An Overview of the Web++ Framework

Bing Swen (Sun Bin)
Dept. of Computer Science Technology, Peking University, Beijing 100971, CHINA
bswen@cs.pku.edu.cn

Abstract

This paper presents an overview of the Web++ framework, a new mechanism of hypertext resource transmission specifically designed to further improve Web performance. The three components of the framework are described, along with results from experimental implementation and tests, which shows that the new architecture can be significantly faster than HTTP, with the improvement of transmission time around 70% to 400% and the same magnitude of packet savings. This, together with the full compatibility feature, validates the potential of the framework as a possible development direction to meet the challenges of future Web evolution.

Keywords: World-Wide Web, HTTP, Performance, Hypertext, Resource transmission

1 Introduction

The Word-Wide Web (hereafter the Web for short) is hitherto the most important application form of resource access on the Internet, with its load being dominant on most all TCP/IP networks. The future development of the Web is full of technical challenges since its existing architecture has come close to its limits and new technologies must be compatible with this architecture of tremendous use and investment. It is well known that HTTP/0.*x* and HTTP/1.0 [27] interact with TCP/IP in a low-efficiency manner, due to the initial protocol design of connection establishment per URL when retrieving resources. Retrieval of a complete Web page requires separate requests for formatted text and each linked objects, thus making network traffic bursty. In the past years much work has been done to address this issue [11-13, 16-19, 24, 28, 29], eventually leading to the new HTTP version [10]. HTTP/1.1 [22] significantly improves the efficiency of TCP use by introducing the mechanisms of persistent connection, request pipelining and fine control of caching.

Though a few other minor improvements and tune-ups are still possible to experiment and test [6, 8, 13, 17, 32, 33], it seems that there is little room left to further greatly improve Web performance under the existing HTTP infrastructure. As Web loads keep increasing quickly, the performance limits of present Web framework has emerged in many aspects. Though link speeds are also gradually increasing, the traditional "one request for one resource" transfer model is far from optimization in terms of network use. Actually, the structural characteristics of "hypertexted Web pages" still provide a great potential for performance improvement. A Web page is composed of multiple files, and they can be efficiently retrieved within a single transaction when sufficient information is available for the client to construct appropriate requests. The key point is to design a simple yet sophisticated mechanism to describe the detailed meta-information of each object in a compact form. Based on these considerations, this paper presents a novel mechanism to improve the Web and at the same time retain the simplicity and full compatibility. We call it *the Web++* [37,38], with emphasis on its compatibility with the existing Web.

2 Web++ Overview

The framework of Web++ includes three components:

- A new URL scheme *sttp* for identifying resources on the Web++. An STTP URL has the general format *sttp: // host : port / path ? parameters*. The default port of STTP service is 90. An example is *sttp: // wpp.org /index.stml*.

- The Structured Hypertext Transfer Protocol (STTP), defining a message set of requests and responses for the transmission control of resources on the Web++.

- The Structured Hypertext Markup Language (STML), for describing the structural information of Web pages, including information of the root page file, number and types of the linked objects, entity attributes of each object, file offsets and sizes of partial update, etc. With the meta-information description in STML, STTP can transfer resources in an efficient way.

The basic idea behind the Web++ is very simple. Namely, before sending a page file to the client, the server first processes the page into a more compact format (*structured hypertext*) with sufficient meta-information of each element related to the page, so that the client can handle them directly, without any repeated network transmission. We will refer to this process as STML compilation (or encoding) in this paper. On the other hand, the client also

presents sufficient meta-information about its desired objects to the server for the optimization of the compilation. Such processing of Web page allows the server and client to have a good knowledge of the contents that are transmitted. This helps make a more efficient use of TCP connection, and introduce new possible functionality to the Web as well.

Introducing a new URL scheme here is necessary for the client to differentiate between new request methods and those of HTTP (though using HTTP/1.1's *Upgrade* header helps switch protocols from HTTP to STTP, that would be an inefficient choice. See discussion below).

2.1 Typical STTP Transactions

By default, a request for an STTP URL will specify the server to send back an STML description of the URL rather than a single Web page file. Major STTP transactions are performed within two messages, that is, one submission and one reply. The typical 2-message process of a client to retrieve a Web page (*sttp://host/xdoc*) is as the following.

First, the client checks the local cache to see if the Web page has been visited, and if not, it tries to get the page together with all the related objects by sending a single (possible selective) *S-GET* request, expecting a single response from the server with the message body being a full STML document generated for the page;

If the page is already cached, then the client generates a partial STML document (*head-part*) listing the meta-information of all the interesting objects related to the page (including the page itself) obtained since the last visit, and send an *S-COMPARE* request, expecting a single response with an STML document containing all the necessary information of update for modified objects.

In each case, there are only two messages needed to transmit: one request (*S-GET* or *S-COMPARE*) and one response, which makes the most efficient page retrieval model. For a typical Web page with 10 linked objects (such as images, scripts, applets, style sheets, etc.), there are at least 11 requests and 11 responses (totally 22 messages) needed to transmit between an HTTP client and server (together with mutual acknowledgement for each packet). Though the request pipelining method usually helps reduce the latency, this model is far from optimization in terms of number of messages and usage of bandwidth. With STML and STTP, the number of messages is kept to the minimum: there are only one request and one response for the transmission of the 11 objects (the Web page file and all the linked objects), eliminating the other "stupid" 10 requests and responses. In section 3.3 we will see that the *S-POST* process can also be performed within two messages. Thus STTP reduces the network traffic by greatly reducing the number of client requests and keeping most of the packets in full size.

