Network Working Group   Keith Moore
Internet-Draft   University of Tennessee
5 August 1998   Patrik Faltstrom
Expires: 5 February 1999   Tele2

On the use of HTTP as a Substrate for Other Protocols

draft-iesg-using-http-00.txt

Abstract

Recently there has been widespread interest in using Hypertext Transport Protocol (HTTP) as a substrate for other applications-level protocols. This document relates current IESG and IAB thinking on technical particulars of such use, including use of default ports, URL schemes, and HTTP security mechanisms. This thinking is subject to change as discussion continues and more experience is gained with such use.

1. Introduction

Recently there has been widespread interest in using Hypertext Transport Protocol (HTTP) [1] as a substrate for other applications-level protocols. Various reasons cited for this interest have included:

o   familiarity and mindshare,

o   compatibility with widely deployed browsers,

o   ability to reuse existing servers and client libraries,

- ease of prototyping servers using CGI scripts and similar extension mechanisms,

- ability to use existing security mechanisms such as HTTP digest authentication [2] and SSL or TLS [3],

- the ability of HTTP to traverse firewalls, and

- cases where a server often needs to support HTTP anyway.

The Internet community has a long tradition of protocol reuse, dating back to the use of Telnet [4] as a substrate for FTP [5] and SMTP [6]. However, the recent interest in layering new protocols over HTTP has raised a number of questions when such use is appropriate, and the proper way to use HTTP in contexts where it is appropriate. Specifically, for a given application that is layered on top of HTTP:

- Should the application use a different port than the HTTP default of 80?

- Should the application use traditional HTTP methods (GET, POST, etc.) or should it define new methods?

- Should the application use http: URLs or define its own prefix?

- Should the application define its own MIME-types, or use something that already exists (like registering a new type of MIME-directory structure)?

This memo attempts to illustrate the current thinking of the Applications Area Directors and other members of IESG and IAB, on these questions. The answers to some of these questions may change over time as discussions on these issues continue, or as the community acquires additional experience.

We also expect that these recommendations may eventually be superseded by a working group chartered to establish mechanisms for layering new protocols over a subset of HTTP.

2. Issues Regarding the Design Choice to use HTTP

Despite the advantages listed above, it's worth asking the question as to whether HTTP should be used at all, or whether the entire HTTP protocol should be used.

2.1 Complexity

HTTP started out as a simple protocol, but quickly became much more complex due to the addition of several features unanticipated by its original design. These features include persistent connections, byte ranges, content negotiation, and cache support. All of these are useful for traditional web applications but may not be useful for the layered application. The need to support (or circumvent) these features can add additional complexity to the design and implementation of a protocol layered on top of HTTP. Even when HTTP can be "profiled" to minimize implementation overhead, the effort of specifying such a profile might be more than the effort of specifying a purpose-built protocol which is better suited to the task at hand. Even if existing HTTP client and server code can often be re-used, the additional complexity of HTTP over a purpose-built protocol can increase the number of interoperability problems.

2.2 Overhead

Further, although HTTP can be used as the transport for a "remote procedure call" paradigm, HTTP's protocol overhead, along with the connection setup overhead of TCP, can make HTTP a poor choice. A protocol based on UDP, or with both UDP and TCP variants, should be considered if the payloads are very likely to be small (less than a few hundred bytes) for the foreseeable future. This is especially true if the protocol might be heavily used.

On the other hand, the connection setup overhead can become negligible if the layered protocol can utilize HTTP/1.1's persistent connections, and if the same client and server are likely to perform several transactions during the time the HTTP connection is open.

2.3 Security

Although HTTP appears at first glance to be one of the few "mature" Internet protocols that can provide good security, there are many applications for which neither HTTP's digest authentication nor TLS are sufficient by themselves.

Digest authentication requires a secret (e.g. a password) to be shared between client and server. This further requires that each client know the secret to be used with each server, but it does not provide any means of securely transmitting such secrets between the parties. Shared secrets can work fine for small groups where everyone is physically co-located; they don't work as well for large or dispersed communities of users. Further, if the server is compromised a large number of secrets may be exposed, which is especially dangerous if the same secret (or password) is used for several applications.

