N for the Price of 1: Bundling Web Objects for More Efficient Content Delivery*

Craig E. Wills
Computer Science Dept.
Worcester Polytechnic Institute
Worcester, MA 01609 USA
cew@cs.wpi.edu

Mikhail Mikhailov
Computer Science Dept.
Worcester Polytechnic Institute
Worcester, MA 01609 USA
mikhail@cs.wpi.edu

Hao Shang
Computer Science Dept.
Worcester Polytechnic Institute
Worcester, MA 01609 USA
hao@cs.wpi.edu

Abstract:

Persistent connections address inefficiencies associated with multiple concurrent connections. They can improve response time when successfully used with pipelining to retrieve a set of objects from a Web server. In practice, however, there is inconsistent support for persistent connections, particularly with pipelining, from Web servers, user agents, and intermediaries. Web browsers continue to open multiple concurrent TCP connections to the same server.

This paper proposes a new idea of packaging the set of objects embedded on a Web page into a single *bundle* object for retrieval by clients. Our analysis indicates that if embedded objects on a Web page are delivered to clients as a single bundle, the response time experienced by the clients is as good as or better than that provided by currently deployed mechanisms. We also show that, relative to the currently used retrieval methods, our approach reduces the load on the network and servers. The key contribution of our work is a mechanism that gives Web servers better control over the number and duration of TCP connections they support. Implementation of the mechanism requires no changes to the HTTP protocol.

Keywords: Web Performance, HTTP, Persistent Connections, Delta Encoding

Introduction

The World Wide Web has significantly evolved over the past few years. The amount of offered content and the number of content consumers have grown exponentially, complexity and richness of content has increased, functionality and performance of Web servers and user agents has improved. Web transactions, carried over the HyperText Transfer Protocol (HTTP) [3,8] running on top of the Transmission Control Protocol (TCP) [21], account for 70-75% of traffic on the Internet backbone, according to one study [5]. With the Web being the main application on the Internet, it is vital to ensure the efficient use of network (and server) resources by Web transfers while quickly delivering the requested content to end users. In this paper we propose a new idea of packaging the set of objects embedded on a Web page into a single *bundle* object for retrieval by clients. This idea is intended to address problems we see with current approaches for retrieving multiple objects needed to render a Web page.

*This work is partially supported by the National Science Foundation Grant CCR-9988250.

WWW10, May 1-5, 2001, Hong Kong.
ACM 1-58113-348-0/01/0005.

Versions 0.9 and 1.0 of the HTTP protocol are based on the model that uses a new TCP connection for each request/reply exchange with the Web server. To speed up the retrieval of content, many popular Web browsers open multiple concurrent TCP connections. Such a model, whether implemented with serialized or concurrent TCP connections, makes inefficient use of network and server resources. The overhead of setting up and tearing down each TCP connection contributes to router congestion. The operating system of the Web server also incurs per connection overhead and experiences TCP TIME_WAIT loading for each closed TCP connection. Response latency is also affected when a new TCP connection is created for each request and because each transfer independently goes through the TCP slow start phase.

Early on in the development of the Web this inefficiency of the simple HTTP model was realized and addressed by a number of proposals [23,9,17] advocating the use of a single TCP connection for multiple request/reply exchanges. Two new HTTP methods were also suggested, GETALL and GETLIST, which could be used to get all objects from a Web page in a single retrieval, and to get an arbitrary list of objects from a server in a single request [20]. These efforts contributed to the decision to adopt persistent connections as default in the HTTP/1.1 protocol, which was recently standardized by IETF [8,12].

HTTP/1.1 compliant clients and servers SHOULD (not MUST) (for the definition of these terms as applied to standards see [4]) permit persistent connections, although either side is allowed to terminate a persistent connection at any time. Both clients and servers can also indicate their unwillingness to hold a connection as persistent by including a "Connection: close" header as part of the HTTP request or reply respectively.

This flexibility in dealing with persistent connections was deemed necessary, particularly for servers, during the development of the HTTP/1.1 specification, but has led to mixed success in the current use of persistent connections. Recent studies have shown that 70-75% of Web servers at popular Web sites support HTTP/1.1 persistent connections, meaning that more than one object was successfully retrieved over the same TCP connection [11,14].

However, support for persistent connections does not necessarily lead to reduced retrieval times for a set of objects from a server. In a recent study, one of the authors evaluated the impact of different strategies for using TCP connections to retrieve multiple objects from the same server on the end-to-end response time and found a number of interesting results [14]. First, a single persistent connection with serialized HTTP requests generally does not provide better response time than parallel requests. Second, a single persistent connection with pipelining generally provides better response than parallel requests, but pipelining was only supported

by about 30% of the servers. Moreover, performance benefits are lost if TCP connections must be re-established.

