache and IIS Web servers. For example, the bare PC server has a maximum request rate that is twice that of the Apache and IIS servers when serving dynamic content for small dataset sizes. Furthermore, at capacity the CPU utilization of the bare PC server is 1/5th that of the other servers. The bare PC server can also sustain a higher maximum request rate for dynamic pages with a given request rate for static pages. The studies demonstrate that the performance of the bare PC server when serving dynamic content is limited only by the latency of the database server.**

Index Terms—**Application Object, Bare PC, Dynamic Content, Performance, Web Server.**

I. Introduction

A bare PC runs applications directly over the hardware without an operating system (OS) or any type of kernel installed on the machine. Since only the required functionality for running on the hardware is implemented, applications can perform better on bare PC systems than on OS-based systems. Bare PC servers provide an alternative approach to designing high-performance servers without incurring any OS-related overhead. Self-contained self-executing bare computing applications may be carried on a portable storage medium such as a USB flash drive and run on a PC without using an OS or a hard disk.

In previous studies, a bare PC Web server that can serve only static content was shown to perform significantly better than the popular Apache and IIS servers, and the high-performance Tux server when running on an ordinary desktop [4, 5]. However, workloads of Web servers contain a mix of requests for dynamic and static content. To better characterize bare PC Web server performance, we extend the design of the existing bare PC Web server enabling it to handle both dynamic and static pages, and conduct experiments to measure its performance. The results are compared with those for Apache and IIS Web servers running the same workloads (we are unable to compare the bare PC Web server with the Tux server since it only handles static Web pages).

The rest of this paper is as follows. Section 2 provides a brief overview of related work. Section 3 describes the design of a bare PC Web server that can serve dynamic and static content. Section 4 contains the results of experiments that were conducted to study server performance, and Section 5 contains the conclusion.

II. Related Work

Many attempts have been made to address the growth in size and complexity of the OS, and to improve performance. Examples include Microkernel [2] and Exokernel [3] that move OS functionality to application domains and eliminate OS abstractions; OSKit [8], a complete set of OS components; and TinyOS [9], an operating system designed for embedded sensor networks. In addition to streamlining the OS, many techniques including new socket functions, per-byte and per-connection optimizations [14] and caching systems [13] have been used to improve Web server performance; and architectural designs such as [15] have enabled high-performance portable Web servers to be built. Linux-based Tux is an example of a Web server that incorporates several optimizations to significantly improve throughput under conditions of high load [1]. More recently, a comparison of OS-based Web servers using static workloads has shown that event-driven and hybrid pipeline-based architectures perform better than a thread-per-connection design [16]. In contrast to conventional systems, a bare PC Web server optimizes performance by eliminating the OS layer completely and completely dedicating processor and memory resources to the application.

III. Web Server Design

Bare PC Web server design is based on the dispersed operating system computing (DOSC) concept [10] in which an application object (AO) [12] contains the code required to boot, load and execute applications on the hardware. The AO manages memory, tasks and execution flow, and communicates directly to the hardware through an API [11]. A single AO may consist of multiple programs to implement a complex user application or several applications. Real memory is used by an AO so that no virtual memory or paging is required, and minimization of interrupts and data copying are used to improve performance.

The Web server runs on any IA-32 (Intel Architecture 32-bit) compatible PC. It uses 512 MB of memory. The code has less than 10K executable C++ statements including the code for the hardware interfaces but excluding part of the Ethernet driver that has about 1500 lines of assembly code. The executable is 200K bytes not including the boot code. There are only two interrupts: a hardware timer interrupt and a network card transmit interrupt. To optimize performance, the implementations of the HTTP, TCP and IP protocols and the Ethernet driver are lean and tightly integrated with the server application.

A. Web Server Operation

Fig. 1 shows the internal design flow of the bare PC Web server. The numbered labels describe server operation. Scheduling involves only four types of tasks: MAIN (idle task), RCV, HTTP, and PHP. The RCV task has higher priority than the HTTP and PHP tasks. When a new packet arrives (1), the RCV task runs (2), reads data from the UPD (upload program descriptor) buffer and validates the packet (3). In case of TCP, the IPHandler() (4) and TCPHandler (5) process the packet. The latter creates a TCP control block (TCB) entry and updates the state of a given request (6). Each request has a separate TCB entry during its lifetime. When the RCV task terminates, control returns back to the MAIN task (7). If a received packet requires that a response be sent, the TCPHandler() calls SendMiscPkt() (8), which does the relevant TCP, IP, and Ethernet processing (9). The DPD (download program descriptor) pointers (10) are then updated before returning to the MAIN task (11).