2.2 STML Summary

STTP servers and clients try to exchange sufficient information about a Web page and each object related to it. In order to record the structural information of Web pages, we need to introduce a very simple markup language called STML (the Structured Hypertext Markup Language). Roughly speaking, an STML document is a "hypertext of hypertexts", that is, a set of hypertexts that related to the same root hypertext. (The set may or may not be "closed" with respect to the closure of links.) Thus STTP may also be called the protocol for the transmission of a set of hypertexts. (For a summary of STML syntax see [37] and appendix B.) Here is an example,

```
[stml]
[head]
[root Name= "/index.html" Content-Type="text/html" Content-Encoding= "czip"  ETag= "0-54e-383712c4" Offset-Size= "2371/55720" Linked-Object="-text/html, +*/*"]
[object Name= "/../img/logo.jpg" Content-Type= "image/jpeg" Content-Encoding= "czip" ETag= "0-b7f-39e37ad2" Offset-Size= "62083/27960" /]
[/root]
[/head]
[body]
[object Name= "/index.html"]
  ...compressed content...
[/object]
…
[object Name= "/logo.bmp"]
  ...compressed content...
[/object]
[/body]
[/html]
```

2.3 STTP Messages

STTP uses the same message format as that of HTTP (the generic message format of [34]). The client uses request messages to retrieve resources, and the server answers the requests using response messages.

For the access of resources described in STML, STTP Currently introduces three requests: “STML GET”, “STML COMPAE” and “STML POST”, corresponding to three new methods for STML document retrieval, namely S-GET, S-COMPARE and S-POST. The method S-GET is used to retrieve an STML description of a resource, usually for the first time retrieval. The following is a "selective" S-GET:

S-GET /xpage STTP/1.0
Host: w++.w++.org.cn
Linked-Object: head-only, -image/, +image/gif, -audio/**

S-COMPARE is used to realize the most efficient cache-based Web page revisiting model. It constructs a partial STML document for update comparison of all objects related to the revisiting page. For example, when user specify an URL sttp://wpp.org/index.html that has been visited, the client issues the following message:

S-COMPARE /index.html STTP/1.0
Host: wpp.org
Linked-Object: -text/html -text/xml +image/ local-only*
ETag: 0-85f-724334c4 // ETag of the original STML document

[head]
[root Name= "/index.html" Content-Type="text/html" Offset-Size="502/27371" ETag= "0-54e-383712c4" Linked-Object="-text/html, +/*"]*
[object Name="/logo.jpg" Content-Type="image/" Offset-Size="27960/66808" ETag= "0-23f-626854c4" /]*
[object Name= "/menu.js" Content-Type="text/" Offset-Size="94920/8033" ETag="0-31d-652413c4" /]*
[/root]
[/head]

The S-POST method is used when the client needs to send some data to the server for processing, similarly to HTTP POST method. STTP supports a caching based post method so that the client may get rid of extra interactions, keeping the total messages to the minimum (that is, two messages). This is achieved using the S-POST method together with a new header, Followed-By. For example,

S-POST url-1 STTP/1.0
Host: wpp.org.cn
Linked-Object: …
Followed-By: url-2 // next stop after posting

*[head] …… [*head-part for url-2*] [/head]*

……post-body……

An STTP client should understand all HTTP responses in addition to the new ones, which begin from status code 600. STTP status code has the following categories:

100 ~ 599: HTTP status code
600 ~ 999: STTP status code
- 6xx: successful
- 7xx: redirection
- 8xx: client error
- 9xx: server error

For example, 600 – STML transfer OK; 704 – STML not modified (ETag’s the same); 71x – STML partial update, where 710 – only root page modified; 711 – only linked object(s) modified; 712 – linked objects added; 713 – linked objects removed.

2.4 STTP Servers and Clients

There are a few interesting issues in designing and implementing STTP servers and clients.

The ubiquitous use of server-side scripts (e.g., as database access interfaces) provides a large amount of dynamic contents in Web pages. Typically only a small part of a Web page is marked as dynamic content [2, 5, 20]. Therefore, partial update can be greatly helpful. With the combination of the *Offset-Size* attribute and the HTTP *Content-Range* header, partial update of a single object can be efficiently realized in STTP. For example, in a Web page (or non-root object) there are two parts (marked between specific tokens) corresponding to dynamic contents,

……<%!?# … #?!%> ……<%!?# … #?!%> ……

0 r1 r2 r3 r4 r5

When constructing a response for this page, the server may indicate that the page has two parts that are dynamic using a '+' indicator at the corresponding offset/size values,

[object …… ETag = "0-54e-383712c4" Content-Range="0-r1/, r1-r2/*, r2-r3/*, r3-r4/*, r4-r5/*" Offset-Size="o1/s1,+o2/s2, o3/s3, +o4/s4, o5/s5" … /]*

Then when revisiting the page, the client issues an *S-COMPARE* request with the information

[object …… ETag = "0-54e-383712c4" Range="r1-r2/, r3-r4/*" …]*

The server may then send only the dynamic contents for update (if the root page is not modified). In partial update messages, the server should treat the entity tags of dynamic pages as weak validators [22], which are not affected by dynamic contents.

