An Outline of Initial Design of the Structured Hypertext Transfer Protocol (STTP)*

Bing Swen (*bswen@pku.edu.cn*)
Computer Science Dept., Peking University, Beijing 100971, CHINA

Abstract This paper presents an introduction to the initial design of the Structured Hypertext Transfer Protocol (STTP), a compatible extension to the HTTP. It includes a new message set for the control of resource transmission, and the Structured Hypertext Markup Language (STML) for describing the structural information of Web pages. Experimental tests show that STTP can be significantly faster than HTTP, with the improvement of transmission time being around 70% to 400% and the same magnitude of packet savings, which is among the best performance improvement ever reported. The paper discusses the basic idea and major design considerations of these components, as well as a few important issues in developing STTP servers and clients.
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 appears to have come close to its limits in many aspects and new technologies must be compatible with this architecture of tremendous use and investment.

The Hypertext Transfer Protocol (HTTP) is the core protocol used to access resources on the Web. While sufficiently simple for implementation, use and rapid popularization, it is well known that HTTP/0.*x* and HTTP/1.0 [1] 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 object, thus making network traffic bursty. In the past years much work has been done to address this issue [2-11], eventually leading to the new HTTP version [12]. HTTP/1.1 [13] 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 [14, 15, 4, 6, 16, 17], it seems that there is little room left to further greatly improve Web performance under the existing HTTP infrastructure. The activity of HTTP Next Generation (HTTP-NG) [18, 19, 20] tries to make a stride further on the performance enhancement and functionality extension of the Web by radical architecture redesigns, even at the cost of compatibility with the current Web technologies. There were also attempts to treat Web resources as distributed objects and migrate the whole Web toward a distributed object system [21, 22, 23]. But there are strong reasons argued against such changes. In the real world of computer communication, and especially in the Web world where a vast amount of investment has been and is still being made, compatibility can be an essential issue for the success of any new technology. It is obvious that the future Web will grow out of (and be compatible with) the existing one, rather than be based on a completely new infrastructure.

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

* This research is supported by Chinese 863 Hi-Tech Plans under the project no. 2001 AA 112081.

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 STTP [24,25], with emphasis on its compatibility with the existing Web.

The rest of this paper is organized as follows. In section 2, we discuss the previous work related to our research. We then describe the major two components of the STTP framework in section 3 (STML and STTP messages). Section 4 discusses the compatibility with the Web. An experimental implementation of STTP and the test results are described in section 5. The last section is the summary and future work.

2 Previous 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[①]. 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) [3, 5, 8]; (2) pipelining of client's requests, to reduce multiple request processing time [7]; (3) caching of server's responses, which is the topic of much previous work [6, 7, 14-17, 28]. Most of the suggested improvement methods of significance have been integrated into HTTP/1.1 [12, 13].

STTP 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.

First, HTTP/1.1 206 (Partial Content) response supports a multipart media type, which enables a single response message to transfer multiple non-overlapping parts of a resource. This provides a flexible method for requests using ranges. But it cannot be used to transfer multiple resources in a single message.

The "collection resource" of WebDAV [29] uses a multipart/related MIME entity to represent a WebDAV resource as a single document, based on an XML syntax for describing resources. The collection is essentially an XML document with properties and *href*'s to other Web pages, images, etc. Though it is useful to encapsulate all aspects of such an enhanced resource, 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. 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 [30]. 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.

[①] T/TCP [38, 39] can help reduce connection establishment costs in HTTP, but the effect is approximated by other higher-level methods such as Persistent HTTP [28, 13].

