H. Frystyk Nielsen, W3C
Mike Spreitzer, Xerox PARC
Bill Janssen, Xerox PARC
Jim Gettys, Compaq/W3C
draft-frystyk-httpng-overview-00.txt

Expires: May 17, 1999 Tuesday, November 17, 1998

HTTP-NG Overview
Problem Statement, Requirements, and Solution Outline

Status of this Document

Abstract

This document gives the authors' opinion of a rough outline of (1) the problems to be addressed by the proposed IETF HTTP-NG Working Group; (2) the requirements on the solution; and (3) the architecture of the solution. It draws heavily on contributions from the whole Protocol Design Group of the W3C HTTP-NG Activity. A suite of problems should be addressed, summarized as observing that the World Wide Web's tremendous success has created some strains on the Internet, its users, and its application developers. The requirements on the solution include modularity, extensibility, scalability, and efficiency. The proposed solution is to factor HTTP, the Web's central protocol, into three layers and look into performance improvements in the lower two of those resultant layers.

Table of Contents

1. Problem Statement

The World Wide Web is a tremendous and growing success and HTTP has been at the core of this success as the primary substrate for exchanging information on the Web. However, HTTP/1.1 [3] is becoming strained modularity wise as well as performance wise and those problems are to be addressed by HTTP-NG.

Modularity is an important kind of simplicity, and HTTP/1.x isn't very modular. If we look carefully at HTTP/1.x, we can see it addresses three layers of concerns, but in a way that does not cleanly separate those layers: message transport, general-purpose remote method invocation, and a particular set of methods historically focused on document processing (broadly construed to include things like forms processing and searching).

The lack of modularity makes the specification and evolution of HTTP more difficult than necessary and also causes problems for other applications. Applications are being layered on top of HTTP, and these applications are thus forced to include a lot of HTTP's design --- whether this is technically ideal or not. Other general invocation systems (notably including originally Web independent ones like DCOM, Java RMI, and some CORBA implementations) are also being layered on top of HTTP. Furthermore, to avoid some of the problems associated with layering on top of HTTP, other applications start by cloning a subset of HTTP and layering on top of that. Some of the particular problems that arise from HTTP/1.x's lack of modularity include (but are not limited to) the following.

- o The fact that message delimiting in HTTP/1.1 is not done in a distinct layer but rather is mixed with other functionality leads to a very complex specification involving five different ways to delimit a message (see [3], section 4.4).
- o Because general applications are being tunneled through HTTP's document fetching and forms processing methods (GET and POST) --- and in a wide variety of ways --- it is very difficult for a firewall to discern the semantic content of a given interaction, which makes it very difficult for a firewall to apply security policies.

- o Tunneling other applications through document processing methods invites confusion (between the document processing application and the tunneled application(s)) on a number of fronts (see [5] for a discussion of complexities of using HTTP as a substrate that arise even when such layering is deemed appropriate within HTTP/1.x)
- o Tunneling other applications through HTTP's document processing application requires a degree of quoting/encoding that would not be necessary with a more modular HTTP.
- o Because HTTP's invocation layer design is intimately tied to its document processing application, designers of other applications have a non-trivial job in trying to figure out how to use the invocation layer for their own applications.

There are two sides to the performance strains to be addressed. One side is the load presented to the Internet (HTTP accounts for the largest fraction of traffic on the Internet). Making HTTP use Internet resources more efficiently would have a real benefit for everyone. The other side is the performance delivered to end users and applications, which is often low on the general Internet today. Furthermore, usage of wireless is anticipated to grow significantly in the near future, and many wireless technologies deliver relatively low bandwidth and high latency which makes delivering good performance to users and applications even more challenging.

2. Requirements

The continuous growth of the Web depends on the availability of a simple yet powerful mechanism for exchanging information on the Internet. The purpose of the work proposed here is to design the next generation of the HTTP protocol that fulfills a set of requirements and at the same time preserve a simplistic design: complex features should be built on a simple base.

Even though HTTP/1.1 overcomes many of the deficiencies in HTTP/1.0 [1], it does not change the overall nature of the protocol. The explicit design decision of keeping HTTP/1.1 backward compatible with HTTP/1.0 has prevented a real cleanup of its architecture. By loosening this constraint, HTTP-NG can address the following requirements.