When a GET() arrives from a client (12), the TCPHandler() parses the request, pops an HTTP task from the HttpStack and inserts it into the Run List (13). In the MAIN task, if there is an HTTP or PHP task in the Run List, it will be scheduled to run (14) if the RCV task is not running. When an HTTP task runs, it sends packets by calling SendNPkt() (15). When all packets are successfully sent, control returns to the MAIN task (16). There can be many HTTP tasks running in the system concurrently that follow the same thread of execution as described. When an HTTP task has to wait for a client response, it will call Suspend() to return control to the MAIN task and resume when the response is received.

In case of a dynamic Web page request, two linked TCB entries are used to store state information: one for the HTTP request, and the other for the PHP request to the DB server. A PHP task is inserted into the Run List when the TCPHandler() detects a request for a PHP file (17). If more than one packet has to be sent, the PHP task calls SendNPkt() to send data (22). When the response data arrives from the DB Server, it is stored in the DB Buffer (18). The PHP task parses the requested PHP file and dynamically modifies it to include the data from the database (19). This file is stored in the DPD (20). Once the file is ready to be sent, the PHP task inserts an HTTP task and sets up pointers for the outgoing data in the TCB (21). The HTTP task sends the data to the client in the same manner as for a static page request. When the PHP task is waiting for a response from the DB Server, it suspends itself and returns to the MAIN task (24). Upon request completion, a TCB entry is deleted and the task is replaced in its stack (23).

B. Message Exchange

Fig. 2 shows the message exchange between the Web server and the client consisting of handshake, file transfer, and disconnection phases. It also shows the message exchange (based on MySQL protocol v10) between the Web server and the DB server to handle an HTTP request for a dynamic Web page consisting of a connection phase for exchange of a greeting message and password signature, and a tabular response phase during which data is transferred. When a GET request for a PHP file arrives, a TCP connection is established with the DB server. The DB server then sends a “Greetings” message to the Web server with a salt value. The latter is used to generate a password signature which is sent in a “Password” message. Additional messages as shown in the figure are exchanged before the data is sent.