Franks [31] proposed an *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 [8] proposed *GETALL* and *GETLIST* methods to make pipeline requests along with a simple scheme of Web page preprocessing. *GETALL* method specifies that the server should return an HTML page and all of its inlined (local) images in a single response. The client uses the *Content-Length* fields to split the response into its components. The server can either parse HTML files dynamically or keep a precomputed database of parsed files. The *GETALL* method can be easily added to HTTP as an ordinary *GET* with an additional header filed to indicate a batch processing. *GETLIST* is used to get a subset of components of an HTML document, avoiding the server returning all the images (caused by *GETALL*) when some of them are already in the client's cache. The typical operation would be that for the first time retrieval, a client uses *GETALL*; then when revisiting the same page, it uses a *GET* to retrieve the HTML file, after which it uses *GETLIST* to retrieve in one exchange all the images not in its cache, which is equivalent to a series of *GET*s sent without waiting for the previous one to complete. Both *MGET* and *GETALL/LIST* seem promisingly efficient, but there are also fundamental inefficiencies in these models 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* [32], where 2 passes of request and response are used to retrieve a Web page and its contents separately. Though performance improvement can be gained in some cases, it suffers from the same kind of difficulties as the above methods, which are particularly significant for the transmission of partially updated Web pages (e.g., script-based dynamic pages). 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. The three caching approaches of bundles, especially the delta encoding of aggregate resources, come close to this end, but again they didn't handle the "batch-selective updating" problem (updating all frequently modified objects along with their root page in one transaction) well either. 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.

3 STTP Overview

The framework of STTP includes two components:

- A 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.

- A extended message set of requests and responses for the transmission control of resources on the STTP (in addition to those defined by HTTP);

The basic idea behind the STTP 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).

3.1 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.

3.2 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). (For a summary of STML syntax see [24-27].)

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.

An STML document consists of a *head-part* and an optional *body-part*. The *head-part* is something like an index table of the items contained under the *[root]* item, with comprehensive descriptions for each entry. A complete STML document is actually a preprocessed HTML or XML document. 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]
```

The *head-part* consists of items of object specifications. The central one, *root-spec*, specifies the root object of an STML document and the objects that will be embedded (content inlined) in the document's *body-part*. The root object can be any sort of resource, but usually it is either a Web page file (HTML/XML) or a (sub)directory. A sequence of *dir-spec*s describes the directory structure of interesting resources, where the *root-spec* specifies the root directory of all other directories and files. The meta-information of objects is described as *attribute-field*s of corresponding items. Most HTTP message headers are reused as attribute descriptions. These well-defined headers provide a relatively complete set for the description of various important meta-information of resources and can be well integrated into the STML framework. Since the values of some HTTP header fields may have more than one token, we need a terminator symbol to indicate the end of an *attribute-field*. All *attribute-field*'s have the uniform format *Label="Value"*. That is, the value of an *attribute-field*, including values of any HTTP header field, must be occur between ".....", and we use '=' rather than ':' as the delimiter between a header name and its value.

Besides HTTP headers, some new headers are necessary, including *Linked-Object*, *Offset-Size*, *Name*, etc. A *[root]* object must have a *Linked-Object* attribute to indicate what kinds of media types will be included into the STML document. An example is:

```
[root Name="/index.html"… Linked-Object="-text/html, +image/*, local-only"]
```

which specifies that the STML document includes all image objects at the server's local file system linked to *index.html*, but not any other HTML files (except for the root page). *local-only* is the default value. By default, if a linked object of a root is not described in the root's *object-description-seq*, the content of the object is not regarded to be present (though it may be present). An attribute *content-present* may be used for explicit indication.

An STTP client also uses the *Linked-Object* attribute as a request header to specify the object types desired to be included in the requested STML document. The requested STTP server is expected to send back the (dynamically or pre-) generated STML document with the compilation directed by this *Linked-Object* head field, which will be copied to the *attribute-field-seq* of the *[root]* tag.

If both the *[root]* attributes and an *S-GET*, *S-POST* or *S-COMPARE* request's head have a *Linked-Object* field, then the request header field takes precedence. This is intended to allow the client to adjust its retrieval options at any time (e.g., when an STML *head-part* has been cached).

The *offset-size-attribute* description is used to indicate the start position and size (in byte) of the content (not including the surrounding *[object …]* and *[/object]* tags) of an embedded object. With this information, a client can find the content of an embedded object in an STML document very efficiently. Each embedded object (including the root page) must have the *Offset-Size* attribute.

Using multiple offset/size pairs as the value of an *Offset-Size* attribute, STML makes the description of inconsecutive objects possible, where each offset/size pair corresponding to an occurrence of the object's content in the STML document. For example, the description for an object