The requirements listed below are specific points put forward where improvements are needed (see also [11]). These are in addition to the general design principles laid out in [2]. We hope that the list will spur further discussion on the working group mailing list about inconsistencies or omissions (see [12]).

2.1 Simplicity at the Core

Simplicity is a key element in systems that are easy to understand, implement, and maintain. Over time, HTTP has lost a significant part of its initial simplicity by combining a number of different concerns at different levels into a single large protocol. The result is a protocol that is difficult to understand, implement, and modify in a robust and reliable manner; primarily because the line between extensions/applications and the core infrastructure has become blurry.

Modularity provides one form of simplicity with the potential of allowing the core infrastructure to remain small and well defined over time while applications can grow yet more complex on top. This is likely to help minimal implementations with limited capabilities to coexist with more capable implementations. Furthermore, applications themselves are likely to benefit from modularization as they are less prone to inadvertently stepping on each other.

By factoring the elements of HTTP into appropriate layers and modules, HTTP-NG must attempt to produce a simpler but more capable and more flexible system than the one currently provided by HTTP.

2.2 Distributed Extensibility

A wide range of applications have proposed various extensions to HTTP including distributed authoring, collaboration, printing, and remote procedure call mechanisms leading to a growing tension between dynamically extensible applications and public, static specifications. Due to the inherently unstructured extensibility model of HTTP/1.x, there is no guarantee that these extensions are dealt with as intended nor that they can applied to the same message without conflicting. The result is a lack of robustness in the current Web infrastructure which is unacceptable for many potential Web applications and which may lead to Web fragmentation and lack of interoperability.

The requirement to extensibility is that extended applications do not require agreement across the whole Internet; rather, it suffices:

- that conforming peers supporting a particular protocol extension or feature can employ it with no prior agreement;
- that it is possible for one party having a capability for a new protocol to require that the other party either understand and abide by the new protocol or abort the operation; and
- that negotiation of capabilities is possible.

Note that the requirements are on the core infrastructure and not on the extensions and their semantic interactions using this infrastructure. That is, it is not a requirement that interactions between extensions be defined, only interactions between extensions and the core infrastructure. The former is a much harder problem that we don't believe can be solved generically.

The HTTP Extension Framework proposes a mechanism for defining extensions in HTTP/1.x and keeping them separate from each other enabling better support for the type of evolution of features and applications seen in the Web. However, this mechanism can not change the behavior of existing HTTP features like the existing HTTP/1.1 caching model, for example, and so the applicability of the extension framework is limited.

By putting this kind of extensibility into the core of HTTP-NG, new features can be introduced and existing features can be replaced dynamically putting better evolvability into the heart of the architecture.

2.3 Global Scalability

HTTP needs to be effective and efficient when deployed in the full global system that includes all the clients, servers, caches, proxies, gateways, tunnels, and other intermediaries and their interactions over the global Internet and all the connected intranets. HTTP is the single protocol that now consumes most of the Internet bandwidth. Any replacement to HTTP/1.1 will have to address the foreseeable growth that one can expect in the near future. Measurements show [4] that scalability is not isolated to a single part in the Web model but affects all layers ranging from low level transport protocols to high-level user interfaces. Even though HTTP-NG focuses on the protocol related different encodings of the contents may have as big an impact as improving the underlying transport. An example is the potential savings using style sheets instead of inlined images for representing typographical effects in Web pages.

2.4 Network Efficiency

One can argue that bandwidth and latency of the Internet will improve dramatically over the next couple of years. However, wireless PDA’s, portable machines and satellite links will continue to impose severe practical limits on the available bandwidth, latency and on-line connectivity on parts of the Internet. We consider it likely that low bandwidths in the 9600-19200 bps range and latency in the >1/2 second range will be with us for a long time.

It is important to note that latency and bandwidth are independent variables; for example satellite IP systems exist today which provide good bandwidth to remote locations, but poor latency. Most users of the Web are today at home using a dial-up connection with a 28.8 kbps. On the optimistic side, this provides a minimum of 160 ms from the closest part of the Internet. Cellular modems and many wireless systems have even higher latency and lower bandwidth.