If two Web pages share some related objects, then the requests for these pages are related by cache information. Some techniques are necessary to handle related requests efficiently. As the first choice, the client may use an *S-COMPARE* request with an "up-to-date" *head-part* to request the page. If the page has been visited and cached, it then simply updates the cached *head-part* extracted from the STML document using the newly cached objects, and everything goes the usual way. The other choice is to first get the root page description using a *head-only S-GET* and then retrieve all the other objects via an ordinary *S-COMPARE*, as discussed in section 3.3. This is usually the most reliable way for such a purpose, but needs two requests.

The server should treat the STML document *ETag*s provided by client requests in a special way, that is, as a "necessary condition" of update (or a sufficient condition of no update): if the client's *ETag* is the same as that of the STML document on the server side, then no update is necessary; if the two are not the same, then the server needs to further make a thorough update check for each item listed in the *head-part*, possibly adding new linked objects (712 response).

Since STML documents may be related (when an object is related to multiple pages), both the server and client do not have to maintain full STML documents. The server should maintain only the *head-part*s of its Web pages, and construct the corresponding *body-part* based on information of requests (though the *body-part* may be cached after the first request). If the server detects that a local object related to a Web page has been modified, it simply adjusts the *Offset-Size* values of the modified object and all the others that occur after the object. For a single Web page, the server may choose to maintain a "complete *head-part*" that includes the descriptions of all the linked objects, and then construct a version for each request by removing uninteresting objects according to the *Linked-Object* information. For frequently visited and/or infrequently modified pages, the sever may pre-make several versions of *head-part* corresponding to some most possible *Linked-Object* options to optimize performance.

The client should maintain a local cache for only the *Etag*s, *head-part*s, and various information and contents of linked objects extracted from STML documents, but not the documents themselves. Maintaining a cache for STML documents on client side is not necessary and can be a big burden. The client would have to do "incremental STML compilation" (rebuild the documents) for related pages when linked objects are updated, which is not a trivial work – the *offset-size* values (offsets) of the linked objects are usually not maintainable, since the STML documents may be stale, so are the *ETag*'s of STML documents. Since the *ETag* of an STML document on the client side may be an old one, the server should not infer update and resend a whole STML document based only on the (weak) *ETag* provided by the client's request, as discussed above.

3 Web Compatibility

STTP is fully compatible with HTTP/1.*x*. STTP retains all HTTP requests and responses while supporting new messages, so that STTP clients and servers can recognize all HTTP messages. This means HTTP is a strict subset of STTP. The advantage of STTP's compatibility with HTTP is that HTTP and STTP clients/servers can coexist and communicate with each other. An existing HTTP client can talk to an STTP server as if talking to an HTTP server, and an STTP client can also talk to an existing HTTP server after getting the very first response (which requires HTTP/1.*x* rather than STTP/1.*x*). This aim is very significant for saving the investment on both the server and client sides of the Web, and crucial for the successful transition.

The most obvious behavioral difference is that by default, STTP clients use *S-GET* and *S-COMPARE* methods to retrieve resources in STML format while HTTP clients use *GET* to retrieve a single object. The default port of STTP service is 90, though other TCP ports can also be used. When retrieving resources on an STTP server via HTTP, the client should explicitly specify the port number used by the STTP server. For example, using the following URLs, *sttp://wpp.org.cn/*, and *http://wpp.org.cn*:*90/*, the client should present exactly the same content to the user.

An STTP client may first use the *sttp://* scheme to retrieve resources on a server. If the server returns status code indicating HTTP client error (400, 403 or 405), then it should be an HTTP server and the client may then try the *http://* scheme. On the other hand, an STTP server can easily differentiate between HTTP and STTP clients from the version field of the request line, in addition to the methods used.

4 Related Work

We are not aware of any other work that uses a special transfer encoding together with a transfer control mechanism to speed up HTTP transactions, though the performance problem of HTTP has been widely studied in the last decade, and several methods have been proposed to improve Web latency.

To improve Web services on existing networks without any hardware update is to improve the transfer protocols used by the Web. The lower-level TCP is a firm foundation of today's Internet, so the source of possible improvement is HTTP . The major aspects are: (1) connection reuse, to avoid or alleviate TCP slow-start, which is represented by the work on Persistent HTTP (P/HTTP) [12, 16, 19]; (2) pipelining of client's requests, to reduce multiple request processing time [18]; (3) caching of server's responses, which is the topic of much previous work [6, 8, 17, 18, 31-33]. Most of the suggested improvement methods of significance have been integrated into HTTP/1.1 [10, 22].

Web++ aims at new mechanisms to further reduce message transfer time and provide more efficient caching support using transfer models of encoded Web pages. There are several previous works that are close to this aim.

The "collection resource" of WebDAV [25] uses a multipart/related MIME entity to represent a WebDAV resource as a single document, based on an XML syntax for describing resources. Though it is useful, a collection is not a compact and efficient description of Web pages. For example, there are no provisions for efficiently locating and updating objects in a collection (at file-offset level). Collection is not intended to be an ideal format of transfer encoding to enhance the performance of the Web.