```
Offset-Size= "1008/512, 2025/512, 3742/384"
```

specifies that its content occurs at three different positions in the STML document. This facility provides good support for STTP servers that constantly generate dynamic data as the content of objects linked to Web pages, where the content can be divided to several parts, with static parts being pre-allocated in the *body-part*, and the generated parts being appended to the end of the *body-part* each time when sending the pages.

Compiling an HTML "source file" to an STML document is usually a trivial and efficient process, and STML generation is a typical domain to apply various "incremental compilation" techniques. Frequent and usually small rebuildings can be done incrementally. Usually, full STML documents are not necessarily maintained, especially for the client, since with information from the *head-part* of an STML, the *body-part* can be (re)generated very efficiently on both the server side and the client side. Incremental compilation may also help pregenerate "partial STML documents" (with only the *head-part*s).

3.3 STTP Message**s**

STTP uses the same message format as that of HTTP (the generic message format of [33]). The client uses request messages to retrieve resources, and the server answers the requests using response messages.

- Requests

For the access of resources described in STML, STTP Currently introduces three requests: "STML GET", "STML COMPARE" 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/*
```

The *Linked-Object* header indicates what media types are to be taken into account. The above request specifies that the client wants only the *head-part* of the STML description of *xpage*, without necessarily the information of any image (except GIF files) or audio objects related to *xpage*. Other *STML-Part* options include *body-only*, *head-body* and *local-only* (including only objects at the requested host). +*/* means to get all linked objects; *-text/html* means not to inline any other HTML pages pointed to in the root page. The first appearing and most concrete media type in the list takes precedence. When a non-HTML file is requested using an *S-GET* method, the server will ignore the *Linked-Object* header, considering only entity headers (*Accept*, *ETag*, etc).

When a client has cached a Web page and is revisiting the page, it may choose to send a selective *S-GET* request to get the STML *head-part* of the Web page, as described above. If the STML document has been modified (and so any linked objects), the *head-part* will have the changes. With the new version of *head-part*, the client knows well what objects related to the current Web page need to be updated, and then may request each one using ordinary HTTP *GET* method. For Web pages that are not constantly modified, this way can considerably reduce the number of requests and improve the performance. The number of messages needed is (2 + 2 * (the number of updated objects)), rather than (2 + 2 * (the number of all related objects)).

STTP further introduces another request method, *S-COMPARE*, 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]
```

Usually with this method the client sends back the STML *head-part* of the cached Web page to server for update comparison, indicating that it only needs to know whether anything of the specified media types related to the current page has been modified since the last visit, and send only the contents of all modified objects if so. The *head-part* is usually a partial one, containing only descriptions of interesting objects. For example, if the client has turned off the request for images, video or audio files, Java applets, etc., then it may not list any of these objects in the *head-part*, even if the original STML document contains them. Upon receiving the request, the server checks modifications for all objects listed in the *head-part*, generates and sends in a single response message a (partial) STML document for all modified objects. The server may also add some new objects of the specified media types, which are not listed in the *S-COMPARE*'s *head-part* but recently added to the updated Web page. With this response message the client then successfully updates its local cache and displays the page. There are totally two (one request and one response) messages in the process.

The items listed in an *S-COMPARE*'s *head-part* need not to be complete. Any related objects not listed are regarded as newly added ones by the server. But the server will actually construct update information only when the *ETag* of the STML document has been modified. For Web pages that are infrequently modified (such as electronic library or historical archives, etc), it is worth sending a "bare" *S-COMPARE* request (listing only the root page) for update query of all the linked objects. For example,

```
S-COMPARE /ourpast.html STTP/1.0
Host: w++.w++.org.cn
Linked-Object: -text/mls +image/*
ETag: 0-961-31da10a6  // ETag of the original STML document