HTTP is a simple request/response protocol, not designed for the environment where it is now most heavily used. In [4], it is described how persistent connections and pipelining in HTTP/1.1 will solve some, but not all of these problems. The reason is that HTTP/1.1 is designed to limit TCP overhead produced by HTTP/1.0 but not protocol overhead due to HTTP itself. As an example, HTTP/1.1 defines 5 different mechanisms for finding the length of a message, of which all but closing the TCP connection require significant parsing to determine which one is used.

Automatable, machine-readable messages are different from human readable messages even though they may both be encoded using ASCII strings. The choice of MIME based header encoding in HTTP has led to the general misconception that HTTP is intended as a human readable protocol. The result has been verbose messages and extremely complicated parsers. As an example, a typical HTTP request is about 250 bytes long. Due to the nature of typical Web usage, subsequent requests are often closely related leading to about 90% in redundancy between requests. This means slowing down information exchange over low bandwidth connections.

If HTTP does not improve its performance dramatically on low bandwidth connections, it is likely that other more compact and lightweight protocols will be deployed with the risk of incompatibility between low bandwidth sensitive devices. HTTP-NG will attempt to optimize the bandwidth/latency usage of HTTP, at several levels.

2.5 Transport Flexibility

Although ensuring the stability of URIs to a high degree is a social engineering task, it is as important that the Web infrastructure supports evolution of transports. For example, a single resource may be available through different access protocols supported by the party serving the resource. These access protocols may or may not be compatible: HTTP/0.9, HTTP/1.0, and HTTP/1.1 are backwards compatible protocols but HTTP running on top of SSL is not although it is in fact using HTTP as part of the transport stack.

HTTP-NG should have an architected way of using any of an open, extensible set of transport substacks and should allow for transport stacks that do not necessarily include TCP. Furthermore, HTTP/NG's architecture should have a generalized notion of transport transformers, of which SSL and TLS are examples but not the only possibilities.

3. Solution Outline: The Three Layers of HTTP-NG

The solution outlined in this section describes the solution that has been developed as part of the W3C HTTP-NG Project using a prototype implementation [10]. It attempts to address the requirements laid out above by factoring HTTP into three layers and by looking into performance improvements in the lower two of those resultant layers. Here is a brief outline of the three layers into which HTTP is to be factored (see also [6]).

3.1 Message Transport

The lowest layer addresses concerns of simply transporting opaque messages for use by the middle (remote invocation) layer. This layer identifies a "message transport" abstraction, and the concept of "transformers" or "filters" that produce new message transports from other message transports. "Message transports" are built on top of existing -- and potentially future -- Internet "transport" protocols, such as TCP and UDP. HTTP-NG allows the use of a variety of message transport substacks or services; this provides a welcome flexibility for addressing current, future, and evolvability concerns at the message transport layer.

In particular, there is a set of services that have shown to be often needed in the Web and other applications:

- o Batching and pipelining of messages in order to save round trip times which especially on dialup lines and wireless connections are significant.
- o Chunking and multiplexing of messages which can help fast rendering of pages as well as faster responses from caches where already cached responses can be returned out-of-order without waiting for non-cached responses.
- o Efficient record marking for lowering parsing overhead of determining the message length.
- o Support for callback functions via endpoint identification for notifications etc. needed by an important set of applications.

Although these services are distinct services, they are related in such a way that we propose to combine them in a single filter called WebMux [9]. WebMux is not to be considered an independent transport, it is likely to provide the services listed above by taking advantage of the services provided by lower level transports like TCP, for example, much like HTTP/1.x provides a set of transport services in combination with TCP.

While WebMux potentially can work with other transports, a particular important combination is WebMux running on top of TCP. We are still in the evaluation phase of the interactions between WebMux and TCP with respect to handling support for announcing buffer capabilities. HTTP/1.x clients are currently expected to provide essentially unlimited buffering which especially in certain HTTP/1.0 and HTTP/1.1 proxy interactions can cause clients to fail in unpredictable ways leading to unreliable situations and denial of service attacks. We intend to discuss this further as part of the HTTP-NG working group.

3.2 Remote Invocation

The middle layer is a generic request/response messaging layer where clients make use of services exported from a server by invoking operations on resources resident in that server. It provides a mechanism for remote invocations suitable for use by the Web application and also by other applications that are currently being layered on top of HTTP (whether directly or indirectly via other layered remote invocation systems).