The most relevant work related to STML is MHTML by Palme and Hopmann [21]. It defines the use of a MIME multipart/related structure to aggregate a text/html root resource and the subsidiary resources it references, and specifies a MIME content-header to reference each resource within the composite e-mail message. Though claimed to be able to be employed by other transfer protocols (e.g., HTTP or FTP) to retrieve a complete Web page in a single transfer, MHTML has several obviously insufficiencies to be seriously considered for that purpose. First, it does not provide sufficient and/or efficient meta-information to completely describe the document elements of a Web page, such as the information of number, size, offset, time of creation and modification, entity tag (ETag), etc of each subsidiary resource (or linked object called by this paper). And thus second, it does not provide support for caching the aggregated resources that have been retrieved, which is essential for the scalability of the Web. Finally, as a media encoding specification, it dose not necessarily provide any transfer control methods for the access of MHTML files.

Franks [35] proposed a primitive MGET method using multiple If-Modified-Since header for the various objects requested. Before sending an MGET request the client must first get the base HTML file using a normal GET request. Padmanabhan and Mogul [19] proposed GETALL and GETLIST methods to make pipeline requests along with a simple scheme of Web page preprocessing. Both MGET and GETALL/LIST have fundamental inefficiencies as in the case of MHTML: no sufficient meta-information is provided for each linked object; the component extraction is primitive at best; and so that no effective support for object caching, partial revalidate and update, content encoding, etc.

The most recent relevant work to the idea of "batch-fetching" a web page and all of its related objects is the proposal to use bundles to transfer Web pages, presented by Wills et al [36], where 2 passes of request and response are used to retrieve a Web page and its contents separately. Since a bundle is a simple form of resource aggregation, it dose not provide a mechanism for the description of the detailed meta-information of the embedded objects. The major insufficiency of bundles and the similar proposals is in the difficulty to handle various partial modifications of related objects. It would be exceedingly difficult to design a uniform and consistent scheme of aggregate resource updating without the help of a structural information description. Bundle reconstruction, delta generation and updating would also bring significant load and contribute to user perceived latency, for these have to be done at retrieval time. In this regard, delta encoding of individual object would be preferable when only a few objects are constantly modified, as opposed to the intended use of bundles.

5 Experimental Implementation and Tests

To validate the effect of our mechanism, we made an experimental implementation (Apache HTTPd [1] based) to compare the elapsed time in transmission of an identical set of Web pages using HTTP/1.1 and STTP/STML. The test set consists of 20 different HTML files, containing 2, 4, 6, …, 40 linked images respectively. The files also include a paragraph of the same text, amounting to 1876 characters. The images are saved using different file names from the same JPEG file, which has 2471 bytes. The page with 40 images is also used to test the caching based retrieval with 0, 2, 4, …, 40 images locally cached.

The network environments tested include two typical connection conditions: a fast intranet and a slow dialup line. The intranet is a 100Mbps Ethernet LAN, with RTT < 1ms and MSS = 1460. The dialup line is a 48Kpbs PPP

modem line using a major public commercial dialup service, with RTT $\approx$ 220ms and MSS = 1460. On the intranet, there is one router hop between the server and the client, while on the modem line there are 8. In order to make up for network fluctuations, the tests were made after midnight at several weekends and most runs were repeated more than 10 times.

The performance tests of elapsed time and packet number and the results are listed in appendix A. The results show that STTP outperformed HTTP under all circumstances tested. For the first time retrieval, the improvement is around 70% on the LAN and 25% on modem line. For 50% update retrieval, the improvement are 170% and 60% respectively. STTP is superior to HTTP for revalidate tests, even though HTTP/1.1 has been dramatically improved over HTTP/1.0 at this aspect by exploiting request pipelining [18]. The later has a more significant impact since most resources on Web servers remain to be stable [2, 5], and even on some highly dynamic web sites files tend to change little when they are modified, and the variation ratio is often extremely small [20]. For update retrieval of average pages with less than a quarter of related objects that are frequently modified, a 4 or 5 times improvement is commonly expectable. The savings in terms of number of packets are of the same magnitude.

STTP also shows the desired scalability, that is, the faster the connection, the better it performed. Connection conditions are constantly improved, from which STTP will benefit more than HTTP.

6 Summary and Future Work

In this paper we describe the Web++ framework, which is intended to be a simple mechanism to further improve Web performance. STTP and STML are designed to be a flexible transmission control mechanism for the access of hypermedia resources, and at the same time sufficiently simple and efficient, which helps implementation and the compatibility with existing technologies. Adding STML handling to an HTTP server is usually a simple task (though adding it to HTTP browsers is somewhat more complicated).

Tests on a simple and primitive STTP server show that the STTP/STML mechanism can significantly improve Web performance without any hardware upgrades. Web++ retains full compatibility with the Web, and thus all existing Web resources are accessible by Web++ clients and servers, so are Web++ resources by present Web clients and servers. This ensures that existing systems still have their (equal) opportunities to access the same amount of resources as the new ones, and provides a graduate transition approach (most likely starting from the server ends).

The major shortcoming is that STML encoding, decoding and cache synchronization bring additional load for both the server and client. As discussed in the above sections, using a few specific caching methods, a significant part of the load can be optimized away. The cost is low on both the server and the client sides comparing to the improvement. And such load tends to be a smaller and smaller part as computer hardware technology is rapidly progressing, which is much faster than the improvement of the limits of communication connections. The Web++ framework provides a load balance between the communication hosts and connections.