[head][root  Name=  "/ourpast.html"  Content-Type="text/html"  ETag=  "0-425-756a4e7c"
Linked-Object="-text/html, +*/*"]
[/root][/head]
```

The message is small enough to be sent within one packet, and would be the most efficient update query method for such a kind of Web resources. We make use of this method in our experimental tests (described in section 5).

On the other hand, if an *S-COMPARE* request lists any "redundant" objects that are actually not related to the requested Web page, the server may simply ignore them.

The *S-POST* method is used when the client needs to send some data to the server for processing, similarly to HTTP *POST* method. After posting the data to the server, the client may receive a redirection response (302 or 303 Object Moved/Found) that indicates the client should go to another URL (usually predefined) to refer to the results or some new information. In HTTP, this is a multi-step process, leading to considerable latency. 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……
```

This request indicates that the client expects to retrieve resource at *url-2* using the STTP protocol, and the server needs to send back an STML document compiled for *url-2* for update. The combination of the *Followed-By* header and the *head-part* is equivalent to an *S-COMPARE* request. The *Followed-By* header requires that an STML *head-part* must be at the first of the message body (just following the empty line after request headers).

When using the *Follow-By* header, the client should have cached a *head-part*. The URL of *Follow-By* and its STML document are obtained from previous responses of *S-POST*. For the first time posting to *url-1*, the client does not know the ULR that will follow, so it uses a default value '*':

```
S-POST url-1 STTP/1.0
```

```
Host: wpp.org.cn
Linked-Object: …
Followed-By: *

……post-body……
```

Then the usual interaction happens and the server indicates the client to go to *url-2* to get an STML document. The client may cache this URL as well as the *head-part* when *S-GET*ing the document.

An *S-GET* request may also post some information as parameter of the request URL, and use the *Followed-By* header to realize an efficient *S-COMPARE* effect. For example,

```
S-GET url-1?parameter-string STTP/1.1
Followed-By: url-2   // next stop after posting

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

Thus there is a approximate equation that

```
S-COMPARE url  ~  S-GET url
                  Followed-By: url
                  [* head-part must be cached *]
```

- Responses

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

6*xx*: successful 7*xx*: redirection 8*xx*: client error 9*xx*: server error

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

```
STTP/1.0 710 Only root object modified
ETag: 0-57e-765712c4  // ETag of the new STML document