Overall, persistent connections address inefficiencies associated with multiple concurrent connections, and when successfully used with pipelining can improve the response time. In practice, however, there is inconsistent support from Web servers, user agents, and intermediaries for persistent connections, particularly with pipelining. In this environment, browsers continue to open multiple concurrent TCP connections to the same server [26,2], which leads us to consider more efficient approaches for retrieving multiple objects from a server.

This work proposes a new approach to the problem of delivering multiple Web objects from a single server to a client. Our approach is more efficient than using multiple concurrent TCP connections and more deterministic than a persistent connection carrying a variable number of client requests. The basic idea is for the servers to group sets of related objects, such as objects needed to render a Web page, into a new object, called a *bundle*, or a *package*. In its simplest form, a bundle is a concatenation of a URL string, HTTP headers and content for each of its constituent objects. Servers advertise the availability of a bundle (or bundles) for each of their container (HTML) pages via a new HTTP response header Bundles (or perhaps via an HTML META tag). Clients understanding this header can then choose to either issue a single HTTP request (over an existing or a new TCP connection) and fetch the entire bundle or to ignore the availability of the bundle and retrieve all remaining objects for the container page as is currently done. Upon obtaining the bundle, clients recover the objects encapsulated in it and cache each of them using the supplied URLs and HTTP headers. As clients render the page, the embedded objects should already be cached, but if a bundle does not contain a complete list, then clients simply retrieve the needed objects from the server as is currently done.

An origin server or a server in a Content Distribution Network (CDN) can use this approach to serve multiple objects grouped together. In evaluating this approach, our focus has been on a set of objects necessary to fully render a Web page. In general, bundles could include objects that have some other form of relationship than being part of the same page. For example, all "popular" images at a site could be packaged into a bundle.

In the remainder of the paper we discuss the details of this approach and examine its impact on the entities (network, servers, clients) involved in the handling of the HTTP requests and responses. We also evaluate the use of server side compression on the contents of bundles and examine the interaction between our approach and client side caching. We conclude with a summary of our findings and a discussion of directions for future work.

Packaging Objects into Bundles

A common way to distribute software on the Internet is to package a collection of related files into a single file, using tools such as *tar* and *gzip*. We are also aware of Web sites that package a subset of their content (mostly static resources) using similar techniques and provide a pointer to the resulting file on their home pages (see [22] as an example). We are unaware, however, of any work that has proposed and evaluated a more selective and automated packaging of related objects.

Basic Idea

The use of bundles is initiated by the content provider. In the simplest case, an origin server creates a bundle of all of the embedded objects contained on a Web page. The use of bundles is independent of whether the container page itself is static or is dynamically generated, assuming dynamically-generated content uses the same set of embedded objects. Each object in a bundle is represented by its URL, relevant metadata (HTTP headers) and contents. HTTP headers associated with each object, *local headers*, are specific to that object. For example, an object can have Content-Length (used by the client in recovering the encapsulated objects) or Last-Modified HTTP header associated with it. When a bundle is requested by a client, the Web server also assigns it a set of HTTP headers, as it would for any other object. In the context of bundles, we will refer to such headers as *global headers*. In cases where global headers have the same names as the local headers, the local headers always take precedence over global headers. Otherwise, individual objects within a bundle inherit the global headers. Since all objects served by the same Web server naturally share some common HTTP headers, such as Server or Date, serving a single bundle instead of multiple individual objects results in a simple *header compaction* mechanism. With many small objects included in a bundle, byte savings could be significant.

For serving objects embedded in a Web page, a server can adopt a convention that bundles have the same names as their respective pages, except for the extension. For example, the bundle contains the embedded objects for the page. Bundles could be dynamically computed upon a client's request or could be precomputed. Dynamic construction of bundles is likely to incur substantial overhead due to a Web server parsing the page to determine which objects to include in the bundle. Precomputation avoids critical path processing but requires additional storage at the server since both bundled and original objects are stored. Popular objects, such as logos embedded on all site pages, could be separated into a separate "frequently-used object" bundle. Ultimately, servers decide if, when, and how to construct bundles.

One approach for constructing bundles is to group objects on a page, or across pages, based on their relationships and change characteristics, as discussed in [28]. Such grouping is reminiscent of the notion of volumes [15,13,6]. For example, static or infrequently changing objects on a page can be packaged together. Frequently changing objects on the same page can be organized into a separate bundle--or not grouped at all. While clients need to retrieve more than one bundle for a page in such cases, each bundle contains objects with similar change characteristics and presumably cache control directives.

