Decomposing the Hypertext Transfer Protocol
draft-bishop-decomposing-http-00

Abstract

The Hypertext Transfer Protocol in its various versions combines concepts of both an application and transport-layer protocol. As this group contemplates employing alternate transport protocols underneath HTTP, this document attempts to delineate the boundaries between these functions to define a shared vocabulary in discussing the revision and/or replacement of one or more of these components.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 22, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
1. Introduction

The Hypertext Transfer Protocol defines a very flexible tool set enabling client applications to make requests of a server for content or action. This general protocol was conceived for "the web," interconnected pages of Hypertext Markup Language (HTML) and associated resources used to render the HTML, but has since been used as a general-purpose application transport. Server APIs are commonly exposed as REST APIs, accessed over HTTP.

HTTP/1.0 [RFC1945] was a text-based protocol which did not specify its underlying transport, but describes the mapping this way:

On the Internet, HTTP communication generally takes place over TCP/IP connections. The default port is TCP 80, but other ports can be used. This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used, and the mapping of the HTTP/1.0 request and response structures onto the transport data units of the protocol in question is outside the scope of this specification.
HTTP/1.1 [RFC7230] expands on the TCP binding, introducing connection management concepts into the HTTP layer.

HTTP/2 [RFC7540] replaced the simple text-based protocol with a binary framing. The changes are described in the following way:

HTTP/2 addresses these issues by defining an optimized mapping of HTTP’s semantics to an underlying connection. Specifically, it allows interleaving of request and response messages on the same connection and uses an efficient coding for HTTP header fields. It also allows prioritization of requests, letting more important requests complete more quickly, further improving performance.

The resulting protocol is more friendly to the network because fewer TCP connections can be used in comparison to HTTP/1.x. This means less competition with other flows and longer-lived connections, which in turn lead to better utilization of available network capacity.

Finally, HTTP/2 also enables more efficient processing of messages through use of binary message framing.

Conceptually, HTTP/2 achieved the same properties required of a TCP mapping using wildly different strategies from HTTP/1.1. HTTP/1.1 achieves properties such as parallelism and out-of-order delivery by the use of multiple TCP connections. HTTP/2 implements these services on top of TCP to enable the use of a single TCP connection. The working group’s charter to maintain HTTP’s broad applicability meant that there were few or no changes in how HTTP surfaces to applications.

Other efforts have mapped HTTP or a subset of it to various transport protocols besides TCP - HTTP can be implemented over SCTP [RFC4960] as in [I-D.natarajan-http-over-sctp], and useful profiles of HTTP have been mapped to UDP in various ways (HTTPU and HTTPUM in [goland-http-udp] and [UPnP], CoAP [RFC7252], QUIC [I-D.tsvwg-quic-protocol]).

With the publication of HTTP/2 over TCP, the working group is beginning to consider how a mapping to a non-TCP transport would function. This document aims to enable this conversation by describing the services required by the HTTP semantic layer. A mapping of HTTP to a transport other than TCP must define how these services are obtained, either from the new transport or by implementing them at the application layer.
2. The Semantic Layer

At the most fundamental level, the semantic layer of HTTP consists of a client’s ability to request some action of a server and be informed of the outcome of that action. HTTP defines a number of possible actions (methods) the client might request of the server, but permits the list of actions to be extended.

A client’s request consists of a desired action (HTTP method) and a resource on which that action is to be taken (path). The server responds with a status code which informs the client of the result of the request – the outcome of the action or the reason the action was not performed. Actions may or may not be idempotent or safe, and the results may or may not be cached by intermediaries; this is defined as part of the HTTP method.

Each message (request or response) has associated metadata, called "headers," which provide additional information about the operation. In a request this might include client identification, credentials authorizing the client to request the action, or preferences about how the client would prefer the server handle the action. In a response, this might include information about the resulting data, modifications to the cacheability of the response, details about how the server performed the action, or details of the reason the server declined to perform the action.

The headers are structured key-value pairs, with rules defining how keys which occur multiple times should be handled. Due to artifacts of existing usage, these rules vary from key to key. For similar legacy reasons, there is no uniform structure of the values across all keys. Keys are case-insensitive ASCII strings, while values are sequences of octets typically interpreted as ASCII. Many headers are defined by the HTTP RFCs, but the space is not constrained and is frequently extended with little or no notice.

