Network Working Group
Request for Comments: 3205
BCP: 56
Category: Best Current Practice

K. Moore
University of Tennessee

February 2002

On the use of HTTP as a Substrate

Status of this Memo

Copyright Notice

Abstract

Recently there has been widespread interest in using Hypertext Transfer Protocol (HTTP) as a substrate for other applications-level protocols. This document recommends technical particulars of such use, including use of default ports, URL schemes, and HTTP security mechanisms.

1. Introduction

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

- familiarity and mindshare,
- compatibility with widely deployed browsers,
- 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 recommends certain design decisions in answer to these questions.

This memo is intended as advice and recommendation for protocol designers, working groups, implementors, and IESG, rather than as a strict set of rules which must be adhered to in all cases. Accordingly, the capitalized key words defined in RFC 2119, which are intended to indicate conformance to a specification, are not used in this memo.

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 layering something over HTTP vs. using 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, or if it might be used over slow or expensive links.

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. (Similar concerns exist with TLS based clients or servers - if a private key is compromised then the attacker can impersonate the party whose key it has.)

TLS and its predecessor SSL were originally designed to authenticate web servers to clients, so that a user could be assured (for example) that his credit card number was not being sent to an imposter. However, many applications need to authenticate clients to servers, or to provide mutual authentication of client and server. TLS does

have a capability to provide authentication in each direction, but such authentication may or may not be suitable for a particular application.

Web browsers which support TLS or SSL are typically shipped with the public keys of several certificate authorities (CAs) "wired in" so that they can verify the identity of any server whose public key was signed by one of those CAs. For this to work well, every secure web server's public key has to be signed by one of the CAs whose keys are wired into popular browsers. This deployment model works when there are a (relatively) small number of servers whose identities can be verified, and their public keys signed, by the small number of CAs whose keys are included in a small number of different browsers.

This scheme does not work as well to authenticate millions of potential clients to servers. It would take a much larger number of CAs to do the job, each of which would need to be widely trusted by servers. Those CAs would also have a more difficult time verifying the identities of (large numbers of) ordinary users than they do in verifying the identities of (a smaller number of) commercial and other enterprises that need to run secure web servers.

Also, in a situation where there were a large number of clients authenticating with TLS, it seems unlikely that there would be a set of CAs whose keys were trusted by every server. A client that potentially needed to authenticate to multiple servers would therefore need to be configured as to which key to use with which server when attempting to establish a secure connection to the server.

For the reasons stated above, client authentication is rarely used with TLS. A common technique is to use TLS to authenticate the server to the client and to establish a private channel, and for the client to authenticate to the server using some other means - for example, a username and password using HTTP basic or digest authentication.

For any application that requires privacy, the 40-bit ciphersuites provided by some SSL implementations (to conform to outdated US export regulations or to regulations on the use or export of cryptography in other countries) are unsuitable. Even 56-bit DES encryption, which is required of conforming TLS implementations, has been broken in a matter of days with a modest investment in resources. So if TLS is chosen it may be necessary to discourage use of small key lengths, or of weak ciphersuites, in order to provide adequate privacy assurance. If TLS is used to provide privacy for passwords sent by clients then it is especially important to support longer keys.

None of the above should be taken to mean that either digest authentication or TLS are generally inferior to other authentication systems, or that they are unsuitable for use in other applications besides HTTP. Many of the limitations of TLS and digest authentication also apply to other authentication and privacy systems. The point here is that neither TLS nor digest authentication is a "magic pixie dust" solution to authentication or privacy. In every case, an application's designers must carefully determine the application's users' requirements for authentication and privacy before choosing an authentication or privacy mechanism.

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 it is not necessary to use HTTP in order to benefit from either TLS or digest-like authentication. However, HTTP APIs may already support TLS and/or digest.

2.4 Compatibility with Proxies, Firewalls, and NATs

One oft-cited reason for the use of HTTP is its ability to pass through proxies, firewalls, or network address translators (NATs). One unfortunate consequence of firewalls and NATs is that they make it harder to deploy new Internet applications, by requiring explicit permission (or even a software upgrade of the firewall or NAT) to accommodate each new protocol. The existence of firewalls and NATs creates a strong incentive for protocol designers to layer new applications on top of existing protocols, including HTTP.

