Web File Transmission by Object Packaging – Performance Comparison with HTTP 1.0 and HTTP 1.1 Persistent Connection

Hiroshi Fujinoki, Murugesan Sanjay, and Chintan Shah

Department of Computer Science
Southern Illinois University at Edwardsville
Edwardsville, Illinois 62026-1656, USA
E-mail: {hfujino, msanjay, cshah}@siue.edu

Abstract

Recently, the largest concern for corporate owners of web servers is how to minimize the response time. In order to minimize the response time, a new efficient file transmission technique for World-Wide-Web, called web file transmission by Object Packaging, is proposed. The prototype design and preliminary performance evaluation of Object Packaging have been presented in the 27th Conference on Local Computer Networks. In this paper, the performance of Object Packaging is compared to HTTP 1.0 and HTTP 1.1 Persistent Connection in terms of the server and client response time and CPU workload by experiment method. It is found that Object Packaging reduces the response time and CPU workload at the server by up to 92.0 and 91.1% from HTTP 1.0 and 64.5 and 82.4% from HTTP 1.1 Persistent Connection, respectively. From the results of the experiments, it is found that Object Packaging will be an efficient technique to minimize the response time in transferring web files. The results of the experiments also suggest the significance of disk I/O overhead for web servers in high speed networks.

1. Introduction

The recent advances in signal transmission medium have achieved high data transmission rate and low error rate. Those advances, however, shift the performance bottleneck from "wires" to "boxes". Here, "boxes" mean network equipment contained in a case, such as routers, switches and end hosts. In this regard, the largest concern for corporate web server owners becomes the server response time. For example, a survey for web users' activity found that average web users would leave a commercial web site after 8 seconds in delay [1]. Due to the demand on short response time, improving the web server response time becomes one of the most urgent problems for corporate web server owners to avoid loosing potential customers.

Despite the importance, reducing the response time is not an easy task. For example, increasing the link bandwidth of the transmission medium may not be an ultimate solution. This is because if the throughput of a server does not catch up with the link bandwidth, increasing link bandwidth will be of no help to reduce the response time. In the worst case, wider link bandwidth may increase customer arrival rate, which will even aggravate the server response time.

Response time will not be significantly improved just by improving the performance of a particular hardware component in an end host. For example, upgrading the central processing unit at a web server may not solve the problem, either [2]. This is because delay in the response time at a server comes from various types of overhead, such as protocol overhead, operating system overhead for context switching, disk access overhead and overhead due to multiple memory copies between layers in a network protocol stack [3].

In our previous work, disk I/O was focused on as one of the performance bottlenecks in a web server. To cope with disk I/O overhead at a web server, an efficient web file transmission technique, Object Packaging, was proposed [4]. It was found that Object Packaging reduced 34.7, 7.1 and 41.0% of transmission time, the number of bytes transmitted and the number of packets from HTTP 1.0, respectively. In this paper, the server and client overhead of Object Packaging are compared to that of HTTP 1.0 and HTTP 1.1 Persistent Connection. As measurements of overhead, response time and accumulated CPU utilization (defined later) were measured and compared to evaluate the performance of HTTP 1.0, HTTP 1.1 Persistent Connection and Object Packaging.

The rest of this paper is organized as follows. In Section 2, existing methods to reduce response time for web file transmission are reviewed. In Section 3, the web file

transmission by Object Packaging is described. In the section, design motivations of the web file transmission by Object Packaging are described to identify its advantages over HTTP 1.0 and HTTP 1.1 Persistent Connection. In Section 4, performance evaluation using experiment method is presented. In the section, experiment modeling and experiment design are described. Control parameters used in the experiments are defined and described. At the end of Section 4, the outcomes of the experiments are presented. Section 5 concludes this work, followed by a list of references.

2. Existing Methods

Due to the importance of reducing the response time at a web server, various methods have been proposed. The existing methods to reduce response time are classified in the two groups: one based on hardware and the other on software. The hardware based methods include server-side/client-side caching, server clustering, server mirroring and multi-homed servers [5]. The idea behind the hardware-based methods is to reduce the server response time by installing high performance hardware components or by installing extra hardware components. The disadvantage in common for all these hardware based methods is that they require investment.

Regarding the cost for hardware based solutions, it is suggested that the majority of the web sites are operated by small corporate and personal webmasters (40% of web masters are managing only one server and more than 70% of web masters manage four servers or less) [6]. Due to the required investment, the hardware based methods may not be an easy option for most of such personal and small corporate web site owners, which may limit the usability of the hardware based methods to reduce server response time.

As one of the software based solutions to reduce the server response time, HTTP 1.1 Persistent Connection [7, 8, 9] was proposed. HTTP 1.1 Persistent Connection was proposed to reduce overhead in transport layer protocol in HTTP 1.0, which is usually TCP. The largest problem in HTTP 1.0 was in its inefficiency in handling multiple small files. The problem stems from the fact that a new connection is required for each file to be requested by a client. The major overhead in HTTP 1.0 includes the overhead to establish and delete connections and those for managing existing connections at a web server. The three-way handshaking for establishing a connection is especially a serious overhead as the transmission rate increases. Since the three-way handshake entails delay for one round trip, the overhead will be relatively high for small files. Delay from TCP slow-start causes further delay. The significance of the problem is that the overhead is required for each requested file [10].