Each message, whether request or response, also has an optional body. The presence and content of the body will vary based on the action requested.

3. Transport Services Required

The HTTP Semantic Layer depends on the availability of the following services:

- Separate metadata and payload
- Parallelism
- Partial delivery
- Flow control and throttling
- Reliable delivery
- In-order delivery
- Security

No transport over which HTTP can be mapped actually provides all of the services on which the HTTP Semantic Layer depends. In the following table, we can see multiple transports over which HTTP has been deployed and the services they do or do not offer.

<table>
<thead>
<tr>
<th></th>
<th>TCP</th>
<th>UDP</th>
<th>SCTP</th>
<th>QUIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallelism</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partial delivery</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Flow Control</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Reliable</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-order</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Secure</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Based on an initial review of this table, it would seem that UDP is the least-appropriate substrate for an HTTP mapping. However, what ultimately matters is the combined capability of the transport and the application-defined adaptation layer.

4. The Transport Adaptation Layer

In order to compensate for the services not provided by a given underlying transport, each mapping of HTTP onto a new transport must define an intermediate layer implementing the missing services in order to enable the mapping.

Some of these have been wholesale imports of other protocols which exist to provide such an adaptation layer (TLS [RFC5246]) while others have been entirely new protocol machinery constructed.
specifically to serve as an adaptation layer (HTTP/2 framing). Still others take the form of implementation-level meta-protocol behavior (simultaneous connections handled in parallel). Because the existence of this adaptation layer has not been explicitly defined in the past, a clean separation has not always been maintained between the adaptation layer and either the transport or the semantic layer.

Some adaptation layers are so complex and fully-featured that the transport layer plus the adaptation layer can be conceptually treated as a new transport. For example, QUIC was originally designed as a transport adaptation layer for HTTP over UDP, but is now being refactored into a general-purpose transport layer for multiple protocols. Such a refactoring will require separating the services QUIC provides that are general to all applications from the services which exist purely to enable a mapping of HTTP to QUIC.

4.1. Security

Integrity and confidentiality are valuable services for communication over the Internet, and HTTP is no exception. HTTP originally defined no additional integrity or confidentiality mechanisms for the TCP mapping, leaving the integrity and confidentiality levels to those provided by the network transport. These may be minimal (TCP checksums) or rich (IPsec) depending on the network environment.

For situations where the network does not provide integrity and confidentiality guarantees sufficient to the content, [RFC2818] defines the use of TLS as an additional component of the adaptation layer in HTTP/1.1. HTTP/2 directly defines how TLS may be used to provide these services as part of its adaptation layer.

TLS itself requires reliable, in-order delivery. When those services are provided by the adaptation layer itself rather than the underlying transport, the adaptation layer must either provide those services to TLS as well as HTTP (as in QUIC) or a variant of TLS which does not require those services must be substituted (DTLS [RFC6347], as used in CoAP).

4.2. Message Framing and Request Metadata

Any protocol defines how the semantics of the protocol are mapped onto the wire in a transport. Most transports are either bytestreams or message-based, meaning that higher-layer concepts must be laid out in a reasonable structure within the stream or message. Each HTTP request and response contains metadata about the message (headers) and an optional body.
4.2.1. HTTP/1.x and Text-Based Headers

HTTP/1.x projects a message as an octet sequence which typically resembles a block of ASCII text. Specific octets are used to delimit the boundaries between message components. Within the portion of the message dedicated to headers, the key-value pairs are expressed as text, with the ‘:’ character and whitespace separating the key from the value.

Because this region appears to be text, many text conventions have accidentally crept into HTTP/1.x message parsers and even protocol conventions (line-folding, CRLF differences between operating systems, etc.). This is a source of bugs, such as line-folding characters which appear in an HTTP/2 message despite HTTP/2 not using a text-based header framing.

4.2.2. HTTP/2 and HPACK

HTTP/2 projects the requested action into the set of headers, then uses separate HEADERS and DATA frames to delimit the boundary between metadata and message body. These frames are used to provide message-like behaviors and parallelism over a single TCP bytestream.

Because the text-based transfer of repetitive headers represented a major inefficiency in HTTP/1.1, HTTP/2 also introduced HPACK [RFC7541], a custom compression scheme which operates on key-value pairs rather than text blocks. HTTP/2 frame types which transport headers always carry HPACK header block fragments rather than an uncompressed key-value dictionary.