However, if a site's firewall prevents the use of unknown protocols, this is presumably a conscious policy decision on the part of the firewall administrator. While it is arguable that such policies are of limited value in enhancing security, this is beside the point - well-known port numbers are quite useful for a variety of purposes, and the overloading of port numbers erodes this utility. Attempting to circumvent a site's security policy is not an acceptable justification for doing so.

It would be useful to 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

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

(In other words: will the payload size be worth the overhead associated with TCP and HTTP? Or will the application be able to make use of HTTP persistent connections to amortize the cost of that overhead over several requests?)

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

  (For example: Is the request transmitted as if it were a filled-in HTML form? Is the response which is returned viewable from a web browser, say as HTML?)

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

- Are HTTP status codes and the HTTP status code paradigm suitable for this application? (see section 8)

- 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. It would not be appropriate for a substantially new service, even one which uses HTTP as a substrate, to usurp port 80 from its traditional use. A new use of HTTP might be considered a "substantially new service", thus requiring a new port, if any of the following are true:

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

- 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.

- 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.

- If none of the above are true, it is arguable that the new use of HTTP is 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 kind of resource or service which is being accessed. For example, web browsers typically provide a different response when a user mouse-clicks on an "http" URL, than when the user clicks on a "mailto" URL.

Some criteria that might be used in making this determination are:

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

- Whether a new "default port" is needed. If reuse of port 80 is not appropriate (see above), a new "default port" is needed. A new default port in turn requires that a new URL scheme be registered if that URL scheme is expected to be widely used. Explicit port numbers in URLs are regarded as an "escape hatch", not something for use in ordinary circumstances.

- 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 from the network, or to reserve bandwidth, or to present different TLS authentication credentials to the server, or different kind of server provisioning, or any number of other needs.

- 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.

According to the rules in [8] the "http:" URI is part of the "IETF Tree" for URL scheme names, and IETF is the maintainer of the "IETF Tree". Since IESG is the decision-making body for IETF, IESG has the authority to determine whether a resource accessed by a protocol that is layered on top of HTTP, should use http: or some other URL prefix.

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 of limited value. 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. For instance, in order to ensure that adequate security is provided with TLS an application may need to be

configured with a list of acceptable ciphersuites, or with the client certificate to be used to authenticate to a particular server. When it is necessary to specify authentication or other connection setup information in a URL these 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 [9] 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 [10], or XML [11,12], 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.

- Should MIME multipart or message types be allowed? This can be an advantage if it is desirable to incorporate (for example) the multipart/alternative construct or the MIME security framework. On the other hand, these constructs were designed specifically for use in store-and-forward electronic mail systems, and other mechanisms may be more appropriate for the application being considered.

  The point here is that a decision to use MIME content-type names to describe protocol payloads (which is generally desirable if the same payloads may appear in other applications) does not imply that the application must accept arbitrary MIME content-types, including MIME multipart or security mechanisms. Nor does it imply that the application must use MIME syntax or that it must recognize or even tolerate existing MIME header fields.

- If the same payload is likely to be sent over electronic mail, 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.

- o A MIME content-type label describes the nature of the object being labeled. It does not describe, and should not be used to describe, the semantics which should be applied when the object is received. For instance, the transmission of an object with a particular content-type using HTTP POST, should not be taken as a request for some operation based solely on the type. The request should be separate from the content-type label and it should be explicit.

  When it is necessary for a protocol layered on HTTP to allow different operations on the same type of object, this can be communicated in a number of different ways: HTTP methods, HTTP request-URI, HTTP request headers, the MIME Content-Disposition header field, or as part of the payload.

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. The use of new methods may be appropriate, 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 primary 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, it may be necessary for the application 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, when invoking an HTTP client library, to translate its URLs from "xyz://host/something" format to "http://host:###/something" for the purpose of calling that 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 http: prefix when the request is communicated to the origin server.

Existing HTTP client libraries and servers will transmit "HTTP/1.1" (or a different version) in requests and responses. To facilitate reuse of such libraries and servers by a new protocol, such a protocol may therefore need to transmit and accept "HTTP/1.1" rather than its own protocol name and version number. Designers of protocols which are layered on top of HTTP should explicitly choose whether or not to accept "HTTP/1.1" in protocol exchanges.