While processing a request for a Web page that has an associated bundle, a Web server includes an additional HTTP header in its response, advertising the availability of bundles to the client. For example, the server includes the "Bundles: home.bndl" header when processing a request for home.html. If the requesting client chooses to take advantage of the available bundle, it issues a single HTTP request (over the existing or a new TCP connection) and indicates its ability to accept a new content type, application/x-bndl. Upon obtaining the bundle, the client extracts individual objects from it and reconstructs their metadata. These objects can then be stored in the local client cache (whether the client is a user agent or a proxy cache). At this point, the client can discard the bundle itself, although it may want to retain meta-information about the bundle object for future retrievals (see Section 2.3). Clients that do not support bundles, or choose to ignore them, fetch embedded objects as usual. The use of bundles is optional for both clients and servers, and it does not require any changes to the current method of content retrieval.

Using Compression with Bundles

An obvious extension of simply concatenating a set of objects together is to use a compression tool such as *gzip* to reduce the size of the bundle for transmission (potentially for storage as well). In the likely case that bundles contain primarily images, which are already compressed, opportunities for additional compression come mostly from textual HTTP headers within the bundle. In cases where bundles encapsulate textual objects, such as HTML frames, Cascading Style Sheets or scripts to be executed on the client side (JavaScript that is not part of HTML, for example), applying compression could significantly reduce the size of the bundle.

Bundles and Caching

A passive Web cache, whether it is a browser cache or a proxy cache, can only serve objects which have been previously retrieved, subject to per-object restrictions. The first retrievals of a Web page and its associated objects would all result in cache misses and hence the availability of a bundle containing all embedded objects for the page would be of value to the client. As the client subsequently retrieves the same or other bundles from the same server, the client might obtain objects that it already has in its cache. We see three approaches addressing this negative interaction between bundles and caching: validating individual objects, validating entire bundles, and delta encoding of bundles.

Validating Individual Objects within a Bundle

The HTTP response header, rather than simply advertising the bundle, as "Bundles: home.bndl," could also include a list of objects encapsulated in the bundle and their sizes, as

Bundles: home.bndl (main.css size=sz1, img1.gif size=sz2, img2.gif size=sz3);

This list allows a client cache to decide whether the new bundle contains enough new objects to warrant its retrieval. If not, the client should ignore the bundle and retrieve individual objects as needed. Servers could also include other validators, such as Last-Modified or ETag, to aid clients in determining whether new object retrievals are needed.

Validating Entire Bundles

In the second approach, a client cache retrieves a bundle object, extracts and stores its encapsulated objects, discards the bundle itself, but retains its HTTP headers, especially the ones used for cache validation. For example, if the Last-Modified or ETag headers are present, they could be subsequently used by the cache to issue a GET If-Modified-Since (IMS) or a GET If-None-Match request to the server to verify the freshness of the bundle. If the bundle has not changed, the server will indicate so via the HTTP response status code 304 Not Modified. In this case, the client cache would have automatically validated multiple objects by issuing a single *aggregate* IMS request. If the bundle has changed, the server will reply with the HTTP response status code 200 OK followed by the body of the new bundle. If the bundle has changed due to only a subset of its encapsulated objects changing, its retrieval wastes resources. This problem leads to the third approach.

Delta Encoding of Bundles

The third approach uses delta encoding. The idea of the technique, described by Williams et al. [27], is for the origin server (or proxy) to compute a difference (delta) between an old and a new copy of an object and communicate that difference to the client, provided that the client has an old copy of the object. The client can construct the new object by applying the delta to the old object. Mogul, et al. [18] quantified the benefits of end-to-end delta encoding and compression and proposed extensions to the HTTP protocol to support the technique. These extensions are detailed in the recently published Internet Draft [16].

Since bundles are regular objects, the delta encoding technique can be uniformly applied to bundles just like it can be applied to any other object. Example delta encodings include those produced by UNIX *diff -e* or by *vcdiff* [10]. Delta encoding of bundles could also be implemented as a modified *rsync* mechanism [25]. This mechanism efficiently computes differences between two instances of the same file and is commonly used to update software packages produced with tools such as *tar*.