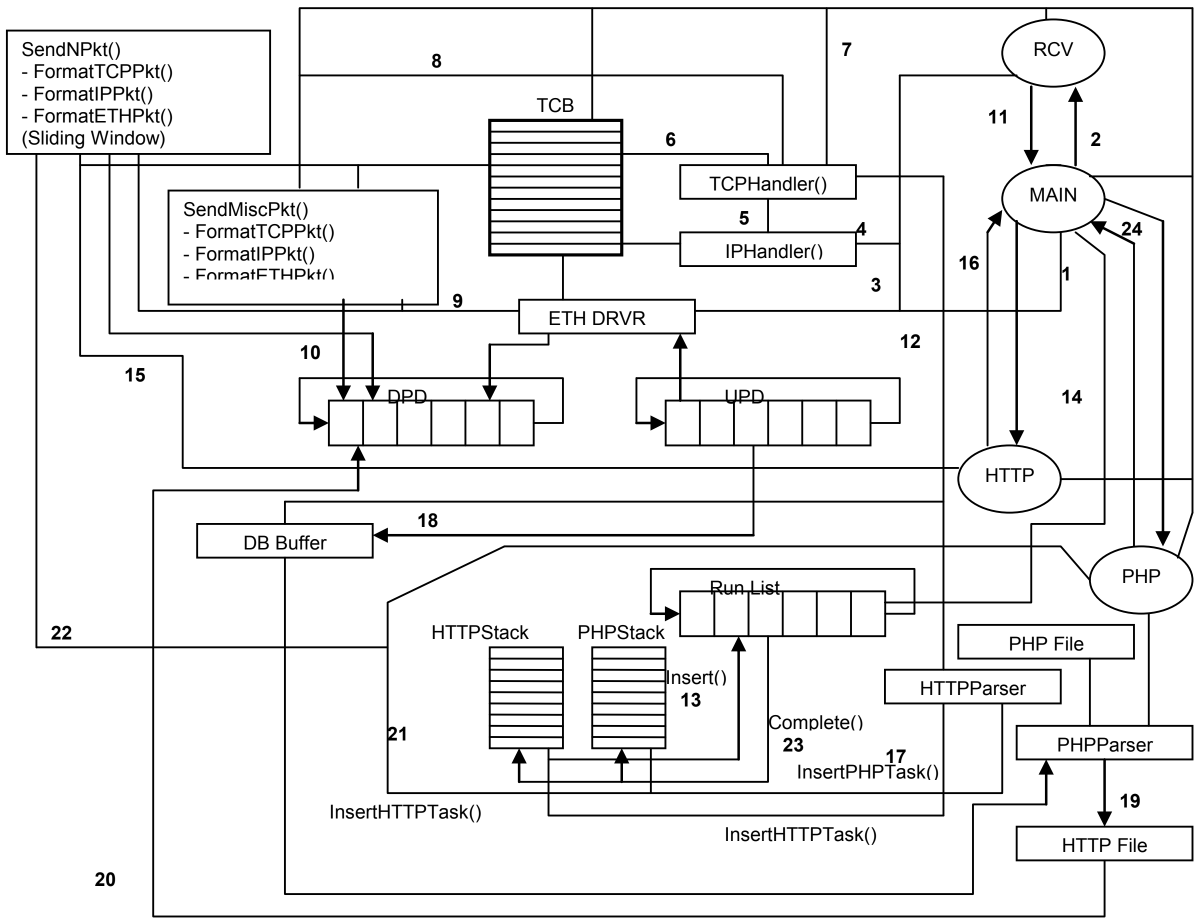

Fig. 1. Internal Design Flow of the Bare PC Web Server

IV. PERFORMANCE MEASUREMENTS

We conduct several experiments to study the performance of the bare PC Web server and to compare its performance with the Apache and IIS Web servers when the latter are tuned for maximum performance. The servers (including the DB server) and clients run on a 2.4 GHz Dell Optiplex GX260 with 512MB memory, which is a commodity single-CPU PC. The clients and the Apache server run Red Hat Linux 2.4.20-8. The DB servers run MySQL server version 5.0 on Windows XP (for bare PC and Apache) and SQL Server 2005 (for IIS). For all experiments involving dynamic Web pages, the client PCs, Web server and DB server are connected by a 100 Mbps Netgear FS 608.v2 Ethernet switch. A gigabit Ethernet switch was not used since the throughput in this case is limited not by network bandwidth but by the latency of the DB server (see Sections A and F). The HTTP version is 1.1. The tools used for workload generation are http_load-12mar2006 [6, 17] for static content and httperf-0.8 [7] for dynamic content.

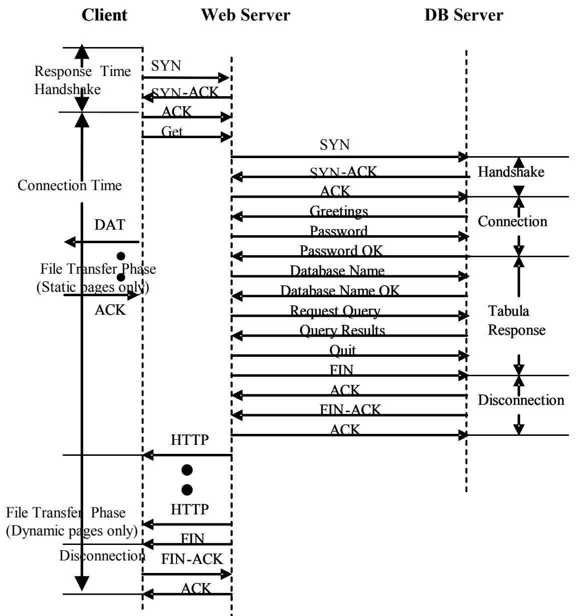

Fig 2. Message Exchange

The throughput is the average number of bits per second transferred by the Web server. The response time is the average time to complete the TCP handshake, and the connection time is the average time for the server to transfer the HTTP data to the client and close the connection (Fig. 2). The connection time for a dynamic Web page includes the time to connect to the database and retrieve the data. The dataset size referred to below is the size of the file that is transferred to the client by the Web server after it receives the tabular response from the DB server.