In addition, the overhead for TCP three-way handshaking will have more serious impact to a server as the transmission rate increases. If the number of clients that arrive in a certain time is $\boldsymbol{q}$ and if each client requests $\boldsymbol{r}$ files in average during the time, the workload to manage TCP connections at a web server is in the order of $\boldsymbol{q} \times \boldsymbol{r}$. Recent improvement in transmission medium, such as fiber optical cables, certainly increases the number of requests that can be delivered to a web server in a short time. Therefore, the overhead for TCP three-way handshaking will be a scalability bottleneck in terms of the server response time.

HTTP 1.1 was proposed to provide a solution to the scalability problem in HTTP 1.0 mentioned above. Persistent connection used in HTTP 1.1 was introduced to reduce overhead for TCP connection establishment and management. Instead of establishing a new TCP connection for each file requested, the TCP connection established for the first requested file will be reused for all the other files in the same web page. Since the same TCP connection will be used for multiple files, the overhead for TCP three-way handshake is required only for the first file. This reduces the server overhead from $\boldsymbol{r} \times \boldsymbol{q}$ to $\boldsymbol{q}$. This implies that the server overhead is now $O(n)$, where n is the number of clients a server needs to service in a certain time.

In addition to the reduction in the overhead for connection establishment, HTTP 1.1 Persistent Connection is expected to reduce the server overhead. We defined the server overhead to be any overhead required at the server after a client request arrives until the last packet in the requested file is transmitted at the server. If a TCP connection is shared by multiple file transmissions, the CPU workload for managing TCP connections will be reduced, too. This is because a large number of processes will be created and deleted in a short time if a TCP connection would have been needed for every file requested by clients. Slashing due to the excess context switching for multiple concurrent processes may further increase the server response time. HTTP 1.1 Persistent Connection will be effective in solving the above problems.

Although HTTP 1.1 Persistent Connection will be effective in reducing overhead for connection establishment and management, and thus in reducing the response time for web file requests, there are some issues HTTP 1.1 Persistent Connection will not improve. We recognized the following two major issues. (1) In HTTP 1.1, requested files will be still individually accessed in the server's local drive although they share a TCP connection. When multiple small files are requested, the impact of the individual disk accesses will be significant to the server

response time. (2) Payload capacity of the last packet for each file transmission may not be fully utilized.

The two problems mentioned in the previous paragraph (individual disk access and underutilized packet capacity) will cause serious inefficiency problem in web file transmissions because of the following three reasons. (1) The average file size of requested files in web is small (the average file size requested in web is 10KB [11]). (2) Multiple files are requested. (3) Multiple small files will be requested in a burst within a short time. If multiple small files are requested as a burst in a short time, the relatively high overhead in the operating system at a server will make reducing the server response time difficult. For further improvement in the server response time achieved by HTTP 1.1 Persistent Connection, tasks in the operating system should be optimized for the unique properties in web file transmissions, that is multiple requests for small files in a short time. We assume that the uplink bandwidth will not be the performance bottleneck for small corporate web servers in the future.

3. Effective Web File Transmission by Object Packaging

3.1 Design Goals of Object Packaging

The design goal of Object Packaging is to minimize the response time in web browsing without requiring any hardware modification in the network infrastructure, such as routers, links and end system hardware. The major premise in Object Packaging is to optimize disk I/O accesses by minimizing FAT look-up overhead and to reduce network protocol overhead by reducing the number of packets. The web file transmission by Object Packaging improves the server response time by transmitting multiple small files as if they were single large file. Multiple files to be packed in an object package are those that construct a web page, such as HTML text files and images for link banners and advertisements.

There are two major differences between Object Packaging and transmitting multiple files using existing archive files such as UNIX ***tar***. The first difference is that individual files in an object package can be used at a client without waiting for the entire object package to be received by a client, which is not the case for the existing archive files. If a receiver has to wait for the entire archive file to be received by a client, the response time will be long especially for the first file in the archive file, which will make response time even worse. The second difference is that multiple files in an object package will be packed with an efficient structure to minimize unpacking overhead at a client. As described in detail later, the design goal of object packaging is to unpack multiple files in an object package in the order of O(***n***) where ***n*** is the number of files packed in an object package. In order to avoid extra overhead that sacrifices the improvement achieved at the server side, object packages will be constructed and transmitted without compression. This decision was made based on the following two reasons: (1) it is based on our assumption that the link bandwidth will not be the performance bottleneck in the future. (2) Each individual file should be available at a client without waiting for the entire object package to be received.

3.2 Definition of the format of the object package

In the web file transmissions by Object Packaging, multiple files requested by a client are transmitted in an object package. The object package is defined as follows. Each object package is a collection of web files in a web page packed in an uncompressed format. Multiple files, such as HTML text and image files, will be packed in an object package file in advance. Those files in a web page are sequentially packed into an object package file without compression to minimize decompression overhead at the receiver. At a client, a web browser should unpack the files from an object package file. Object packages should be recognized by a specific file extension at a client. The format of the object package is shown in Figure 1.