Upon receiving the delta encoded response, the client first reconstructs the older version of the bundle based on the preserved meta data of the bundle and individual cached objects, and then applies the delta to the result to produce the new version of the bundle. Problems arise when one or more of the objects from the old bundle are evicted from the client's cache, making the reconstruction of the old version of the bundle impossible. To address this problem, clients could cache bundle contents instead of discarding them or use a bundle-aware differencing algorithm.

The output of a bundle-aware differencing algorithm, given a new and an old version of a bundle, encapsulates all objects in the new version that are not part of the old version, and all modified objects in the new version. The output also includes a list of objects that were part of the old version, but are not in the new version, to help clients differentiate between removed and unchanged objects. Given that bundles are likely to contain mostly images, computing differences on an object-by-object basis rather than byte-by-byte basis makes sense. When the client receives such a bundle-aware delta, all it needs to have in its cache are the unchanged objects from the old bundle. If some of them were evicted from the cache, the client can retrieve them individually.

We now illustrate how bundle-aware deltas are specified within HTTP. Suppose a client obtained a bundle home.bndl from www.foo.com, cached all the objects encapsulated in it, discarded the bundle itself, and kept meta data for the bundle. If at a later time this client wants to obtain the current value of the bundle it can send the following request to the server (this example is adapted from [16]):

GET /home.bndl HTTP/1.1
Host: www.foo.com
If-None-Match: Bundle-ETag1
A-IM: diffe, vcdiff, bsync, gzip

In this example, the client indicates that it has a cached copy of the bundle, identified by ETag with the value Bundle-ETag1. Delta encoding is considered to be an instance manipulation, and the A-IM HTTP request header, short for Accept-Instance-Manipulation, indicates which delta encodings the client supports. The client can accept delta updates produced by UNIX *diff -e* and *vcdiff*. The *bsync* specification, short for "bundle sync," represents the previously described bundle-aware encoding algorithm. To our knowledge, there is no existing tool that does the proposed bundle differencing, although similar distribution synchronization tools exist. The client also indicates that it can accept compressed responses using *gzip*, whether or not they were delta-encoded.

If the entity tag for home.bndl has changed (say, to Bundle-ETag2), the server, if it supports delta encoding, will compute the difference between the current version of the bundle and the older one, whose

ETag value is Bundle-ETag1, and send the response to the client, as shown below:

HTTP/1.1 226 IM Used
Server: Delta-Aware Server/1.0
ETag: Bundle-ETag2
IM: bsync
Date: Sun, 12 Nov 2000 10:00:00 GMT

.....delta between new and old bundle.....

A new HTTP response code 226 IM Used is required to force HTTP/1.0 proxies to forward all instance-manipulated responses without storing them (other options are discussed in [16]). If the server does not support delta encoding or does not implement any of the algorithms supported by the client then it replies with the HTTP response code 200 OK and the body of the entire new bundle.

Interaction with Content Distribution

The final issue we address is the interaction of the use of bundles with the distribution of Web content across multiple servers. Results obtained in November, 1999, showed a relatively small fraction (15%) of 700 popular Web sites using more than one server to serve content for their home pages [14]. However, in August, 2000, we did a follow-up study of the home pages for 312 popular sites (not a strict subset of the previous set) and found that 63% of these sites used more than one server to serve the objects on the home page. We address the apparent disconnect between this trend to distribute content and our proposal of consolidating it in two ways.

First, the distribution of a relatively small number of objects to different servers does not appear to be a good idea in terms of performance, regardless of whether bundles are available or not. Each new server requires a new DNS lookup on the part of the client as well as opening a new TCP connection.

Second, if many objects are served by auxiliary servers, for instance by a special "image server" at a site or by a CDN server, then these objects are good candidates to be packaged together into a single object. A bundle can be retrieved from a CDN server just as it can be from an origin server. A bundle could also contain objects assigned to multiple servers if these objects were all under the control of a single content provider.

Performance Impact

The previous section discusses how bundles can be constructed and used by Web clients and servers. This section examines the performance implications that the use of bundles could have on the network, servers, and clients. Bundles are unlikely to find wide adoption unless shown to be beneficial to both clients and servers.

Impact on Servers

HTTP uses TCP as its transport protocol. As a connection-oriented protocol, TCP maintains state at each end point of the connection thus placing per-connection memory overhead on each peer host. With a large number of simultaneously active TCP connections, a Web server's memory requirements can grow large. A more significant per-connection overhead occurs when a host performs an active close on the TCP connection. After sending the final TCP ACK, that host must keep the closed connection in the TIME_WAIT state for twice the Maximum Segment Lifetime (MSL) [24]. MSL is specified as 2 minutes [21], but commonly used values are 30 seconds, 1 and 2 minutes [24]. Consequently, the local port number used by the connection that is currently in TIME_WAIT state cannot be reused for 1 to 4 minutes. In HTTP, Web servers are the ones normally closing the connection. Therefore, depending on the local port range made available by the operating system (it is not always 1024-65535), and the client request rate, servers might be unable to open new TCP connections, even if few are currently active. HTTP throughput reductions of up to 50% have been reported [7]. TIME_WAIT loading of busy Web servers is studied in [7].