TLS is descended from SSL, which was originally designed to authenticate servers to clients - not the other way around. Even though TLS now provides mutual authentication, a client that needs to talk to multiple servers must still know which credentials to present to each server before establishing a secure connection to the server. Client and server must each use private keys that are trusted by the other party - typically because they are signed by a certificate authority (CA) known to the other. As in the digest authentication case, both client and server need ways to protect their private keys against exposure.

Web browsers typically are shipped with the public keys of several CAs "wired in" so that they can verify the identity of any server whose public key was signed by one of those CAs. This deployment model does not necessarily work well for other applications, and it doesn't provide any way for a server to verify a client's identity. Even if the client's CA is recognized by the server, this doesn't necessarily convey authorization to use the service. Existing clients and servers may therefore lack the mechanisms needed for robust authentication using TLS or SSL and HTTP.

For any application that requires privacy, the 40-bit encryption provided by "US exportable" SSL implementations is unsuitable. Even 56-bit DES encryption, which is required by TLS, has been broken in a matter of days with a modest investment.

None of the above should be taken to mean that digest authentication or TLS are generally unsuitable for use in other applications - only that they are not a "magic pixie dust" solution to either authentication or privacy. An application's designers should carefully determine the application's users' requirements for authentication and privacy before automatically choosing TLS or digest authentication.

Note also that TLS can be used with other TCP-based protocols, and there are SASL [7] mechanisms similar to HTTP's digest authentication. So even if TLS and/or digest are suitable for an application, this does not imply that HTTP should be used.

2.4 Compatibility with Proxies and Firewalls

One oft-cited reason for the use of HTTP is its ability to pass through proxies or firewalls. Firewalls are an unfortunate consequence of the Internet's explosive growth, in that they decrease the deployability of new Internet applications, by requiring explicit permission (or even a software upgrade) to accommodate each new protocol.

However, the IESG takes the view that if a site's firewall prevents the use of unknown protocols, this is a conscious policy decision on the part of the firewall administrator. While it is arguable whether or not new protocols should be "firewall-friendly", they should definitely not be "firewall-hostile". In particular, new protocols should not attempt to circumvent a site's security policy.

We hope to eventually establish guidelines for "firewall-friendly" protocols, to make it easier for existing firewalls to be compatible with new protocols.

2.5 Questions to be asked when considering use of HTTP

o When considering payload size and traffic patterns, is HTTP an appropriate transport for the anticipated use of this protocol?

o Is this new protocol usable by existing web browsers without modification?

o Are the existing HTTP security mechanisms appropriate for the new application?

o Does the server for this application need to support HTTP anyway?

3. Issues Regarding Reuse of Port 80

IANA has reserved TCP port number 80 for use by HTTP. IESG will not allow a substantially new service, even one which uses HTTP as a substrate, to usurp port 80 from its traditional use. IESG is likely to consider a new use of HTTP a "substantially new service", and thus requiring a new port, if any of the following are true:

o The "new service" and traditional HTTP service are likely to reference different sets of data, even when they both operate on the same host.

o There is a good reason for the "new service" to be implemented by a separate server process, or separate code, than traditional HTTP service on the same host, at least on some platforms.

o There is a good reason to want to easily distinguish the traffic of the "new service" from traditional HTTP, e.g. for the purposes of firewall access control or traffic analysis.

o If none of the above are true, IESG is likely to consider the new use of HTTP an "extension" to traditional HTTP, rather than a "new service". Extensions to HTTP which share data with traditional HTTP services should probably define new HTTP methods to describe

those extensions, rather than using separate ports. If separate ports are used, there is no way for a client to know whether they are separate services or different ways of accessing the same underlying service.

4. Issues Regarding Reuse of the http: Scheme in URLs

A number of different URL schemes are in widespread use and many more are in the process of being standardized. In practice, the URL scheme not only serves as a "tag" to govern the interpretation of the remaining portion of the URL, it also provides coarse identification of the "type" of resource or service. This is used, for instance, by web browsers that provide a different response when a user mouse-clicks on an "http" URL, than when the user clicks on a "mailto" URL.