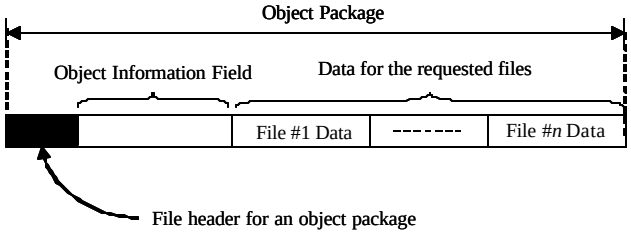

Figure 1 – Major components of the object package

The first element in an object package is the object information field. The object information field contains the information of the packed files that should be restored at a client. The data field contains only the contents of the requested files. A receiver reconstructs the requested files using the information in the object information field and the contents (from data field) of the requested files. The object information field consists of four subfields. See Figure 2 for this.

The first subfield is the number of files packed in an object package. The second is the filename subfield, which is a list of the file names for the files contained in this object package. Each element in the filename subfield, which is a file name, is variable in the length. The null terminator indicates the end of each file name. The third subfield contains the file sizes of the packed files. File size is fixed in the length. The last subfield is the attribute

subfield. The attribute subfield contains a list of file attributes for each file such as binary, text, executable or read-only. Each file attribute is fixed in the length. At a client, the first file (File #1) will be reconstructed based on the first element in the file name, the file size and the file attribute subfields.

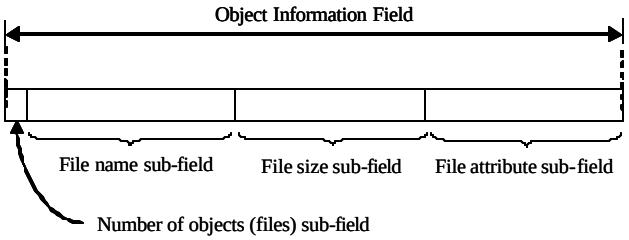

Figure 2 – Elements in object information field

The procedures at a receiver are as follows. The integrity of the object package will be examined based on the object package header when an object package is received. Then the next subfield, the number of objects subfield, will be received. As soon as the number of objects is retrieved, a data structure to manage the packed files will be created. This data structure is called the object package management table. The table has three columns and they are for the file name, the file size and the file attribute subfields. Each row corresponds to a requested file packed in an object package (shown in Table 1). Then, as file name subfield is received, the column for the file names will be filled up. The file size and file attribute subfields will be filled up in the same way.

Table 1 – Structure of object package management table

File Name	File Size (bytes)	File Attribute
File 1	*8241*	*A*
File 2	*10276*	*B*
• • •	• • •	• • •
File n	*4186*	*A*

3.3 Procedures of object package decomposition

After the object package management table is filled, packed files will be retrieved on the fly based on the information in the object package management table. Note that the contents of the packed files will be received at a client only after the package management table is completed. To reconstruct the files packed in an object package on the fly, two pointers will be used. The first pointer is called File Entry Pointer (FEP), which points to the file currently under the receiving process. The other pointer is called Local File Pointer (LFP), which is a counter of the received bytes in a packed file (in another word, it points to the position of the next byte to be received for a packed file).

After the object package management table is filled, FEP is set to the first row in the table and LFP is set to be 0 (i.e., the first byte in a packed file). Then, the file name and the file attributes will be retrieved from the row pointed by FEP and a file will be opened for creation using the retrieved file name with the given file attributes. For each byte received, LFP is incremented by one. When LFP reaches the file size entry in the row currently pointed by FEP, the file will be closed. Then FEP will be shifted to the next row down in the table and LFP will be reset to 0. The same procedures will be repeated until the last file in an object package is processed. The procedures are shown in Figure 3.

1. Receive the header for an object package and check the file integrity
2. Receive the number of objects subfield and set the number to a variable ***n***
3. Create the object package management table with ***n*** rows
4. Receive ***n*** file names
5. Receive ***n*** file sizes
6. Receive ***n*** file attributes
7. Set FEP to the first row in the object package management table
8. Set LFP to be 0
9. Retrieve the file name and attribute from the entry pointed by the object entry pointer
10. Create a file using the information retrieved at step 9 above
11. Read as many bytes as specified by the file size subfield
12. If all contents in a file are received, close the file and shift FEP to the next entry in the table
13. Repeat step 8 through 12 for ***n*** times

Figure 3 – Unpacking procedures of object package

Since the requested files are sequentially packed in an object package without compression, the requested files in an object package can be unpacked on the fly. Requested files are encapsulated within an object package so that they can be handled just as data, which should be transparent from the underlying transmission protocol, such as TCP.

If there are some files common in multiple web pages in a web site, those common files can be included only in the object package for the top page (or for the first page where those files are used) in a web site. Then, such common files can be shared in the client side cache. If a user accesses to another web page without going through its top (or its first) page, then the common files that do not exist in

the object package should be individually requested to a server.

A plug-in program in a web browser running as a user-mode process at a receiver can unpack object packages, which means that object packages are transparent from an operating system and a transmission protocol. The complexity of the package decomposition at a receiver is in the order of O(n), where n is the number of files packed in an object package.

4. Experiments

In this section, performance of HTTP 1.0, HTTP 1.1 Persistent Connection and Object Packaging are compared in terms of CPU workload and the response time at both the server and the client using experiment method. The testbed as shown in Figure4 was setup for the experiments.