Intuitively, it is clear that Web servers should benefit from maintaining fewer TCP connections. Yet experience shows that support for persistent connections is not always implemented in Web server software or is deliberately turned off. The latter might be explained by the fact that lifetime of a persistent connection is *non-deterministic*--Web servers and clients are free to close a persistent connection when they deem necessary. The default connection-per-request model of HTTP/1.0, on the other hand, is *deterministic*--a server closes the connection after forwarding the response to the client. Our proposal utilizes a single TCP connection to effectively retrieve multiple objects while allowing the server to deterministically close the connection.

In addition to per-TCP connection overhead, Web servers also incur per HTTP request and per HTTP response overheads. Each HTTP request must be parsed, possibly logged, each HTTP response must be generated, including the HTTP response headers. Depending on the configuration and load, a Web server might need to fork a new process to service an incoming request.

Consider a Web server serving a Web page with n embedded objects under three different scenarios, as shown in Table 1. Under the connection-per-request model, used by HTTP/1.0 and HTTP/1.1 without persistent connection support, the number of TCP connections, HTTP requests and HTTP responses directly depends on n. With the persistent connections, the number of HTTP requests and responses is still proportional to n. Only when bundles are used the number of connections and requests needed to retrieve a Web page is constant.

Table 1: TCP Connections and HTTP Requests/Responses Resulting From the Retrieval of a Web Page with *n* Embedded Objects

	Number of	
Scenario	Conns	Reqs/Resps
Connection-per-Request	$n + 1$	$n + 1$
Persistent Connection	1	$n + 1$
Bundle Retrieved	1-2	2

Additional overhead is incurred at the servers because bundles must be created for appropriate Web pages, which requires tracking changes to these Web pages. Storage to maintain the bundles is also required. If delta encoding of these bundle objects is supported, then deltas between different versions need to be maintained. Web servers can control these costs by grouping objects in bundles according to object change characteristics. The set of rarely changing objects on a page would form one group. Frequently changing objects contained within a Web page can either be grouped in their own bundle or not grouped at all. In the latter case, clients retrieve these objects individually as is currently done.

Impact on the Network

Each TCP connection places load on the network. Reducing the number of TCP connections required to retrieve all objects on a Web page, and therefore increasing the goodput, is a direct contribution towards making the Internet and Web server sites less congested. In that respect, the use of bundles is as beneficial as the use of persistent connections.

Each HTTP request/response exchange also places load on the network. Consider a client retrieving a Web page with *n* embedded objects. Currently, irrespective of whether persistent TCP connection or pipelining is used, the client issues *n* separate and *identical* (except for the URL) HTTP requests. Examining the HTTP request headers generated by two popular Web browsers, Netscape Communicator 4.75 (NSC) and Microsoft Internet Explorer 4.0 (MSIE), we see that NSC generates 257 and MSIE generates 320 request bytes, not counting the size of the URL string, for each of these *n* requests. We do not consider the overhead introduced by these requests to be a serious issue for clients and servers, but using only one request to retrieve *n* objects--our approach--provides a way to lower the bandwidth requirements and particularly the amount of processing done by routers.

Impact on Clients

The primary objective of Web clients (browsers) is to retrieve and render Web pages as fast as possible. The two most popular Web browsers, NSC and MSIE, indicate their willingness to keep connections persistent: MSIE sends "HTTP/1.1" string in its HTTP request, and NSC sends "Connection: Keep-Alive" header, which was introduced as a way to support persistent connections with HTTP/1.0. In practice, however, browsers routinely open a number of concurrent connections--4, 6 and more [26,2]--even to the same Web server, to retrieve objects necessary to render a Web page. Our own investigation, performed by collecting traces of browser requests to real Web servers with *WinDump* [29], and analyzing them with *tcptrace* [19] and *Perl* scripts, revealed cases of a browser opening up to 17 concurrent TCP connections to a server.