The layer does not contain any application-specific services like security or caching, or other application layer functionality. It assumes a hop-by-hop operation where proxies are supported at the higher-level application layer and uses the set of services provided by the message transport layer.

Suitability for use by the Web application means, among other things that

- o it be byte efficient which not only is important on dialup lines and wireless connections but also for operations like cache validation which are likely to become widespread used.
- o it be easy to parse which especially is important for servers and proxies
- o it supports the type of distributed extensibility currently seen in the Web

By designing an invocation protocol that serves this breadth of applications, particularly including the ones that use other remote method invocation systems, we hope to eventually reduce the number of invocation protocols in use.

It is not a goal to support every single one of the features of CORBA, DCOM, or Java RMI in their current forms, but rather -- by being "good enough" for those systems' actual applications -- to be a force for unification. It is not a viable solution to simply adopt CORBA, DCOM, or Java RMI for this layer, because each -- in its current form -- has technical and/or political liabilities for Web use. The hope is that HTTP-NG's invocation protocol is eventually adopted by those other systems.

The HTTP-NG work should include defining a network interface definition language (the highest layer will need to be expressed in some language). It is recognized that there could be multiple interface languages for use with a given invocation protocol. In particular, it is possible that a XML based solution and/or future versions of existing interface definition languages will be suitable here. For this reason and for general modularity and clarity, the subject matter of the invocation protocol --- objects, other data, and invocations --- is defined in a language-independent way (see [8]).

3.3 The Web Application

With the lower two layers of HTTP-NG in place, it remains to express the operations and application layer services of HTTP/1.1 including the current set of methods (GET, HEAD, PUTà), content negotiation, caching, access authentication, etc. as particular network interfaces using those lower two layers.

The application layer is application-specific meaning that the particular set of network interfaces defining an application varies from application to application. For example, the Web application defined by HTTP/1.1 would constitute a different application than the WebDAV application (though they might share some common part).

The HTTP-NG architecture allows multiple applications to co-exist at this level, and provides a mechanism for adding new applications without stepping on existing applications. Furthermore, applications are defined both statically, in terms of the type system at the second layer, and dynamically, in terms of the transport elements of the first layer allowing for protocol evolution to happen independent of interface evolution.

While it is tempting to try to improve on the existing Web application functionality, that is not the main focus of the HTTP-NG work. HTTP-NG is essentially about transitioning the existing functionality to a better technology base and as every difference from the Web application defined by HTTP/1.x places a strain on the transition, these are to be minimized. However, "the existing functionality" is itself a moving target and so HTTP-NG must track that closely to make HTTP-NG a viable replacement.

4. Security Considerations

The division of responsibility for security services should be as follows. The lower two layers are responsible for providing some subset of the authentication, message integrity, and message secrecy security services; applications provide whatever other security services they need (e.g., authorization, auditing, accounting, further authentication) based on the services provided by the lower layers. Which services are supplied by the lower two layers, and which mechanisms are used to supply them, is a function of the choice of message transport and invocation protocol(s) used. To support this variety of possibilities, the message transport and invocation abstractions must use an open, extensible categorization of security principals and credentials.

HTTP-NG must interact with firewalls at least as well as HTTP. This includes

1. the ability of a firewall and its operator to manage traffic through the firewall; and
2. the ability of users and applications to get through firewalls that don't want to block them.

On the first issue, HTTP-NG's very nature leads to an improvement over the current Web situation: by replacing a wide variety of ways of tunneling other applications through HTTP/1.x with a defined way of basing applications on a standard invocation mechanism, HTTP-NG traffic will be more comprehensible (and thus more manageable) to firewalls. One the second issue, the existing Web architecture already does a decent job in one direction (client behind firewall to server outside), and HTTP-NG should do at least as well as the current Web for the other direction (client outside firewall to server inside).

5. Deployment and Transition Strategies

The purpose of HTTP-NG is not only to allow for new applications to be developed and deployed in the Web but also to provide an incentive for moving existing applications already seen in the Web onto the HTTP-NG substrate. First and foremost, this of course includes the services provided by the HTTP/1.1 protocol itself like content negotiation and caching which are integral parts of the current Web application. However, HTTP/1.x is in fact only the tip of the iceberg. HTTP is being extended or augmented locally in intranets as well as globally on the Internet at large either directly or indirectly via other layered remote invocation systems. The myriad of applications is forcing extensions to HTTP/1.x and threatens the interoperability of the Web. The lack of an interface signature at the invocation layer makes security policy very difficult to enforce, and inhibits the deployment of automated services in the Web.