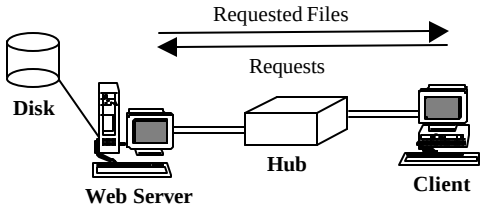

Figure 4 – The testbed for performance evaluation

4.1 Experiment Modeling

The server and client machines were desktop PCs, each of which had Intel Pentium III 1.0G Hz and 512M bytes of memory. As an operating system, Microsoft Windows XP Professional was used at both the server and the client. For the network interface, Intel Pro/100 VE Network Connection was used at both the server and the client. NETGEAR Fast Ethernet Switch (FS-105) was used for the hub that connected the server and client. Transmission rate was 100 Mbps for the links used in the testbed. The testbed network was isolated from other networks for accuracy in the results.

Using the testbed, CPU utilization and the server and the client response time (defined later in this section) were measured. CPU utilization was measured at the fixed intervals of 100ms at the server and the client. The end of file transmissions was detected when zero CPU utilization was measured for more than five consecutive readings.

Server CPU utilization and response time were measured during the time between when the first request arrived at the server and when the last packet of the last file was transmitted (i.e., when the last "send" function was completed for the last file). For example, when 100 files were requested by the client, the server started measuring CPU utilization and the response time when the request for the first file arrived at the server. The server stopped measuring CPU utilization and the response time when the last "send" function for the 100th file was completed.

Client CPU utilization was measured from the time when the request for the first file was sent until the last file was received at the client. The client response time is defined to be the time between when the first call to "read" function was completed for the first file requested and when the last call to "read" function for the last file was completed. For example, after the client sent a request for the first file to the server, the client blocked at the first call to the "read" function. When the first "read" function was completed for the first file requested, the client started measuring the response time. The client stopped measuring the response time when the last call to "read" function for the last file, which was 100th file, was completed.

The client and the server programs for HTTP 1.0, HTTP 1.1 Persistent Connection and Object Packaging were written in C++ and compiled by Microsoft Visual Studio 6.0. The same server program was used for both HTTP 1.0 and Object Packaging. Client programs are unique for each of HTTP 1.0, HTTP 1.1 Persistent Connection and Object Packaging. Thus, total of five programs were developed for this project: three client programs (one for each) and two server programs (one for HTTP 1.0 and Object Packaging and the other for HTTP 1.1 Persistent Connection). No code optimization was applied when the source codes were compiled. All the source codes are available at the author's home page [12].

To model the files to be requested by the client, discrete Pareto distribution was used. Crovella suggested that file size distribution for web can be heavy-tailed with high variance [13]. To simulate the heavy-tail in file size distribution of web traffic, Busari suggested Pareto distribution [14]. Since Pareto distribution has infinite tail, Bounded Pareto distribution was used in our simulation experiments to control the mean of the generated files. The Bounded Pareto random number generator requires three parameters, k, p and α. Parameter k and p specify the smallest and the largest possible file sizes, respectively. Parameter α defines the shape of the Bounded Pareto distribution. Arlitt suggested $\alpha = 0.63$ to represent distribution of the files transferred in the www [11]. The program we used to generate Bounded Pareto random numbers is available at Christensen's tool page [15].

4.2 Experiment Design

Performance of HTTP 1.0, HTTP 1.1 Persistent Connection and Object Packaging is compared in the CPU workload and the response time to complete multiple requests. CPU utilization and the response time were

measured at both the server and the client. To test the performance of the three methods under various situations, multiple files were requested by the client for different file sizes. Table 2 shows all the files used in our experiments. Using the groups of the test files, the following twelve experiments were defined:

Table 2 – Groups of the test files used for the experiments

Test File Groups	Number of sets	Number of files in each set	Type of files	Average file size
#1	10	100	Text Files	4KB
#2	10	100	Text Files	16KB
#3	10	100	Text Files	32KB
#4	10	1	OBJP	4KB
#5	10	1	OBJP	16KB
#6	10	1	OBJP	32KB

Experiment 1 – Server CPU workload for HTTP 1.0: CPU utilization was measured at the server while groups of 100 files were being requested and transferred using HTTP 1.0. The groups of files used for this experiment were Group #1, #2 and #3 in Table 2. To measure CPU utilization for a long time interval, five sets of the test files were requested at once (i.e., a total of 500 files were requested consecutively in this experiment). The same test was repeated one more time using the other five sets of the test files.

Experiment 2 – Server CPU workload for HTTP 1.1 Persistent Connection: The same experiment as performed for Experiment #1 was performed for HTTP 1.1 Persistent Connection.

Experiment 3 – Server CPU workload for Object Packaging: The same experiment as performed for Experiment #1 was performed for Object Packaging. The groups of files used for this experiment were Group #4, #5 and #6 in Table 2. The files used in Experiment #1 were used to create the object packages (OBJPs). Similar to Experiment #1, the five sets of object packages were requested consecutively.

Experiment 4 – Server response time using HTTP 1.0: This experiment measured the response time to complete requests for 100 files at the server. The groups of files used for this experiment were Group #1, #2 and #3 in Table 2. The same experiment was repeated for ten times using the ten sets of 100 files in each test file group.

Experiment 5 – Server response time using HTTP 1.1 Persistent Connection: Experiment #4 was repeated for HTTP 1.1 Persistent Connection, instead of HTTP 1.0.