Given this aggressive client behavior, it is important to understand the potential impact that the use of bundles could have on reducing client-perceived latency. To investigate this issue, we used data collected during the previous study on end-to-end Web performance [14] to estimate the time it would take to download all embedded objects on a Web page if they were packaged into a bundle. The available data contains the times it took nine clients, spread around the world, to download all embedded objects from the home page of 700 popular server sites in November 1999.

Five clients were located in the USA at AT&T Research Labs in New Jersey, Worcester Polytechnic Institute (WPI) in Worcester, Mass., the University of Kentucky, Hewlett-Packard Labs in Palo Alto, Calif., and the AT&T Center for Internet Research at ICSI in Berkeley Calif. The other clients were located at the University of Western Australia, a private site in Cape Town, South Africa, an academic network site in Trondheim, Norway, and a commercial site in Santiago, Chile. The 700 popular server sites were derived from popularity lists of MediaMatrix, Netcraft, 100Hot, Fortune500 and Global500.

Four protocol options were used: serial requests over non-persistent connections, up to four parallel requests over non-persistent connections, serial requests over a persistent connection, and pipelined requests over a persistent connection. The data also contains sizes for all retrieved objects and HTTP response headers. In addition to this data set, we obtained more data by having one client, located at WPI, run every six hours during a five-day period in November, 2000, on the set of 100 popular sites identified by 100Hot [1]. We collected data using the same methodology and software as was used in [14] and labeled results from this client "wpi100" in this paper.

For each Web page in the data set, the size of the bundle is computed as the sum of header and content sizes for all objects embedded on that page. To estimate the time it takes to download a bundle of a given size from a given server, ideally one would retrieve a static object, such as an image, of a similar size from that server. However, since bundles contain multiple objects, their size is substantially larger than that of the individual objects. The existing data set did not have objects large enough to represent the bundle object.

Instead, we estimated the download time of a bundle using detailed timing information stored in the existing data for each object retrieval. This information includes the time elapsed between when the first and the last bytes of a response were received by the client. Using this data, we computed the byte rate of the TCP connection for the largest object in each bundle, and used the result to estimate the download time for the bundle. While this is not an ideal approach, it provides a reasonable estimate of the latency that would be experienced by clients who opted to retrieve a bundle. Furthermore, we believe the estimate is a conservative one, because the byte rate of larger objects would be less influenced by TCP slow-start.

Table 2 shows measured and estimated latencies experienced from nine client sites. This list also includes the "wpi100" client. Table 2 includes measured retrieval times for only those servers that supported pipelining and served more than one object over a single connection. As reported in [14], servers in that category represented only about 30% of all servers serving more than one object. In the wpi100 results we found this value to be 39%. Numbers in parentheses are ratios of the numbers in the respective column to the parallel retrieval results.

Table 2: Results for Servers that Support Persistent Connections with Pipelining. All Retrival Times in Seconds (Ratio against Parallel Retrieval).

			Est.	Parallel	Pipeline	Est. Size of	Est.
Client	Server	Bundle	Bundle	Retrieval	Retrieval	Compressed	Compressed
Site	Measurements	Size	Time	Time	Time	Bundle	Time
aciri	223	33812	1.32 (0.7)	1.82 (1.0)	1.23 (0.7)	27207	1.07 (0.6)
att	167	27065	1.05 (0.7)	1.56 (1.0)	1.15 (0.7)	21294	0.82 (0.5)
aust	206	32516	10.89 (1.3)	8.47 (1.0)	7.36 (0.9)	26068	8.71 (1.0)
chile	207	33029	7.77 (1.0)	7.57 (1.0)	6.63 (0.9)	26517	6.50 (0.9)
hp	204	32991	1.42 (0.9)	1.55 (1.0)	1.17 (0.8)	26763	1.21 (0.8)
norway	133	16164	1.98 (0.9)	2.29 (1.0)	1.86 (0.8)	11102	1.64 (0.7)
safrica	209	28343	16.47 (1.3)	12.31 (1.0)	12.38 (1.0)	22045	12.84 (1.0)
uky	200	35360	3.76 (0.6)	6.05 (1.0)	3.04 (0.5)	28541	3.34 (0.6)
wpi	198	33161	4.23 (0.6)	6.62 (1.0)	4.08 (0.6)	26838	3.51 (0.5)
wpi100	1227	31192	0.57 (0.7)	0.78 (1.0)	0.71 (0.9)	23402	0.49 (0.6)

As reported in [14], the pipelining results are generally better for the set of servers that support it. The table also shows the estimated time to download the bundle object for each of these servers. As shown, the download time for the bundle object is competitive with pipelining and generally better than parallel retrievals except for the clients in Australia and South Africa. In addition, we looked at the results for all servers, irrespective of whether they supported pipelining, and also broke down the results based on the number of objects to be retrieved. The results are consistent with those in shown in Table 2. We expected that retrieval of a single bundle would show improved performance if more objects need to be retrieved, but the results do not support this hypothesis.