It is ultimately IESG's responsibility to determine whether a resource accessed by a protocol that is layered on top of HTTP, should use http: or some other URL prefix. Among the criteria that IESG is likely to consider in making this determination, are:

o Whether this URL is likely to become widely used, versus used only in limited communities or by private agreement.

o Whether a new "default port" is needed. A new "default port" requires a new URL type. Explicit port numbers in URLs are regarded as an "escape hatch", not something for use in ordinary circumstances.

o Whether use of the new service is likely to require a substantially different setup or protocol interaction with the server, than ordinary HTTP service. This could include the need to request a different type of service, or to reserve bandwidth, or to present different TLS authentication credentials to the server, or any number of other needs.

o Whether user interfaces (such as web browsers) are likely to be able to exploit the difference in the URL prefix to produce a significant improvement in usability.

Note that the convention of appending an "s" to the URL scheme to mean "use TLS or SSL" (as in "http:" vs "https:") is nonstandard and should not be propagated. For most applications, a single "use TLS or SSL" bit is not sufficient to adequately convey the information that a client needs to authenticate itself to a server, even if it has the proper credentials. Authentication or other connection setup information should be communicated in URL parameters, rather than in the URL prefix.

5. Issues regarding use of MIME media types

Since HTTP uses the MIME media type system [8] to label its payload, many applications which layer on HTTP will need to define, or select, MIME media types for use by that application. Especially when using a multipart structure, the choice of media types requires careful consideration. In particular:

- Should some existing framework be used, such as text/directory [9], or XML [10,11], or should the new content-types be built from scratch? Just as with HTTP, it's useful if code can be reused, but protocol designers should not be over-eager to incorporate a general but complex framework into a new protocol. Experience with ASN.1, for example, suggests that the advantage of using a general framework may not be worth the cost.

- If it is at all useful to be able to use the same payload over email, the differences between HTTP encoding of the payload and email encoding of the payload should be minimized. Ideally, there should be no differences in the "canonical form" used in the two environments. Text/* media types can be problematic in this regard because MIME email requires CRLF for line endings of text/* body parts, where HTTP traditionally uses LF only.

- Different "commands" or "operations" on the same kind of object can be communicated in a number of different ways, including different HTTP methods, different Content-Types, different Content-Type parameters, the Content-Disposition field, or inside the payload. Different protocols have solved this problem in different ways. Again, if it's desirable to provide the same services over electronic mail, the means of communicating the operation should ideally be the same in both environments.

6. Issues Regarding Existing vs. New HTTP Methods

It has been suggested that a new service layered on top of HTTP should define one or more new HTTP methods, rather than allocating a new port. This may be useful, but is not sufficient in all cases. The definition of one or more new methods for use in a new protocol, does not by itself alleviate the need for use of a new port, or a new URL type.

7. Issues regarding reuse of HTTP client, server, and proxy code

As mentioned earlier, one of the prime reasons for the use of HTTP as a substrate for new protocols, is to allow reuse of existing HTTP client, server, or proxy code. However, HTTP was not designed for such layering. Existing HTTP client and code may have "http" assumptions

wired into them. For instance, client libraries and proxies may expect "http:" URLs, and clients and servers may send (and expect) "HTTP/1.1", in requests and responses, as opposed to the name of the layered protocol and its version number.

Existing client libraries may not understand new URL types. In order to get a new HTTP-layered application client to work with an existing client library, the application may need to convert its URLs to an "http equivalent" form. For instance, if service "xyz" is layered on top of HTTP using port ###, the xyz client may need to translate URLs of the form "xyz://host/something" to "http://host:###/something" for the purpose of calling the existing HTTP client library. This should be done ONLY when calling the HTTP client library - such URLs should not be used in other parts of the protocol, nor should they be exposed to users.