Experiment 6 – Server response time using Object Packaging: The same experiment as performed for Experiment #4 was performed for Object Packaging. The groups of files used for this experiment were Group #4, #5 and #6 in Table 2.

Experiment 7 through 9 – Client CPU workload for HTTP 1.0, HTTP 1.1 Persistent Connection and Object Packaging: These experiments were the same as Experiment #1 through #3 except that CPU utilization was measured at the client.

Experiment #10 through #12 – Client response time for HTTP 1.0, HTTP 1.1 Persistent Connection and Object Packaging: These experiments are the same as Experiment 4 through 6 except that the response time was measured at the client.

4.3 Observations

Figure 5 shows the CPU utilization measured at the server for transmitting a group of 100 files of 4KB using HTTP 1.0 and the Object Packaging. Measurement count means the number of measurements for CPU utilization made at the 100ms intervals after the server receives the first request from the client. As can be seen from Figure 5, the CPU utilization for Object Packaging hit 0% at and after the 4th measurement (which is 400ms after the request for the first file was received by the server), while the CPU utilization of 0% was hit at and after the 28th measurement for HTTP 1.0.

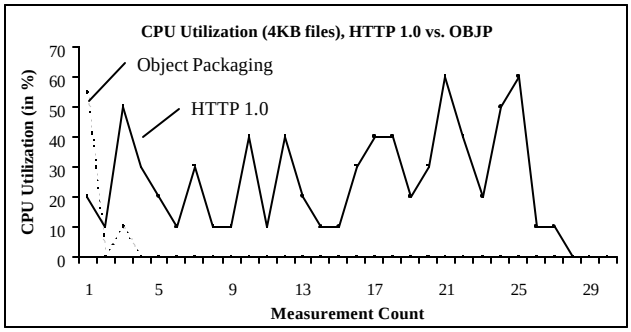

Figure 5 – Server CPU Utilization for HTTP 1.0 and Object Packaging for 100 files of 4KB

Figure 6 shows the server CPU utilization measured for HTTP 1.1 Persistent Connection. From Figure 6, it can be seen that the CPU utilization for HTTP 1.1 Persistent Connection hit 0% at and after the 9th measurement. It was observed that the time duration the CPU was used for Object Packaging was 14.3% (= 4/28) (from Figure 5) and 44.4% (= 4/9) (from Figure 6) of that of HTTP 1.0 and HTTP 1.1 Persistent Connection, respectively. Figure 7 and 8 show the results for 32KB files. Figure 7 shows the results for HTTP 1.0 and Object Packaging. Figure 8 shows the results for HTTP 1.1 Persistent Connection (the plots for Object Packaging are shown for comparison).

Figure 7 and 8 show that the CPU utilization for Object

Packaging hits 0% at and after 14th measurement, while they were the 46th and the 26th for HTTP 1.0 and HTTP 1.1 Persistent Connection, respectively. Table 3 shows the accumulated CPU utilization for HTTP 1.0, HTTP 1.1 Persistent Connection and Object Packaging. Accumulated CPU utilization is a summation of CPU utilization readings, and it is an indicator of total CPU workload required to transfer the files. It was found that Object Packaging resulted in only 8.9% (= 65/730) in the accumulated CPU utilization compared to that of HTTP 1.0 for the 4KB files, while it was 50.7% (= 370/730) for HTTP 1.1 Persistent Connection. For the 16KB files, Object Packaging resulted in 36.4% (= 280/770), while it was 92.2% (= 710/770) for HTTP 1.1 Persistent Connection. For the 32KB files, Object Packaging resulted in 35.7% (= 418/1170) while HTTP 1.1 Persistent Connection resulted in 68.4% (= 800/1170). For comparison between HTTP 1.1 Persistent Connection and Object Packaging, Object Packaging reduced 82.4, 60.6 and 47.8% of CPU workload from HTTP 1.1 Persistent Connection for 4KB, 16KB and 32KB files, respectively at the server (see Figure 9).

Table 3 – Accumulated CPU utilization at the server to complete requests for groups of 100 files

Transmission Methods	4K	16K	32K
HTTP 1.0	730	770	1170
HTTP 1.1 Persistent Connection	370	710	800
Object Packaging	65	280	418

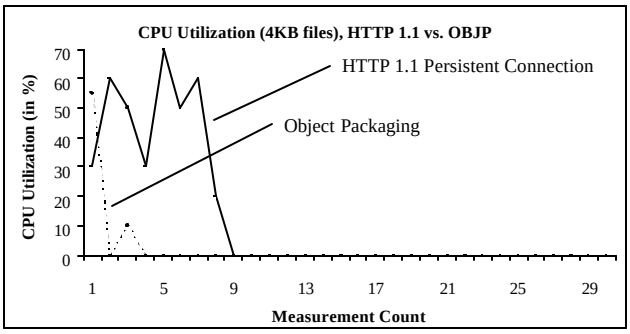

Figure 6 – Server CPU Utilization for HTTP 1.1 Persistent Connection and Object Packaging for transmitting 100 files of 4KB

Results of Experiment 4 through 6 are shown in Table 4. It was observed that Object Packaging and HTTP 1.1 Persistent Connection reduced 92.0 and 77.4% of the server response time from HTTP 1.0 when a group of 100 files of 4 KB were used (see Figure 10). The reduction was 75.0 and 62.1% for Object Packaging and HTTP 1.1 Persistent Connection, respectively for 16KB files and they were 55.9 and 47.7%, respectively for 32KB files. Object Packaging reduced 64.5, 34.0 and 15.5% of the server response time from HTTP 1.1 Persistent Connection for the 4KB, 16KB and 32KB files, respectively.