Impact of Data Compression

Table 2 also shows estimated download times of compressed bundles. We assume compression is done off-line and does not increase retrieval latency. We estimated the sizes for compressed objects by compressing a sample set of objects and their HTTP headers using *gzip*. For headers we saw a reduction of 20% for the first set of HTTP headers in the bundle with more extensive reduction of 80% for subsequent headers, as duplication in headers occurs. As expected, we found a small amount of compression available in image contents, with approximately a 6% reduction if multiple images were contained within a bundle. For textual objects, such as scripts and style sheets, we estimated a 75% reduction due to compression in our experiments.

Using these reduction percentages and the object types from our original results we estimated the sizes of compressed bundles and used the same methodology as before to estimate the download time for objects of these sizes. The last two columns in Table 2 show the results. Because bundles contain a large number of images, the amount of compression is relatively small--in the range of 20-30%. All bundles show some amount of compression due to the reuse of headers amongst all of the bundled objects. Despite the relatively small estimated compression, the download times for these compressed bundles are almost always better than for parallel retrievals and often better than when compared to the pipelined retrieval results.

While both the sizes and download times of compressed bundles are estimates, the results show promising response time for clients. Retrieval of compressed bundles does result in additional processing for the client, but tools such as *gzip* allow decompression to be done as the stream of content is received at the client.

Interaction with Client Caching

We examined the frequency at which the set of bundled objects changed at each server. For this study we used only the results from the wpi100 client. Retrievals were done every six hours from each of the servers to both assess any time-of-day effects on the download results and to allow us to gauge the frequency with which the set of embedded objects on these pages change. In terms of time-of-day effects, we did not find any patterns where the retrieval of a bundle object performed better at peak versus off-peak time.

Estimating the number of embedded objects that could be reused, we assumed that an embedded object was reusable if an object of the same name had been retrieved in a previous time period and if the size of the previously retrieved object matched the size of the current object. While this approach ignores cache control directives and the possibility of a changed object retaining the same size, it is a reasonable first-cut estimate. What we found was that on the first download of a Web page the average number of embedded objects on the page was approximately 12 and the average bundle size 31192 bytes. After the first retrieval, the average number of new (or changed) objects was 0.5 with an average size of 2689 bytes.

These reuse results are similar to those found in [14] and indicate that the set of embedded objects does not change frequently. These

results are good from a server standpoint because the set of embedded objects is relatively constant and new bundle objects would need to be recomputed relatively infrequently.

However, these results also show that if a client caches the set of objects it has previously retrieved for a page via a bundle object then it should not retrieve the entire bundle object for the page on subsequent accesses. In this case, the client would either retrieve individual objects as needed or use one of the approaches discussed in Section 2.3 to validate the previous bundle object as still fresh or retrieve just the delta-encoded updates.

Summary and Future Work

In this paper we have described a new approach to retrieving multiple Web objects. In our approach, servers package related objects at a site into a bundle object, which can be obtained by clients using a single HTTP request/response exchange. This approach lessens the need for clients to use parallel requests or try and maintain persistent connections with the server because fewer objects need to be retrieved.

We compared each of the three existing content retrieval approaches--parallel connections, persistent connections, persistent connections with pipelining--to our approach, from the standpoint of their performance impact on clients, servers and the network. Our analysis indicates that if embedded objects on a Web page are packaged and delivered to clients as a single bundle object, the response time experienced by the clients is as good as or better than that provided by currently deployed mechanisms. We have also shown that, relative to the currently used retrieval methods, our approach reduces the load on the network and servers.

The key contribution of our work is a mechanism that gives Web servers better control over the number and duration of TCP connections they support. We feel that our approach brings together the simplicity and determinism of the request-per-connection model and performance advantages of persistent connections. Our technique should be particularly useful for serving the most frequently accessed content at a site. The use of bundles allows small, relatively static, objects to be grouped to reduce requests and more efficiently use a single TCP connection. Its implementation requires no changes to the HTTP protocol.

Many directions for future work are possible. First, we need to better estimate the time to download bundle objects from Web sites. This direction will involve locating larger objects at these sites and using their download time to better estimate the retrieval time for bundles. We also need to explore different compression and delta encoding algorithms for reducing the amount of content needed to be sent by a server to a client. We need to implement the *bsync* bundle-aware delta encoding mechanism. The idea of bundles could also be combined with prefetching as servers identify groups of objects that are most likely to be used. Finally, we need to work on standardizing how bundles are computed along with how their existence is known to Web servers.