[head][root …]  …  [/root][/head]
[body] [*send only the partial update ("delta") of the root object*]
 ……
[/body]
```

Some HTTP headers are reused as STTP response headers, and a few new ones are introduced, which currently include *Linked-Object* and *Followed-By*, as explained above. When the requested STML documents are regarded as ordinary Web resources, most HTTP headers remain to be meaningful, though usually they are more appropriately used to be attributes of individual objects.

3.4 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 [34, 35, 36]. 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 [13], 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.

4 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.

An STTP client may first "venture" to use the extended requests (*S-GET*, etc) to retrieve resources on a server. If the server returns status code indicating an HTTP Client Error, then the server should be an HTTP server and the client may then try the HTTP's requests (*GET*, etc). 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.

5 Experimental Implementation and Tests

To validate the effect of our mechanism, we made an experimental implementation to compare the elapsed time in transmission of an identical set of Web pages using HTTP/1.1 and STTP/STML. The server is a modified version of Apache-1.3.14 [37], running on an HP NetServer LH3 and Windows 2000 Server. Client software used to retrieve the Web pages is Microsoft Internet Explorer 5.5 and a customized HTTP program that can parse STML descriptions. Both HTTP and STTP use persistent connections. The elapsed time recorded on the client side is normalized to the interval between the client sending the first request and the arrival of the last response. The server parses the requests and decides to send back STML or HTML files based on the request lines. The STML documents of Web pages are pre-generated and cached on the server side. We also estimated the overhead of STML generation and parsing and found it to be the order of magnitude of on-the-fly text compression, which relies heavily on the system being used.

The test set consists of 10 different HTML files, containing 4, 8, …, 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, 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 ≈ 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 The following. 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. We use the *S-COMAPRE* technique discussed in section 4 to achieve a constant (and minimum) overhead for page revalidate.

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.

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 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
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
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
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
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
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
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
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
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
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
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

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
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
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
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
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
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
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
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
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
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
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

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
4	159	2.013	102	1.367	0.56	0.47
8	151	1.913	91	1.251	0.64	0.53
12	143	1.783	82	1.031	0.74	0.73
16	135	1.662	69	0.911	0.96	0.82
20	127	1.528	61	0.711	1.08	1.10
24	119	1.392	49	0.471	1.43	1.96
28	111	1.272	39	0.330	1.85	2.85
32	103	1.167	28	0.231	2.68	4.05
36	95	1.042	18	0.170	4.28	5.13
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
4	195	21.26	130	19.14	0.50	0.11
8	181	20.87	114	17.17	0.58	0.22
12	173	18.43	100	15.19	0.73	0.21
16	163	17.26	87	12.93	0.87	0.33
20	155	14.75	74	10.93	1.09	0.35
24	147	13.32	61	9.04	1.41	0.47
28	139	12.09	46	7.17	2.02	0.69
32	131	10.27	34	4.64	2.85	1.21
36	122	8.77	20	2.61	5.10	2.36
40	91	4.21	7	0.60	12.00	6.02

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 [7]. The later has a more significant impact since most resources on Web servers remain to be stable [34, 35], 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 [36]. 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 some initial design aspects of STTP, which is intended to be a simple and effective mechanism to further improve Web performance. STTP and STML are designed to be a flexible transmission control mechanism for 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).

Experimental tests show that the STTP/STML mechanism can significantly improve Web performance without any hardware upgrades. STTP retains full compatibility with the Web, and thus all existing Web resources are accessible by STTP clients and servers, so are STTP 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 STTP framework provides a load balance between the communication hosts and connections.

The work planned in the near future includes further improvement of the STTP design and implementation, and larger scale and more extensive experiments and tests on both research network environments and a few possible commercial sites. Another important work is to develop of a full STTP proxy server, which is planned to construct from an STTP server using configuration options. Based on this experimental design and tests, currently we are making the next version STTP (called “STTP/0.9”) integrate more nicely with HTTP as an extension of HTTP (rather than a whole new protocol), with the intended flavor very much similar to the HTTP binding mechanism of SOAP/1.1 (but not of SOAP/1.2).