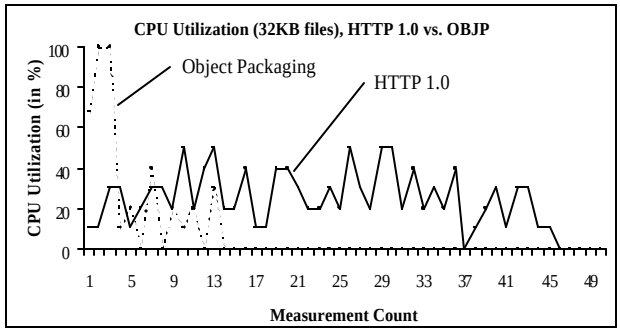

Figure 7 – Server CPU utilization for HTTP 1.0 and Object Packaging for 100 files of 32KB

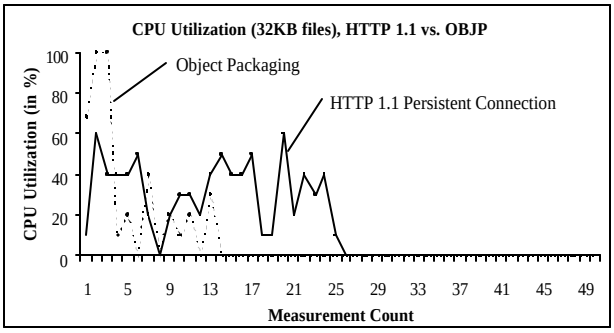

Figure 8 – Server CPU utilization for HTTP 1.1 Persistent Connection and Object Packaging for transmitting 100 files of 32KB

Table 4 – The average server response time to complete requests for groups of 100 files (in milliseconds)

Transmission Methods	4K	16K	32K
HTTP 1.0	550.3	703.5	890.2
HTTP 1.1 Persistent Conn.	124.5	266.8	465.2
Object Packaging	44.2	176.0	392.9

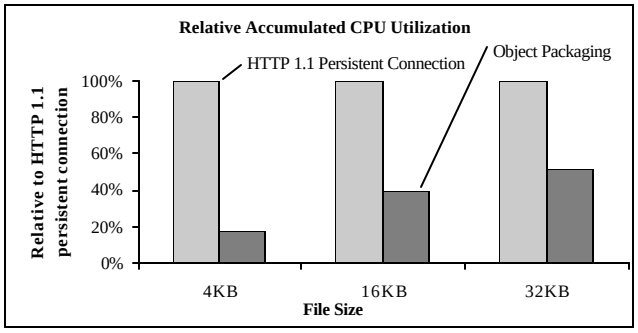

Figure 9 – Comparison between HTTP 1.1 Persistent Connection and Object Packaging for the server accumulated CPU utilization

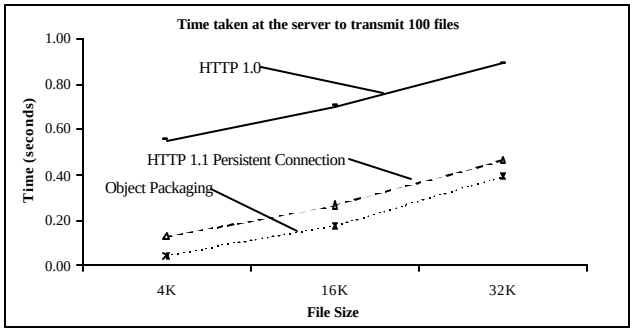

Figure 10 – Server response time for transmitting groups of 100 files for 4KB, 16KB and 32KB

As can be seen in common in all the experiments, Object Packaging was efficient when the file size was small. When the file size was small (a group of 100 4KB files), the relative reduction in CPU utilization and the server response time was large (for both, more than 90% is reduced from HTTP 1.0). Regarding this point, if small files are requested and transmitted in a burst, Object Packaging will be more efficient than HTTP 1.0 and HTTP 1.1 Persistent Connection in reducing the response time and CPU workload at a web server (Figure 10).

Table 5 – Client accumulated CPU utilization to receive groups of 100 files (average for 10 sets)

Transmission Methods	4K	16K	32K
HTTP 1.0	2080	2410	3190
HTTP 1.1 Persistent Conn.	140	340	580
Object Packaging	60	200	204

Figure 11 and 12 show the plots of CPU utilization at the client for HTTP 1.0, HTTP 1.1 Persistent Connection and Object Packaging for the 4KB files. Table 5 shows the client accumulated CPU utilization for Experiment 7 through 9.