Acknowledgements

The authors thank Jeffrey Mogul and Balachander Krishnamurthy for comments on the original idea, and Mark Allman for answering questions regarding his recent paper. Authors are also grateful to the National Science Foundation for providing partial support for this research.

Bibliography

1. 100hot.com.
2. M. Allman. A Web Server's View of the Transport Layer. *Computer Communication Review*, 30(5), Oct. 2000.
3. T. Berners-Lee, R. T. Fielding, and H. F. Nielsen. Hypertext Transfer Protocol--HTTP/1.0. RFC 1945, May 1996.
4. S. Bradner. Key words for use in RFCs to indicate requirement levels. RFC 2119, IETF, Mar. 1997.
5. K. C. Claffy, G. J. Miller, and K. Thompson. The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone. In *Proceedings of the INET '98 Conference*, Geneva, Switzerland, July 1998. Internet Society. < Inet98 papers outreach www.caida.org>A>.
6. E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the Web using server volumes and proxy filters. In *ACM SIGCOMM'98 Conference*, September 1998.
7. T. Faber, J. Touch, and W. Yue. The TIME-WAIT state in TCP and Its Effect on Busy Servers. In *Proceedings of the IEEE Infocom '99 Conference*, pages 1573-1583, New York, NY, March 1999. IEEE.
8. R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext Transfer Protocol--HTTP/1.1. RFC 2616, June 1999.
9. J. Gettys and H. F. Nielsen. Smux protocol specification. W3C Working Draft, July 1998.
10. D. Korn and K.-P. Vo. The VCDIFF Generic Differencing and Compression Data Format. Internet Draft draft-korn-vcdiff-02.txt, Nov. 2000.
11. B. Krishnamurthy and M. Arlitt. PRO-COW: protocol compliance on the Web. In *Proceedings of the USENIX Symposium on Internet Technology and Systems*, San Francisco, California, USA, Mar. 2001. USENIX Association.
12. B. Krishnamurthy, J. C. Mogul, and D. M. Kristol. Key Differences Between HTTP/1.0 and HTTP/1.1. In *Eighth International World Wide Web Conference*, Toronto, Canada, May 1999.
13. B. Krishnamurthy and C. E. Wills. Piggyback server invalidation for proxy cache coherency. In *Seventh International World Wide Web Conference*, pages 185-193, Brisbane, Australia, Apr. 1998.
14. B. Krishnamurthy and C. E. Wills. Analyzing Factors That Influence End-to-End Web Performance. In *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, Netherlands, April 2000.
15. J. Mogul. An alternative to explicit revocation?, Jan 1996.
16. J. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Goland, A. van Hoff, and D. Hellerstein. Delta encoding in HTTP, Oct 2000.
17. J. C. Mogul. The case for persistent-connection HTTP. In *Proceedings of the ACM SIGCOMM '95 Conference*. ACM, Aug. 1995.
18. J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *ACM SIGCOMM'97 Conference*, September 1997.
19. S. Ostermann. tcptrace.

20. V. N. Padmanabhan and J. C. Mogul. Improving HTTP latency. In *Second International World Wide Web Conference*, Chicago, Illinois, USA, Oct. 1994.
21. J. Postel. Transmission Control Protocol. RFC 793, Sept. 1981.
22. National janet web caching service. Look for a local copy of this web site at the bottom of the page. < wwwcache.ja.net>A>.
23. S. E. Spero. SCP--Session Control Protocol V 1.1.
24. W. R. Stevens. *TCP/IP Illustrated, Volume 1, The Protocols*, volume 1. Addison-Wesley, Reading, MA, nov 1994.
25. Tridgell and P. Mackerras. The rsync algorithm, Nov. 1998. < rsync.samba.org>A>.
26. Z. Wang and P. Cao. Persistent Connection Behavior of Popular Browsers. Research Note, Dec. 1998.
27. S. Williams, M. Abrams, C. R. Standbridge, G. Abdulla, and E. A. Fox. Removal Policies in Network Caches for World-Wide Web Documents. In *Proceedings of the ACM SIGCOMM Conference*, pages 293-305, August 1996.
28. C. E. Wills and M. Mikhailov. Studying the Impact of More Complete Server Information on Web Caching. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.
29. WinDump: tcpdump for Windows. < windump netgroup-serv.polito.it>A>.