A. *Static Web Pages*

Our previous studies of a bare PC Web server capable only of handling only static pages used a 100 Mbps switch [4, 5]. To determine the capacity (i.e., maximum number of supported requests/sec) of the extended Web server design when serving only static Web pages we used a gigabit Ethernet switch. The response and connection time respectively for the IIS and bare PC servers for up to 2000 request/sec with a 3593 byte file are shown in Fig. 3 and Fig. 4. We only compare these two servers as Apache's performance degrades rapidly when the rate exceeds 2000 requests/sec (for rates up to 1000 requests/sec, the response time of the servers are similar). The connection time (after the handshake) is dominated by the delay for the GET request and the disconnection time. We found that the capacity of the bare PC server is 6000 requests/sec compared to 3000 requests/sec for the IIS and Apache servers. However, the response time of the bare PC server only remains stable until 5000 requests/sec and rapidly increases thereafter. The connection time for the server shows similar behavior except that it only increases slightly after 5000 requests/sec because of the small file size. At capacity, the CPU utilization of the IIS and Apache servers reaches 99%. In contrast, the bare PC server CPU utilization is respectively 41% and 82% for 3000 and 6000 requests/sec, and approximately linear for up to 5000 requests/sec.

B. *Dynamic Web Pages*

We studied the performance of the bare PC Web server when serving only dynamic Web pages with a dataset size of 211 bytes. In this case, the original PHP file is 469 bytes, and the actual data returned by the DB server consists of 11 bytes within a 414-byte TCP Payload. The Web server reformats the PHP file resulting in the 211-byte HTTP file (plus a 151-byte HTTP header) that is transferred to the client. The capacity of the bare PC Web server is 1000 dynamic page requests/sec with a corresponding response time of 66.7 ms but its performance starts to degrade when the request rate exceeds 800 requests/sec. In contrast, the capacity of the Apache and IIS servers is 600 requests/sec and the corresponding response times are 1812 ms and 2189 ms respectively with a significant increase in response and connection times after 400 requests/sec. The server response and connection times for up to 400 dynamic page requests/sec are shown in Fig. 5 and Fig. 6 respectively. The similarity in performance of the bare PC and IIS servers is because the MySQL server requires the bare PC server to open a new TCP connection for each request whereas the SQL server maintains persistent TCP connections to the IIS server. We also observed that the CPU utilization of the Apache and IIS servers reaches 99% at 600 requests/sec (with 427 threads at capacity) and 500 requests/sec respectively, while the bare PC server's CPU utilization is only 17.7% for 600 requests/sec and 29% at capacity.

C. *Dynamic and Static Web Pages*

To compare performance of the servers with requests for a mix of static and dynamic Web pages, we measured the response times with 1000 static page requests/sec for a file size of 3593 bytes and varying dynamic page request rates (up to 400 requests/sec) for a dataset size of 211 bytes. As shown in Fig. 7, the Apache server and IIS server reach their capacity with increased response times at 300 and 400 dynamic page requests/sec respectively, but the bare PC server's response times for these workloads are stable. Similar behavior was observed with connection times. We also measured the capacity of each server for dynamic page requests corresponding to a given static page request rate (for the above file and dataset sizes). The results shown in Fig. 8 indicate that for all static page request rates, the bare PC server has higher capacity than the other two servers.

D. *Dataset Size*

The response and connection times shown in Fig. 9 and Fig. 10 respectively correspond to fixing the dynamic page request rate at 100 requests/sec for small (211 bytes), medium (10,217 bytes) and large (19,248 bytes) dataset sizes respectively. These results suggest that the bare PC server will perform better than the Apache and IIS servers for dynamic Web page requests irrespective of the dataset size. The response and connection times for the bare PC server with a varying dynamic page request rate for the same (small, medium, and large) dataset sizes are shown in Fig. 11 and Fig. 12 respectively. We found that for dataset sizes up to about 10K bytes, the response and connection times of the bare PC server are moderate for all request rates. For larger dataset sizes (close to 20K bytes), the connection time is large for all request rates and the response time increases significantly when the request rate exceeds 100 requests/sec. This behavior is to be expected since the DB server has more data to send when the dataset size is large. Again, the results suggest that DB latency limits the performance of the bare PC Web server.