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.

8. Issues regarding use of HTTP status codes

HTTP's three-digit status codes were designed for use with traditional HTTP applications (e.g., document retrieval, forms-based queries), and are unlikely to be suitable to communicate the specifics of errors encountered in dissimilar applications. Even when it seems like there is a close match between HTTP status codes and the codes needed by the application, experience with reuse of other protocols indicates that subtle variations in usage are likely; and that this is likely to degrade interoperability of both the original protocol (in this case HTTP) and any layered applications.

HTTP status codes therefore should not be used to indicate subtle errors of layered applications. At most, the "generic" HTTP codes 200 (for complete success) and 500 (for complete failure) should be used to indicate errors resulting from the content of the request message-body. Under certain circumstances, additional detail about the nature of the error can then be included in the response message-body. Other status codes than 200 or 500 should only appear if the error was detected by the HTTP server or by an intermediary.

A layered application should not define new HTTP status codes. The set of available status codes is small, conflicts in code assignment between different layered applications are likely, and they may be needed by future versions of, or extensions to, mainstream HTTP.

Use of HTTP's error codes is problematic when the layered application does not share same notion of success or failure as HTTP. The problem exists when the client does not connect directly to the origin server, but via one or more HTTP caches or proxies. (Since the ability of HTTP to communicate through intermediaries is often the primary motivation for reusing HTTP, the ability of the application to operate in the presence of such intermediaries is considered very important.) Such caches and proxies will interpret HTTP's error codes and may take additional action based on those codes. For instance, on receipt of a 200 error code from an origin server (and under other appropriate conditions) a proxy may cache the response and re-issue it in response to a similar request. Or a proxy may modify the result of a request which returns a 500 error code in order to add a "helpful" error message. Other response codes may produce other behaviors.

A few guidelines are therefore in order:

- A layered application should use appropriate HTTP error codes to report errors resulting from information in the HTTP request-line and header fields associated with the request. This request information is part of the HTTP protocol and errors which are associated with that information should therefore be reported using HTTP protocol mechanisms.

- A layered application for which all errors resulting from the message-body can be classified as either "complete success" or "complete failure" may use 200 and 500 for those conditions, respectively. However, the specification for such an application must define the mechanism which ensures that its successful (200) responses are not cached by intermediaries, or demonstrate that such caching will do no harm; and it must be able to operate even if the message-body of an error (500) response is not transmitted back to the client intact.

- A layered application may return a 200 response code for both successfully processed requests and errors (or other exceptional conditions) resulting from the request message-body (but not from the request headers). Such an application must return its error code as part of the response message body, and the specification for that application protocol must define the mechanism by which the application ensures that its responses are not cached by intermediaries. In this case a response other than 200 should be used only to indicate errors with, or the status of, the HTTP protocol layer (including the request headers), or to indicate the inability of the HTTP server to communicate with the application server.

- o A layered application which cannot operate in the presence of intermediaries or proxies that cache and/or alter error responses, should not use HTTP as a substrate.

9. Summary of recommendations regarding reuse of HTTP

1. All protocols should 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 for their particular application.

2. New protocols - including but not limited to those using HTTP - should not attempt to circumvent users' firewall policies, particularly by masquerading as existing protocols. "Substantially new services" should not reuse existing ports.

3. In general, new protocols or services should not reuse http: or other URL schemes.

4. Each new protocol specification that uses HTTP as a substrate should 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 follow the guidelines in section 8 regarding use of HTTP status codes.

10. 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.

11. References

[1] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[2] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[3] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[4] Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, May 1983.

[5] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[6] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

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

[8] Petke, R. and I. King, "Registration Procedures for URL Scheme Names", BCP 35, RFC 2717, November 1999.

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

[10] Howes, T., Smith, M. and F. Dawson, "A MIME Content-Type for Directory Information", RFC 2425, September 1998.

[11] Bray, T., Paoli, J. and C. Sperberg-McQueen, "Extensible Markup Language (XML)" World Wide Web Consortium Recommendation REC-xml-19980210, February 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[12] Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.

12. Author's Address

Keith Moore
University of Tennessee
Computer Science Department
1122 Volunteer Blvd, Suite 203
Knoxville TN, 37996-3450
USA

EMail: moore@cs.utk.edu

13. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.