The work planned in the near future includes the improvement of our STTP design and implementation, and larger scale and extensive experiments and tests on both research network environments and a few possible commercial sites. Another important work is the development of a Web++ proxy server, which is planned to construct from an STTP server using configuration options. Work worth doing also includes the development of draft specifications of HTTP and STML.

References

[1] Apache, The Group, URL http://www.apache.org/.

[2] Arlitt, Martin F. and Carey L. Williamson. *Web Server Workload Characterization: The Search for Invariants (Extended Version*). DISCUS Working Paper 96-3, Dept. of Computer Science, University of Saskatchewan, March, 1996. ftp://ftp.cs.usask.ca/pub/discus/paper.96-3.ps.Z.

[3] Braden, R., "Extending TCP for Transactions -- Concepts," RFC-1379, USC/ISI, November 1992.

[4] Braden, R., "T/TCP -- TCP Extensions for Transactions: Functional Specification," RFC-1644, USC/ISI, July 1994.

[5] Braun, H., and K. Claffy, "Web Traffic Characterization: An Assessment of the Impact of Caching Documents from NCSA's Web Server," *Proc. 2nd Int. WWW Conference*, Chicago, Oct. 1994. URL http://www.ncsa.uiuc.edu/SDG/IT94/ Proceedings/ DDay/claffy/main.html.

[6] Cohen, E., B. Krishnamurthy and J. Rexford, “Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters”, Proceedings of ACM SIGCOMM ’98.

[7] Connolly, Dan, WWW and OOP, http://www.w3.org/pub/WWW/OOP/Activity.html.

[8] Fan, L., P. Cao and J. Almeida, “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol”, Proceedings of ACM SIGCOMM ’98

[9] Frystyk, H., M. Spreitzer, B. Janssen, and J. Gettys, “HTTP-NG Overview” (draft-frystyk-httpng-overview-00.txt), Nov. 1998. Internet Draft, IETF.

[10] Gettys, Jim, “Hypertext Transport Protocol HTTP/1.1”, W3C, Oct. 1996. URL http://www.w3.org/Protocols/HTTP/Performance/.
[11] Habib, Md. A., M. Abrams, “Analysis of Sources of Latency in Downloading Web Pages”, Proceedings of WebNet 2000. URL http://vtopus.cs.vt.edu/~nrg.
[12] Heidemann, J., "Performance Interactions Between P-HTTP and TCP Implementation," *ACM Computer Communication Review*, 27 2, 65-73, April 1997. URL http://www.isi.edu/lsam/publications/phttp_tcp_interactions/.
[13] Heidemann, J., K. Obraczka, J. Touch, “Modeling the Performance of HTTP Over Several Transport Protocols”, June 1997. IEEE/ACM Transactions on Networking 1997. URL http://www.isi.edu/~johnh/PAPERS/Heidemann96a.html
[14] Ingham, David, Mark Little, Steve Caughey, Santosh Shnvastava, “W3Objects: Bringing Object-Oriented Technology to the Web”, The World Wide Web Journal, Issue 1, Dec 95, O'Reilly, http://www.w3.org/pub/WWW/Journal/1/ingham.141/paper/141.html
[15] Larner, Dan, “Migrating the Web toward Distributed Objects”, 1996, Xerox PARC. URL ftp://ftp.parc.xerox.com/pub/ilu/misc/webilu.html.
[16] Mogul, J. "The Case for Persistent-Connection HTTP", *Western Research Laboratory Research Report 95/4*, Digital Equipment Corporation, May 1995. Also in Proceedings of ACM SIGCOMM '95. URL http://www.research.digital.com/wrl/publications/abstracts/95.4.html
[17] Mogul, Jeffery, Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, "Potential benefits of delta-encoding and data compression for HTTP," *Proceedings of ACM SIGCOMM '97*, Cannes France, September 1997.
[18] Nielsen, H.F., Gettys, J., Baird-Smith, A., Prud’hommeaux, E., Lie, H., and C. Lilley. “Network Performance Effects of HTTP/1.1, CSS1, and PNG,” Proceedings of ACM SIGCOMM '97, Cannes France, September 1997.
[19] Padmanabhan, Venkata N., and Jeffrey C. Mogul. “Improving HTTP Latency”, Computer Networks and ISDN Systems, v. 28, pp. 25-35, Dec. 1995. Slightly revised version of paper in Proc. 2nd International WWW Conference '94: Mosaic and the Web, Oct. 1994, which is available at http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html.
[20] Padmanabhan, Venkata N., and Lili Qiu, “The Content and Access Dynamics of a Busy Web Site: Findings and Implications”, Proceedings of ACM SIGCOMM 2000.
[21] Palme, J., and A. Hopmann, “MIME E-mail Encapsulation of Aggregate Documents, such as HTML (MHTML),” RFC 2557, March 1999.
[22] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. “Hypertext Transfer Protocol – HTTP/1.1”, RFC 2616. June 1999.
[23] Spero, Simon E., "Progress on HTTP-NG," URL http://www.w3.org/pub/WWW/Protocols/HTTP-NG/htto-ng-status.html
[24] Spero, Simon E., “Analysis of HTTP Performance Problems,” July 1994. URL http://sunsite.unc.edu/mdma-release/http-prob.html, http://elanor.oit.unc.edu/http-prob.html.
[25] Stracke, J., “Encoding a DAV resource in MIME” (draft-stracke-webdav-mime-resource-00.txt), Feb. 1999. Internet Draft, IETF.
[26] T. Berners-Lee, L. Masinter and M. McCahill. “Uniform Resource Locators (URL)”, RFC 1738, Dec. 1994.
[27] T. Berners-Lee, R. Fielding and H. Frystyk. “Hypertext Transfer Protocol – HTTP/1.0”, RFC 1945, May 1996.
[28] Touch, J., J. Heidemann, K. Obraczka, "Analysis of HTTP Performance," USC/Information Sciences Institute, August, 1996. URL http://www.isi.edu/lsam/publications/http-perf/.
[29] W3C, “HTTP Performance Overview”, Oct. 1999. URL http://www.w3.org/Protocols/HTTP/Performance/overview.html.
[30] W3C, W3C's work on HTTP Next Generation (HTTP-NG), URL http://www.w3.org/Protocols/HTTP-NG/.
[31] Wang, J., “A Survey of Web Caching Schemes for the Internet”, *ACM Computer Communication Review*, Vol. 29 No. 5, Oct. 1997.
[32] Williams, S., M. Abrams, C. Standridge, G. Abdulla, and E. Fox. Removal Policies in Network Caches for World-Wide Web Documents. In *Proc. SIGCOMM ’9*6, pp. 293-305. Stanford, CA, August, 1996.
[33] Yu, H., and L. Breslau, “A Scalable Web Cache Consistency Architecture”, Proceedings of ACM SIGCOMM ’99.
[34] Crocker, D. H., “Standard for the Format of ARPA Internet Text Messages”, STD 11, RFC 822, August 1982.
[35] Franks, John. MGET proposal, October 1994, http://www.ics.uci.edu/pub/ietf/http/hypermail/1994q4/0260.html
[36] Craig E. Wills, Mikhail Mikhailov, Hao Shang, “N for the Price of 1: Bundling Web Objects for More Efficient Content Delivery”. In Proceedings of WWW10 (10th International World Wide Web Conference), May 1-5, 2001, Hong Kong (http://www10.org).
[37] Swen, Bing. Improving Web Performance Using Structural Information of Web Pages, Tech. Rept., ICL, CS Dept., Peking University, Jan. 2001. (Available at http://icl.pku.edu.cn/bswen/web++/w++intro.html)
[38] Swen, Bing. Speeding Up the Web Using the Web++ Framework. In Proceedings (CD-ROM) of WebNet 2001 Conference, WebTech Session. Orlando, Florida, October 23-27, 2001.