Note that when a client is sending requests directly to an origin server, the URL prefix ("http:") is not normally sent. So translating xyz: URLs to http: URLs when calling the client library should not actually cause http: URLs to be sent over the wire. But when the same client is sending requests to a proxy server, the client will normally send the entire URL (including the http: prefix) in those requests. The proxy will remove the URL prefix when the request is communicated to the origin server.

Existing clients and servers will transmit "HTTP/1.1" (or a different version) in requests and responses. To facilitate reuse of existing code, protocols layered on top of HTTP must therefore transmit and accept "HTTP/1.1" rather than their own protocol name and version number. This may change in the future if client libraries and servers gain more flexibility.

For certain applications it may be necessary to require or limit use of certain HTTP features, for example, to defeat caching of responses by proxies. Each protocol layered on HTTP must therefore specify the specific way that HTTP will be used, and in particular, how the client and server should interact with HTTP proxies.

HTTP's three-digit status codes were designed for use with traditional HTTP applications, and may not be suitable to communicate the specifics of errors encountered in other applications. HTTP status codes should therefore not be used to indicate subtle errors of layered applications. They should be re-used only to indicate errors with, or the status of, the HTTP protocol layer; or to indicate the inability of the HTTP server to communicate with the application server.

8. Summary of IESG Policy regarding reuse of HTTP

1. **All standards-track protocols must provide adequate security.** The security needs of a particular application will vary widely depending on the application and its anticipated use environment. Merely using HTTP and/or TLS as a substrate for a protocol does not automatically provide adequate security for all environments, nor does it relieve the protocol developers of the need to analyze security considerations.

2. **New standards-track protocols - including those using HTTP** - must not attempt to circumvent users' firewall policies, particularly by masquerading as existing protocols. "Substantially new services" will not be allowed to re-use existing ports.

3. **New services should use new URL types.**

4. **Each** new protocol specification that uses HTTP as a substrate must describe the specific way that HTTP is to be used by that protocol, including how the client and server interact with proxies.

5. **New** services should define their own error reporting mechanisms, and use HTTP status codes only for communicating the state of the HTTP protocol.

9. Security Considerations

Much of this document is about security. Section 2.3 discusses whether HTTP security is adequate for the needs of a particular application, section 2.4 discusses interactions between new HTTP-based protocols and firewalls, section 3 discusses use of separate ports so that firewalls are not circumvented, and section 4 discusses the inadequacy of the "s" suffix of a URL prefix for specifying security levels.

10. Authors' addresses

Keith Moore
University of Tennessee, Knoxville
104 Ayres Hall
Knoxville TN, 37996-1301
USA
email: moore@cs.utk.edu

Patrik Faltstrom
Borgarfjordsgatan 16
BOX 64
162 92 Kista
Sweden
email: paf@swip.net

11. References

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1. RFC 2068, January 1997.

[2] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, L. Stewart. An Extension to HTTP: Digest Access Authentication. RFC 2069, January 1997.

[3] C. Allen, T. Dierks. The TLS Protocol Version 1.0. Internet-Draft draft-ietf-tls-protocol-05.txt (work in progress). November 1997.

[4] J. Postel, J.K. Reynolds. Telnet Protocol Specification. RFC 854, May 1983.

[5] J. Postel, J.K. Reynolds. File Transfer Protocol. RFC 959, October 1985.

[6] J. Postel. Simple Mail Transfer Protocol. RFC 821, August 1982.

[7] J. Myers. Simple Authentication and Security Layer (SASL). RFC 2222, October 1997.

[8] N. Freed, N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046, November 1996.

[9] T. Howes, M. Smith, F. Dawson Jr. A MIME Content-Type for Directory Information. Internet-Draft <draft-ietf-asid-mime-direct-08.txt>, July 1998. (work in progress)

[10] T. Bray, J. Paoli, C. M. Sperberg-McQueen. "Extensible Markup Language (XML)". World Wide Web Consortium Recommendation REC-

xml-19980210. http://www.w3.org/TR/1998/REC-xml-19980210.

[11] E. Whitehead, M. Murata. XML Media Types. RFC 2376, July 1998.