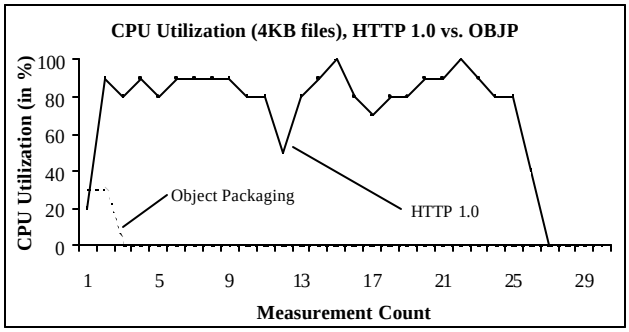

Figure 11 – Client CPU utilization for HTTP 1.0 and Object Packaging for transmitting 100 files of 4KB

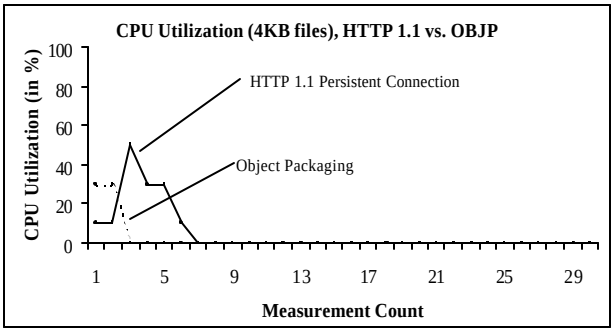

Figure 12 – Client CPU utilization for HTTP 1.1 Persistent Connection and Object Packaging for transmitting 100 files of 4KB

Figure 13 shows the comparison in the client accumulated CPU utilization between HTTP 1.1 Persistent Connection and Object Packaging. Table 6 shows the results for Experiment 10 through 12. Table 6 shows the client response time to complete requests for 100 files. It took 499.1, 688.5 and 875.3ms in the average for 10 sets for HTTP 1.0 to complete requests for groups of 4KB, 16KB and 32KB files, respectively. Similarly, it took 88.0, 225.4 and 437.5ms for HTTP 1.1 Persistent Connection. For Object Packaging, they were 45.2, 113.3 and 245.7ms. Object Packaging reduced 48.6% ((88.0-45.2)/88.0) of HTTP 1.1 Persistent Connection for 4KB files (Figure 14).

Table 6 – The average client response time to receive groups of 100 files (in milliseconds)

Transmission Methods	4K	16K	32K
HTTP 1.0	499.1	668.5	875.3
HTTP 1.1 Persistent Conn.	88.0	225.4	437.5
Object Packaging	45.2	113.3	245.1

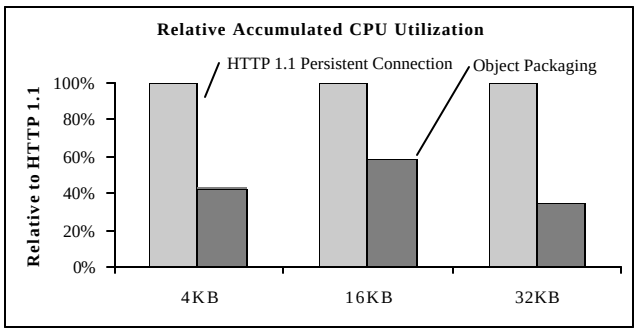

Figure 13 – Comparison between HTTP 1.1 Persistent Connection and Object Packaging in the accumulated CPU utilization at the client

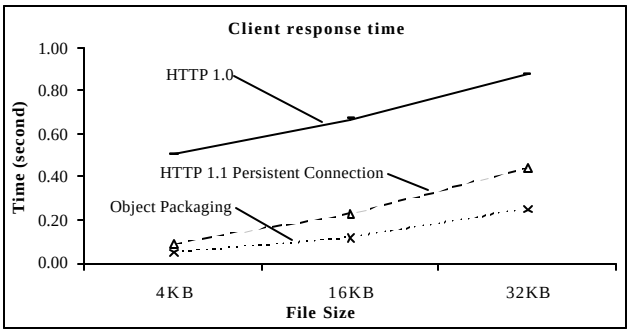

Figure 14 – Client response time for transmitting groups of 100 files for 4KB, 16KB and 32KB

Table 7 shows the performance comparison of HTTP 1.0, HTTP 1.1 Persistent Connection and Object Packaging in the relative (relative to HTTP 1.0 in percentage) response time and the relative accumulated CPU utilization at the server and the client. HTTP 1.1 Persistent Connection resulted in 51, 92 and 68% of HTTP 1.0 in the accumulated CPU utilization at the server for 4KB, 16KB and 32KB files, respectively. The server response time for HTTP 1.1 Persistent Connection resulted in 23, 38 and 52% of HTTP 1.0 for 4KB, 16KB and 32KB files, respectively (this order is assumed for the rest). Object Packaging resulted in 9, 36 and 36% of HTTP 1.0 in the accumulated CPU utilization for the three different file sizes. Object Packaging resulted in 8, 25 and 44% of HTTP 1.0 for server response time. For the accumulated CPU utilization at the client, HTTP 1.1 Persistent Connection resulted in 7, 14 and 18% of HTTP 1.0, while the client response times were 18, 34 and 50% of HTTP 1.0. Object Packaging resulted in 3, 8 and 7% of HTTP 1.0 in the accumu lated CPU utilization at the client. The client response times by Object Packaging were 9, 17 and 28% of HTTP 1.0.