Appendix A STTP and HTTP Performance Comparison Tests

Table 1 and 2 are the results of three different tests, that is, the packet number and elapsed time for first-time retrieval, 50% update (half of the linked images cached) and reload. Reload or revalidate is revisiting a Web page where the contents are already available in a local cache. In our cases, revalidate of a cached page results in no actual resource transfer.

Table 1 Performance Comparison on a 100Mbps LAN

linked objects	first-time retr. (packets/sec.)						50% update (packets/sec.)						reload (packets/sec.)					
	HTTP		STTP		PR	AR	HTTP		STTP		PR	AR	HTTP		STTP		PR	AR
2	12	0.121	9	0.105	0.33	0.15	11	0.060	7	0.051	0.57	0.18	8	0.040	3	0.035	1.67	0.14
4	20	0.162	14	0.124	0.43	0.31	16	0.087	9	0.070	0.78	0.24	12	0.059	3	0.035	3.00	0.69
6	28	0.204	21	0.162	0.33	0.26	23	0.128	12	0.095	0.92	0.35	16	0.073	3	0.035	4.33	1.09
8	36	0.235	25	0.187	0.44	0.26	28	0.143	16	0.121	0.75	0.18	20	0.089	3	0.035	5.67	1.54
10	45	0.471	30	0.215	0.50	1.19	35	0.196	18	0.146	0.94	0.34	24	0.110	3	0.035	7.00	2.14
12	53	0.541	35	0.260	0.51	1.08	41	0.292	21	0.176	0.95	0.66	28	0.125	3	0.035	8.33	2.57
14	61	0.727	40	0.351	0.53	1.07	47	0.390	23	0.192	1.04	1.03	32	0.133	3	0.035	9.67	2.80
16	69	0.846	45	0.441	0.53	0.92	53	0.535	25	0.208	1.12	1.57	36	0.173	3	0.035	11.00	3.94
18	77	0.929	51	0.494	0.51	0.88	59	0.634	28	0.218	1.11	1.91	40	0.250	3	0.035	12.33	6.14
20	85	1.160	56	0.641	0.52	0.81	65	0.751	30	0.232	1.17	2.24	44	0.305	3	0.035	13.67	7.71
22	93	1.337	60	0.726	0.55	0.84	71	0.863	33	0.245	1.15	2.52	48	0.406	3	0.035	15.00	10.60
24	101	1.472	65	0.818	0.55	0.80	77	0.968	36	0.274	1.14	2.53	52	0.481	3	0.035	16.33	12.74
26	113	1.627	69	0.891	0.64	0.83	83	1.099	39	0.342	1.13	2.21	56	0.561	3	0.035	17.67	15.02
28	117	1.753	76	0.974	0.54	0.80	89	1.207	42	0.389	1.12	2.10	60	0.621	3	0.035	19.00	16.74
30	125	1.933	80	1.087	0.56	0.78	95	1.302	43	0.451	1.21	1.89	64	0.721	3	0.035	20.33	19.60
32	133	2.143	86	1.167	0.55	0.84	101	1.422	46	0.521	1.20	1.73	68	0.761	3	0.035	21.67	20.74
34	143	2.243	90	1.307	0.59	0.72	109	1.556	49	0.550	1.22	1.83	72	0.788	3	0.035	23.00	21.51
36	151	2.414	94	1.392	0.61	0.73	115	1.652	51	0.561	1.25	1.94	76	0.809	3	0.035	24.33	22.11
38	159	2.553	99	1.583	0.61	0.61	121	1.767	54	0.580	1.24	2.05	80	0.831	3	0.035	25.67	22.74
40	167	2.639	104	1.667	0.61	0.58	127	1.873	58	0.661	1.19	1.83	84	0.876	3	0.035	27.00	24.03
Total	1788	25.492	1149	14.592	0.56	0.75	1366	16.925	640	6.083	1.13	1.78	920	8.212	60	0.700	14.33	10.73