E. *Throughput*

Fig. 13 shows server throughput with dynamic page requests for a fixed dataset size of 211 bytes when the request rate is varied. The bare PC server throughput increases linearly until 900 requests/sec, whereas the Apache and IIS server throughput increases linearly until 500 requests/sec; these rates are close to the respective dynamic content capacities of the servers noted earlier. Fig. 14 shows similar throughput behavior when the workload consists of 1000 static page requests (file size=3593 bytes) and a varying number of dynamic page requests (dataset size=211 bytes). The bare PC, Apache and IIS server throughput is linear up to 400, 200 and 300 dynamic page requests/sec respectively. In OS-based servers, there is more overhead due to scheduling a large number of threads, and maximum CPU utilization is reached even for moderately large workloads.

If f and g are the file sizes, and r and s are the request rates for dynamic and static page requests respectively, since there are 3 ACKs per request sent by the server (ACKs for SYN and GET, and the final ACK), the expected server throughput is given by

$$\left(f + h\left(\left\lceil f / MSS \right\rceil + 3\right)\right) r + \left(g + h\left(\left\lceil g / MSS \right\rceil + 3\right)\right) s,$$

where h is the header overhead due to Ethernet, IP and TCP; and MSS is the TCP maximum segment size. Table 1 shows the expected throughput for various dynamic and static page request rates assuming a 211-byte dataset (plus a 151-byte HTTP header) and a 3593-byte static page file. The expected values are close to the measured bare PC server throughput in Fig. 14 with 1000 static page requests/sec and for dynamic requests rates up to 400 request/sec (the expected throughput is also close to the measured throughput for the IIS and Apache servers up to 300 dynamic page requests/sec). The measured throughput for dynamic page requests only (Fig. 13) is much lower than the expected throughput due to DB server latency.

TABLE 1
EXPECTED THROUGHPUT: DYNAMIC AND STATIC REQUESTS

Dynamic	Static	Throughput
400	1000	33.4
300	1000	33.0
200	1000	32.5
100	1000	32.0
600	0	2.9
800	0	3.8
1000	0	4.8
0	6000	189.2

F. *Database Latency*

The respective times for the handshake, connection, tabular response, and disconnect phases of a bare PC DB connection (Fig. 2) are shown in Fig. 15 for various dataset sizes when making 100 dynamic page requests/sec. Note that since the request rate is small, the handshake, connection, and disconnection times are fairly stable but the tabular response time increases significantly for larger dataset sizes. Fig. 16 shows the time for handshake, connection, tabular response, and disconnection phases for a fixed dataset size of 211 bytes and varying request rates. When the rate increases, all phases except for the handshake incur moderately increased times due to the increased load. The handshake time is the dominant contributor to response time and it shows a larger increase when the request rate increases from 300 request/sec to 600 requests/sec. This explains the difference in the measured throughput in Fig. 13 and expected throughput in Table 1.

V. CONCLUSION

The bare PC Web server exploits an OS-less architecture to minimize the overhead when processing dynamic and static requests. In particular, the novel server design includes integration of lean network protocols, efficient task scheduling, interrupt reduction, minimal data copying, and direct hardware/driver interfaces. Our experimental results indicate that the bare PC Web server has significantly better performance and lower CPU utilization than the Apache and IIS servers for workloads consisting of requests for dynamic and static content.

REFERENCES

[1] T. Brecht, D. Pariag, and L. Gammo, “Accept()able strategies for improving Web server performance,” Proc. Usenix 2004 Annual Technical Conference, General Track, pp. 227–240, 2004.

[2] B. Ford, M. Hibler, J. Lepreau, J. R. McGrath, and P. Tullman, “Interface and execution models in the Fluke kernel,” Proc. Third Symposium on Operating Systems Design and Implementation, pp. 101-115, 1999.