Table 7 – Comparison for HTTP 1.1 Persistent Connection and Object Packaging in relative to HTTP 1.0

	4KB	**16KB**	**32KB**
Server CPU Utilization			
HTTP 1.0	100%	100%	100%
HTTP 1.1 Persistent Conn.	51	92	68
Object Packaging	9	36	36
Server Response Time			
HTTP 1.1 Persistent Conn.	23%	38%	52%
Object Packaging	8	25	44
Client CPU Utilization			
HTTP 1.1 Persistent Conn.	7%	14%	18%
Object Packaging	3	8	7
Client Response Time			
HTTP 1.1 Persistent Conn.	18%	34%	50%
Object Packaging	9	17	28

Table 8 shows the statistics collected for CPU utilization consumed for background processes. The background CPU utilization was measured after one minute of non-operation for five minutes at a fixed interval of 100ms at the server. It was found that the highest and the average background CPU utilization during the idle time (during the five minutes) were 20% and 0.48%, respectively. The average utilization of 0.48% is equivalent to observing the accumulated CPU utilization of 480 during 100 seconds. The variance in the measured background utilization was 7.1. Intervals of any two positive background CPU readings were also measured. The minimum interval was 0 (two positive background CPU readings were consecutively observed), while the maximum interval was 178 readings (i.e., 17.8 seconds between two positive background CPU utilization readings). The variance for intervals was 838.5. Similar background CPU utilization was observed at the client.

Table 8 – Noise level analysis in server CPU utilization

	Utilization (%)	**Interval (ms)**
Minimum	0	0
Maximum	20	17800
Average	0.48	2750

5. Conclusions and Future Work

Object Packaging reduced up to 64.5% of the server response time from HTTP 1.1 Persistent Connection. It was also demonstrated that Object Packaging transmitted the same amount of data with less accumulated CPU utilization (up to 82.4% of the server accumulated CPU utilization was reduced from HTTP 1.1 Persistent Connection). As for the client overhead, Object Packaging resulted in less client response time and client accumulated CPU utilization than those for HTTP 1.1 Persistent Connection by up to 48.6% and 57.1%, respectively. Reduction in the number of transmitted packets and traffic load (see [4] for this) could be a reason for lower client overhead in Object Packaging. Object Packaging achieved these improvements by utilizing unique properties of requesting multiple small files in a burst with minimum changes in the existing protocol, while it does not require any hardware investment. Object Packaging will be an efficient solution for reducing the overhead not only at the server but also at the client, especially when link bandwidth is not a performance bottleneck. The results of the experiments also suggest the significance of disk I/O overhead for web server in high speed networks. Regarding the relationship between disk I/O overhead and CPU utilization, our speculation is that eliminating some disk I/O calls reduced CPU utilization at the server.

A drawback in Object Packaging is that object packages

must be available before client requests arrive at a server. In this regard, an extension of object packaging for dynamically generated files, such as those generated by CGI, is currently being developed. Other future work includes performance evaluation for client side caching and comparison to HTTP 1.1 Pipelining, which allows a client to establish multiple connections in parallel.

6. References

[1] Zona Research, "The Economic Impacts of Unacceptable Web-Site Download Speeds", Zona Research white paper, 1999. *http://www.webperf.net/info/wp_downloadspeed.pdf*

[2] P. Markatos, "Speeding-up TCP/IP: Faster Processors are not Enough," *Proceedings of the 21st IEEE International Performance, Computing, and Communications Conference*, April 2002.

[3] P. Druschel, "Operating System Support for High-Speed Networking," *Communications of the ACM*, vol. 39, no. 2, pp. 41-51, September 1996.

[4] H. Fujinoki and K. Gollamudi, "Web Transmission Delay Improvement for Slow and Busy Web Servers," *Proceedings of the IEEE 27th Conference on Local Computer Networks*, pp. 345-347, October 2002.

[5] J. Wang, "A Survey of Web Caching Schemes for the Internet," *ACM Computer Communication Revie*w, vol. 29, no. 5, pp. 36 - 46, October 1999.

[6] GVU's WWW User Surveys, *Georgia Institute of Technology* URL*: http://www.gvu.gatech.edu/user_surveys*

[7] V. N. Padmanabhan and J. C. Mogul, "Improving HTTP Latency," *Computer Networks and ISDN Systems*, vol. 28, no. 1, pp.25-35, December 1995.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, Hyper Text Transmission Protocol – HTTP/1.1, RFC 2616, *Internet Engineering Task Force*, June 1999.

[9] H. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, and C. Lilley, "Network Performance Effects of HTTP/1.1, CSS1, and PNG," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 4, pp. 155-166, October 1997.

[10] J. C. Mogul, "The Case for Persistent-Connection HTTP," *Computer Communication Revie*w, vol. 25, no. 4, pp. 299–313, October 1995.

[11] M. Arlitt and C. Williamson, "Web Server Workload Characterization: The Search for Invariants," *Proceedings of the 1996 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, pp. 126-137, Philadelphia, May 1996.

[12] H. Fujinoki Homepage URL: *http://www.siue.edu/~hfujino*

[13] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proceedings of Performance ACM SIGMETRICS*, pp. 151-160, June 1998.

[14] M. Busari and C. Williamson, "On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics," *Proceedings of IEEE INFOCOM*, pp. 1225-1234, April 2001.

[15] K. Christensen Tool Page URL: *http://www.csee.usf.edu/~christen*