References

[1] T. Berners-Lee, R. Fielding and H. Frystyk. "Hypertext Transfer Protocol - HTTP/1.0", RFC 1945, May 1996.

[2] 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.

[3] 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/.

[4] 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

[5] 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

[6] 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.
[7] 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.
[8] 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.
[9] 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.
[10] 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/.
[11] W3C, "HTTP Performance Overview", Oct. 1999. URL http://www.w3.org/Protocols/HTTP/Performance/overview.html.
[12] Gettys, Jim, "Hypertext Transport Protocol HTTP/1.1", W3C, Oct. 1996. URL http://www.w3.org/Protocols/HTTP/Performance/.
[13] 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.
[14] 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.
[15] Fan, L., P. Cao and J. Almeida, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol", Proceedings of ACM SIGCOMM '98
[16] Williams, S., M. Abrams, C. Standridge, G. Abdulla, and E. Fox. Removal Policies in Network Caches for World-Wide Web Documents. In Proc. SIGCOMM '96, pp. 293-305. Stan-ford, CA, August, 1996.
[17] Yu, H., and L. Breslau, "A Scalable Web Cache Consistency Architecture", Proceedings of ACM SIGCOMM '99.
[18] Frystyk, H., M. Spreitzer, B. Janssen, and J. Gettys, "HTTP-NG Overview" (draft-frystyk-httpng-overview-00.txt), Nov. 1998. Internet Draft, IETF.
[19] Spero, Simon E., "Progress on HTTP-NG," URL http://www.w3.org/pub/WWW/Protocols/HTTP-NG/htto-ng-status.html
[20] W3C, W3C's work on HTTP Next Generation (HTTP-NG), URL http://www.w3.org/Protocols/HTTP-NG/.
[21] Connolly, Dan, WWW and OOP, http://www.w3.org/pub/WWW/OOP/Activity.html.
[22] 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
[23] Larner, Dan, "Migrating the Web toward Distributed Objects", 1996, Xerox PARC. URL ftp://ftp.parc.xerox.com/pub/ilu/misc/webilu.html.
[24] Swen, Bing(孙斌). 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)
[25] Swen, Bing(孙斌). Speeding Up the Web Using the Web++ Framework. In Proceedings (CD-ROM) of WebNet 2001 Conference, WebTech Session. Orlando, Florida, October 23-27, 2001.
[26] Swen, Bing(孙斌). A Brief Introduction of the Web++ Framework. In WWW2002 Conference Proceedings, Posters Session. Honolulu, Hawaii, USA. 7-11 May 2002.
[27] Swen, Bing(孙斌). An Overview of the Web++ Framework. In Proceedings of International Conferences on Info-tech & Info-net (ICII2001), Conference E (Information Network), E-13 (Web Technology). (Included also in Conference CDROM.) Beijing, Oct.29 - Nov.1, 2001.
[28] Wang, J., "A Survey of Web Caching Schemes for the Internet", ACM Computer Communication Review, Vol. 29 No. 5, Oct. 1997.
[29] Y.Goland et al. WebDAV. RFC2518, IETF, Feb 1999; Stracke, J., "Encoding a DAV resource in MIME" (draft-stracke-webdav-mime-resource-00.txt), Feb. 1999. Internet Draft, IETF.
[30] Palme, J., and A. Hopmann, "MIME E-mail Encapsulation of Aggregate Documents, such as HTML (MHTML)," RFC 2557, March 1999.
[31] Franks, John. MGET proposal, October 1994, http://www.ics.uci.edu/pub/ietf/http/hypermail/1994q4/0260.html
[32] 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).
[33] Crocker, D. H., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, August 1982.
[34] 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.
[35] 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.
[36] Padmanabhan, Venkata N., and Lili Qiu, "The Content and Access Dynamics of a Busy Web Site: Findings and Implications", Proceedings of ACM SIGCOMM 2000.
[37] Apache, The Group, URL http://www.apache.org/.
[38] Braden, R., "Extending TCP for Transactions -- Concepts," RFC-1379, USC/ISI, November 1992.
[39] Braden, R., "T/TCP -- TCP Extensions for Transactions: Functional Specification," RFC-1644, USC/ISI, July 1994.
[40] T. Berners-Lee, L. Masinter and M. McCahill. "Uniform Resource Locators (URL)", RFC 1738, Dec. 1994.

Author's curriculum vitae:

MA (1993) and PhD (2000) from Peking University, and Associate Professor at the Computer Science Department, Peking University. Research interests include: Networking and the Web, Computing Models and Programming Languages, Language Processing and Information Theories. Currently he is responsible for a joint research project of PKU and IBM China Research Center, aiming at providing a customizable Chinese Web page Collection System. Before that, he had participated in multiple research projects as group member, technical principal, and team leader, including a joint project Large-Scale People's Daily Corpus Processing with Fujitsu China Research Center (1998-2000), a Chinese information extraction system of an NSFC project and a joint research project between IBM China Research Center and Peking University (2000-2001), a programming language C** as a Generic Programming extension of C++ together with a type constraint library SCL (1996-2001, part of an NSFC Young Scientist Project), and a new protocol called STTP for the transfer control of Web resources (currently supported by the Chinese 863 Hi-Tech Plans with he being the principle investigator).

(中文摘要)

结构化超文本传输协议(STTP)设计概要*

孙 斌
北京大学计算机系，北京 10871 (Email: bswen@pku.edu.cn)

摘要：本文介绍了 HTTP 的一种兼容扩充 STTP(结构化超文本传输协议)，它包括一个新增的资源传输控制消息集和一个用于描述网页结构信息的结构化超文本标记语言 STML。由此实现一种简单、高效的超文本资源的传输控制。初步测试表明其性能比 HTTP 可提高 70%～400%，并以相同量级减少网络包的数量，性能提升属目前所报道的最好水平。本文介绍了其组件的设计思想、STTP 服务器和客户端涉及的某些重要问题等。
关键词：万维网，HTTP，性能，超文本，资源传输

* 本研究得到 863 计划课题资助（课题编号 2001 AA 112081）。