[3] G. R. Ganger, D. R. Engler, M. F. Kaashoek, H. M. Briceno, R. Hunt, and T. Pinckney, “Fast and flexible application-level networking on exokernel system,” ACM Transactions on Computer Systems, vol. 20, no. 1, pp. 49-83, Feb. 2002.

[4] L. He, R. K. Karne, A. L. Wijesinha, S. Girumala, and G. H. Khaksari, “Design issues in a bare PC Web server,” Proc. SNPD 2006, pp. 165-170, 2006.

[5] L. He, R. K. Karne, and A. L. Wijesinha, “The design and performance of a bare PC Web server,” The International Journal of Computers and Their Applications, vol. 15, no. 2, pp. 100-112, June 2008.

[6] http_load, http://www.acme.com/software/http_load

[7] SourceForge.net:httperf, http://sourceforge.net/projects/httperf

[8] The OSKit Project, http://www.cs.utah.edu/flux/oskit

[9] TinyOS Community Forum||An open-source OS for the networked sensor regime, http://www.tinyos.net

[10] R. K. Karne, K. Venkatasamy, T. Ahmed, and N. Rosa, “DOSC: Dispersed Operating System Computing,” Proc. OOPSLA 2005, Onward Track, San Diego, CA, Oct. 2005.

[11] R. K. Karne, K. Venkatasamy, T. Ahmed, ”How to run C++ applications on a bare PC,” Proc. SNPD 2005, 6th ACIS International Conference, Towson, MD, pp. 50-55, May 2005.

[12] R. K. Karne, R. Gattu, R. Dandu, X. Zhang, and J. Vodela, “Application-oriented object architecture: a revolutionary approach,” Poster Paper, 6th International Conference on High Performance Computing, Bangalore, India, Dec. 2002.

[13] H. Kim, V. Pai, and S. Rixner, “Increasing web server throughput with network interface data caching,” Proc. 10th International Conference on Architectural support for programming languages and operating systems, San Jose, California, pp. 239-250, Oct. 2002.

[14] E. Nahum, T. Barzilai, and D. D. Kandlur, “Performance issues in WWW servers,” IEEE/ACM Transactions on Networking, vol. 10, no.1, pp. 2-11, Feb. 2002.

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

[16] D. Pariag, T. Brecht, A. Harji, P. Buhr, and A. Shukla, “Comparing the performance of Web server architectures,” Proc. EuroSys ’07, pp. 231-243, Lisbon, Portugal, Mar. 2007.

[17] L. Titchkosky, M. Arlitt, and C. Williamson, “A. performance comparison of dynamic Web technologies,” ACM Sigmetrics Performance Evaluation Review, vol. 31, no. 3, pp. 2-11, 2003.