In order for this situation to change, the incentive has to be strong enough for application providers and extension designers to actually deploy the HTTP-NG substrate for new as well as existing applications. As the investments in the current infrastructure is already extremely high, this is likely to be a process that will continue for a very long time and potentially have very slow start phase.

There are at least two different types of transitions that have to evaluated and tested:

- The transition from the current HTTP/1.x infrastructure to one based on HTTP-NG
- The transition of current applications based on the HTTP/1.x infrastructure to the HTTP-NG infrastructure

Support for transition at the application layer can be considered to be a special case of the larger question of support for evolvability, which is one of the primary design goals for HTTP-NG. Therefore, the claim for evolvability can to a certain extent be evaluated by HTTP-NG's capability of supporting applications in transition from the existing infrastructure to HTTP-NG.

Support for transition at the infrastructure level is fundamentally different as this relies on interfacing features in the existing infrastructure and the NG substrate. Some techniques that should be considered for handling this task including but not limited to

- o the HTTP/1.1 Upgrade header field
- o the HTTP Extension Framework
- o protocol-conversion gateways
- o directory services for locating HTTP-NG servers
- o DHCP for locating HTTP-NG servers
- o new DNS record(s) to indicate availability of NG at a given server
- o new URI scheme(s)

No single technique is likely to provide the full answer, but some combination of these and other techniques may well be sufficient to an overall reliable transition between the two infrastructures.

6. References

[1] T. Berners-Lee, R. Fielding, H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, W3C/MIT, UC Irvine, W3C/MIT, May 1996

[2] B. Carpenter, "Architectural Principles of the Internet", RFC 1958, June 1996

[3] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, U.C. Irvine, DEC W3C/MIT, DEC, W3C/MIT, W3C/MIT, January 1997

[4] H.F.Nielsen, J. Gettys, A. Baird-Smith, E. PrudÆhommeaux, H. Lie, and C. Lilley. "Network Performance Effects of HTTP/1.1, CSS1, and PNG", Proceedings of ACM SIGCOMM '97, Cannes France, September 1997

[5] K. Moore, P. Faltstrom, "On the use of HTTP as a Substrate for Other Protocols", Internet Draft, August 1998, draft-iesg-using-http-00.txt. This is work in progress.

[6] B. Janssen, H.F.Nielsen, M.Spreitzer, "HTTP-ng Architectural Model", August 1998, draft-frystyk-httpng-arch-00.txt. This is work in progress.

[7] D. Larner, "HTTP-ng Web Interfaces" (limited prototype used in testbed), Internet Draft, August 1998. draft-larner-nginterfaces-00.txt. This is work in progress.

[8] B. Janssen, "HTTP-ng Binary Wire Protocol", Internet Draft, August 1998, draft-janssen-httpng-wire-00.txt. This is work in progress.

[9] J. Gettys, H.F.Nielsen, "The WebMux Protocol", Internet Draft, August 1998, draft-gettys-webmux-00.txt. This is work in progress.

[10] D.Veillard, "Design of HTTP-ng Testbed", W3C Note, 10 July 1998

[11] M.Spreitzer, H.F.Nielsen, "Short- and Long-Term Goals for the HTTP-NG Project", W3C Note, 27 March 1998

[12] Minutes from HTTP-NG BOF at the IETF Chicago Meeting, August 24, "http://www.w3.org/Protocols/HTTP-NG/1998/08/HTTP-NG-BOF-Minutes.html"

7. Acknowledgements

This work draws heavily on the work of the whole Protocol Design Group of the W3C HTTP-NG Activity notably including contributions from Paul Bennett, Dan Larner, and Paula Newman. Larry Masinter also made valuable contributions.

8. Authors

Henrik Frystyk Nielsen
World Wide Web Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA
Email: frystyk@w3.org

Mike Spreitzer
Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304, USA
Email: spreitze@parc.xerox.com

Bill Janssen
Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304, USA
Email: janssen@parc.xerox.com

James Gettys
World Wide Web Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA
Email: jg@w3.org