Table 2 Performance Comparison on a 48Kbps Modem Line

linked objects	first-time retr. (packets/sec.)						50% update (packets/sec.)						reload (packets/sec.)					
	HTTP		STTP		PR	AR	HTTP		STTP		PR	AR	HTTP		STTP		PR	AR
2	16	1.87	11	1.37	0.45	0.36	12	1.24	8	0.76	0.50	0.63	8	0.55	3	0.22	1.67	1.50
4	26	2.86	17	2.36	0.53	0.21	20	1.96	11	1.43	0.82	0.37	12	0.74	3	0.22	3.00	2.36
6	35	4.28	24	3.24	0.46	0.32	26	2.52	14	1.98	0.86	0.27	16	0.99	3	0.22	4.33	3.50
8	42	5.38	30	4.17	0.40	0.29	33	3.68	18	2.31	0.83	0.59	20	1.21	3	0.22	5.67	4.50
10	50	6.38	36	4.94	0.39	0.29	38	3.95	21	2.69	0.81	0.47	24	1.43	3	0.22	7.00	5.50
12	59	7.36	42	5.83	0.40	0.26	46	4.68	24	3.18	0.92	0.47	28	1.45	3	0.22	8.33	5.59
14	70	8.02	49	6.59	0.43	0.22	53	5.27	27	3.63	0.96	0.45	32	1.87	3	0.22	9.67	7.50
16	76	10.16	59	8.18	0.29	0.24	58	6.53	30	4.12	0.93	0.58	36	2.14	3	0.22	11.00	8.73
18	83	11.42	62	8.67	0.34	0.32	63	7.47	34	4.62	0.85	0.62	40	2.30	3	0.22	12.33	9.45
20	94	12.47	68	10.17	0.38	0.23	70	8.30	37	5.00	0.89	0.66	44	2.53	3	0.22	13.67	10.50
22	102	13.07	74	10.43	0.38	0.25	73	8.84	40	5.44	0.83	0.63	48	2.75	3	0.22	15.00	11.50
24	106	13.73	80	11.32	0.33	0.21	80	9.06	44	5.77	0.82	0.57	52	2.91	3	0.22	16.33	12.23
26	111	15.16	87	12.14	0.28	0.25	87	10.06	47	6.10	0.85	0.65	56	3.18	3	0.22	17.67	13.45
28	122	16.94	93	12.64	0.31	0.34	93	10.36	50	6.26	0.86	0.65	60	3.41	3	0.22	19.00	14.50
30	131	17.72	99	13.70	0.32	0.29	101	10.71	53	6.49	0.91	0.65	64	3.62	3	0.22	20.33	15.45
32	143	19.28	106	14.61	0.35	0.32	105	11.09	56	7.75	0.88	0.43	68	4.06	3	0.22	21.67	17.45
34	152	19.91	112	15.60	0.36	0.28	111	12.91	60	8.24	0.85	0.57	72	4.12	3	0.22	23.00	17.73
36	161	22.16	124	16.48	0.30	0.34	119	13.95	63	8.62	0.89	0.62	76	4.37	3	0.22	24.33	18.86
38	175	22.96	128	18.13	0.37	0.27	126	15.79	66	9.07	0.91	0.74	80	4.47	3	0.22	25.67	19.32
40	183	23.67	131	19.77	0.40	0.20	132	16.48	70	9.39	0.89	0.76	84	4.56	3	0.22	27.00	19.73
Total	1937	254.80	1431	200.34	0.35	0.27	1446	164.85	773	102.85	0.87	0.60	920	52.66	60	6.60	14.33	6.98

Note:

packet saving ratio $PR = (packet\text{-}no_{HTTP} - packet\text{-}no_{STTP}) / packet\text{-}no_{STTP}$
acceleration ratio $AR = (time_{HTTP} - time_{STTP}) / time_{STTP}$

Table 3 and 4 are the comparison of transmission time and packet numbers of a page with 40 linked objects and different numbers of objects being cached (the page is not cached). Again, STTP needs only one request for the revalidate of all the cached images and the retrieval of other files. The packets transmitted were solely used for resources transmission. All response packets (except for the last one) were in the full size.