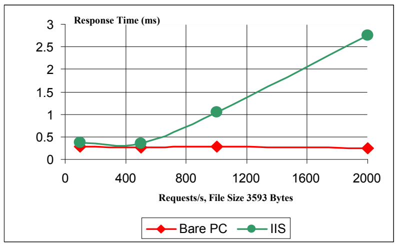

Fig. 3. Response Time: Static Requests

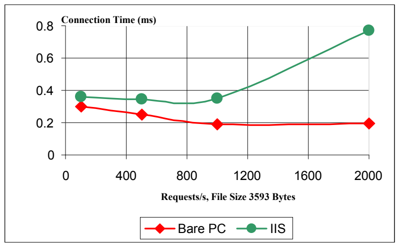

Fig. 4. Connection Time: Static Requests

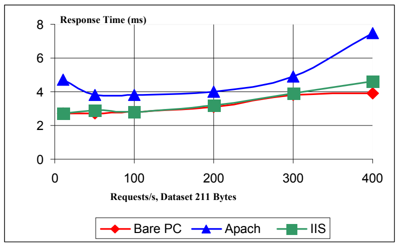

Fig. 5. Response Time: Dynamic Requests

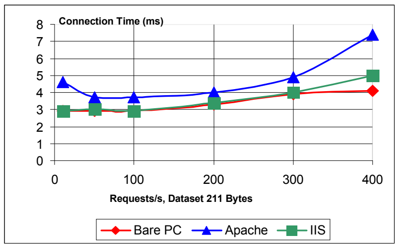

Fig. 6. Connection Time: Dynamic Requests

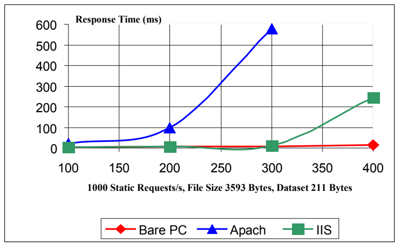

Fig. 7. Response Time: Dynamic and Static Requests

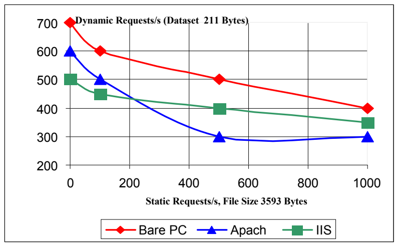

Fig. 8. Dynamic vs. Static Request Rates

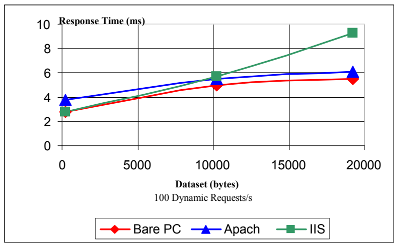

Fig. 9. Response Time vs Dataset Size

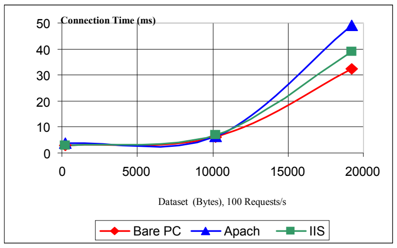

Fig. 10. Connection Time vs Dataset Size

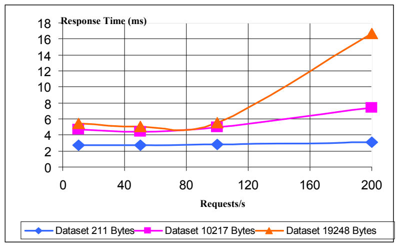

Fig. 11. Bare PC Response Time

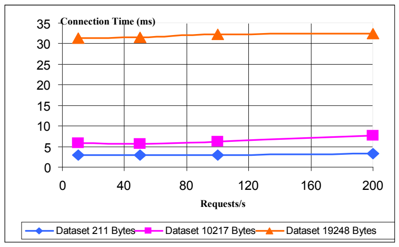

Fig 12. Bare PC Connection Time

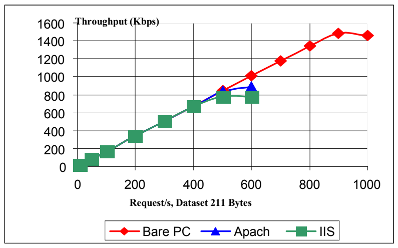

Fig. 13. Throughput: Dynamic Requests

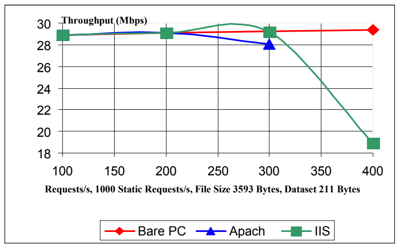

Fig. 14. Throughput: Dynamic and Static Requests

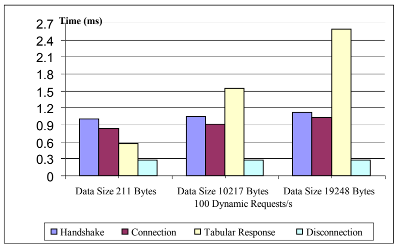

Fig. 15. Database Latency vs. Dataset Size

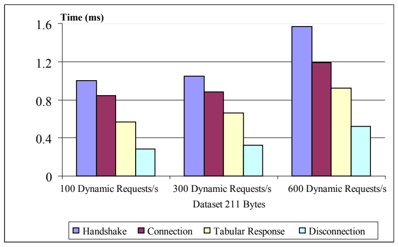

Fig. 16. Database Latency vs. Dynamic Request Rate