Table 3 100Mbps LAN

cached objects	update reload (packets/sec.)					
	HTTP		STTP		PR	AR
0	167	2.078	112	1.702	0.49	0.22
2	163	2.043	105	1.508	0.55	0.35
4	159	2.013	102	1.367	0.56	0.47
6	155	1.963	96	1.262	0.61	0.56
8	151	1.913	91	1.251	0.64	0.53
10	147	1.873	86	1.072	0.71	0.75
12	143	1.783	82	1.031	0.74	0.73
14	139	1.722	78	0.992	0.78	0.74
16	135	1.662	69	0.911	0.96	0.82
18	131	1.598	65	0.762	1.02	1.10
20	127	1.528	61	0.711	1.08	1.10
22	123	1.462	54	0.601	1.27	1.43
24	119	1.392	49	0.471	1.43	1.96
26	115	1.342	45	0.436	1.56	2.08
28	111	1.272	39	0.330	1.85	2.85
30	107	1.226	33	0.261	2.24	3.70
32	103	1.167	28	0.231	2.68	4.05
34	99	1.061	22	0.200	3.50	4.31
36	95	1.042	18	0.170	4.28	5.13
38	91	0.982	12	0.150	6.58	5.55
40	87	0.921	7	0.055	11.43	15.75

Table 4 48Kbps Modem Line

cached objects	update reload (packets/sec.)					
	HTTP		STTP		PR	AR
0	201	21.70	140	21.04	0.44	0.03
2	198	21.42	133	19.36	0.49	0.11
4	195	21.26	130	19.14	0.50	0.11
6	187	21.20	123	18.07	0.52	0.17
8	181	20.87	114	17.17	0.58	0.22
10	177	20.57	106	16.43	0.67	0.25
12	173	18.43	100	15.19	0.73	0.21
14	166	17.94	94	14.28	0.77	0.26
16	163	17.26	87	12.93	0.87	0.33
18	161	16.17	80	12.02	1.01	0.35
20	155	14.75	74	10.93	1.09	0.35
22	151	13.82	67	9.83	1.25	0.41
24	147	13.32	61	9.04	1.41	0.47
26	142	12.30	54	7.91	1.63	0.55
28	139	12.09	46	7.17	2.02	0.69
30	136	11.65	40	5.66	2.40	1.06
32	131	10.27	34	4.64	2.85	1.21
34	128	9.73	26	3.68	3.92	1.64
36	122	8.77	20	2.61	5.10	2.36
38	110	7.01	13	1.73	7.46	3.05
40	91	4.21	7	0.60	12.00	6.02

Appendix B A Brief Summary of STML Syntactical Rules

Syntax notation: X-seq indicates one of more X's (X X ...), X-list is one or more X's separated by intervening commas (X, X, ...), and Xopt an optional X (X or empty). Bold font characters are terminal symbols of the syntax. Alternatives are listed on separate intended lines. Single line comments are indicated using //.

STML-Document
    meta-info-seq$_{opt}$ ***[stml]*** *doc-part* ***[/stml]***

meta-info-seq
    meta-info-seq meta-info
    meta-info-seq

meta-info
    [!stml *attribute-field-seq* ***/]***
    [!head *attribute-field-seq* ***/]***
    [!body *attribute-field-seq* ***/]***

doc-part
    head-part body-part$_{opt}$
    body-part head-part

head-part
    [head] *item-spec-seq* ***[/head]***

item-spec-seq
    item-spec-seq item-spec
    item-spec

item-spec
    root-spec *// at most one*
    dir-spec

root-spec
    [root *attribute-field-seq* ***]*** *object-description-seq*
      [/root]

dir-spec
    [dir *attribute-field-seq* ***]*** *object-description-seq*
      [/dir]

object-description-seq
    object-description-seq object-desc
    object-desc

object-desc
    object-description
    dir-description

object-description
    [object *attribute-field-seq* ***/]***

dir-description
    [dir *attribute-field-seq* ***/]***

attribute-field-seq
    attribute-field-seq attribute-field
    attribute-field

attribute-field
    stml-version
    HTTP-header-field
    name-attribute
    offset-size-attribute
    linked-object-type
    content-present *// partial document support*
    //...

stml-version
    Version **=** ***"STML/*** *digit-string* **.** *digit-string "*

name-attribute
    Name **=** *" URL "*

offset-size-attribute
    Offset-Size **=** *" offset-size-pair-list "*

offset-size-pair-list
    offset-size-pair-list **,** *offset-size-pair*
    offset-size-pair

offset-size-pair
    indicator$_{opt}$ *offset / size*

linked-object-type
    Linked-Object **=** *" indicator media-type-list "*

indicator
    +
    -

media-type-list
    media-type-list **,** *media-type*
    media-type

media-type
    MIME-media-type
    STML-part

STML-part
    head-only
    body-only
    head-body
    local-only *// include only local objects (at the same*
                // server). STTP default value
    remote-inline

content-present
    indicator *// default value +*

body-part
    [body] *body* ***[/body]***

body
    object-content-seq

object-content-seq
    object-content object-content
    object-content

object-content
    [object *attribute-field-seq* ***]***$_{opt}$ *content* ***[/object]***$_{opt}$

content
    CRLF *octet-seq* ***CRLF***

STML-comments *// may occur at anywhere outside* ***[...]***
                // pairs and object-content
    [* *octet-seq* ****]***