4.3. Parallelism and Throttling

Because a client will often need each server to perform multiple actions at once, HTTP requires the ability to deliver requests in parallel and allow the server to respond to each request as the actions complete. In order to avoid overwhelming either the transport or the server, HTTP also requires a mechanism to limit the number of simultaneous requests a client may have outstanding.

4.3.1. HTTP/1.x and Multiple Connections

HTTP/1.0 used a very simple multi-request model - each request was made on a separate TCP connection, and all requests were handled independently. This had the drawback that TCP connection setup was required with each request and flow control almost never exited the slow-start phase, limiting performance.
To improve this, new headers were introduced to manage connection lifetime (e.g. "Connection: keep-alive"), blurring the distinction between message metadata and connection metadata. These headers were formalized in HTTP/1.1. This improvement means that connections are reused - when the end of a response has been received, a new request can be sent.

Throttling of simultaneous requests was fully in the realm of implementations, which constrained themselves to opening only a limited number of connections. HTTP/1.1 originally recommended two, but later implementations increased this to six by default, and more under certain conditions. Because these were fully independent flows, TCP was unable to consider them as a group for purposes of congestion control, leading to suboptimal behavior on the network.

Servers which desired additional parallelism could game such implementations by exposing resources under multiple hostnames, causing the client implementations to open six connections _to each hostname_ and gain an arbitrary amount of parallelism, to the detriment of functional congestion control.

There were further attempts to improve the use of TCP. Pipelining, also introduced in HTTP/1.1, allowed the client to eliminate the round-trip that was incurred between the end of the server’s response to one request and the server’s receipt of the client’s next request. However, pipelining increases the problem of head-of-line blocking since a request on a different connection might complete sooner. The client’s inability to predict the length of requested actions limited the usefulness of pipelining.

SMUX [w3c-smux] allowed the use of a single TCP connection to carry multiple channels over which HTTP could be carried. This would permit the server to answer requests in any order. However, this was never broadly deployed.

### 4.3.2. HTTP/1.1 over SCTP

Because SCTP permits the use of multiple simultaneous streams over a single connection, HTTP/1.1 could be mapped with relative ease. Instead of using separate TCP connections, SCTP flows could be used to provide a multiplexing layer. Each flow was reused for new requests after the completion of a response, just as HTTP/1.1 used TCP connections. This allowed for better flow control performance, since the transport could consider all flows together.

SCTP has seen limited deployment on the Internet, though recent experience has shown SCTP over UDP [RFC6951] to be a more viable combination.
4.3.3. HTTP/2 Framing Layer

HTTP/2 introduced a framing layer that incorporated the concept of streams. Because a very large number of idle streams automatically exist at the beginning of each connection, each stream can be used for a single request and response. One stream is dedicated to the transport of control messages, enabling a cleaner separation between metadata about the connection from metadata about the separate messages within the connection.

4.4. Congestion control

The transport is aware of each concurrent request in HTTP/1.1’s mappings to TCP and SCTP. In TCP, because there is only one request at a time, and in SCTP because each request occurs on a separate flow. This means that the transport’s own congestion control services are sufficient, even if sub-optimal in TCP’s case due to multiple independent connections.

Because HTTP/2’s adaptation layer introduces a concurrency construct above the transport, the adaptation layer must also introduce a means of flow control to keep the concurrent transactions from introducing head-of-line blocking above TCP.

4.5. Reliable delivery

There are many ways for a transport to provide reliable delivery of messages. This may take the form of loss recovery, where the loss of packets is detected and the corresponding information retransmitted. Alternately, a transport may proactively send extra information so that the data stream is tolerant to some loss – the full message can be reconstructed after receipt of a sufficient fraction of the transmission.

Because TCP and SCTP both provide reliable delivery mechanisms, there was no need to introduce new service in this area for HTTP mappings. However, the adaptation layers of HTTP mappings over UDP have needed to introduce this concept.

CoAP dedicates a portion of its message framing to indicating whether a given message requires reliability or not. If reliable delivery is required, the recipient acknowledges receipt and the sender continues to repeat the message until the acknowledgment is received. For non-idempotent requests, this means keeping additional state about which requests have already been processed.

Some applications above HTTP are able to provide their own loss-recovery messages, and therefore do not actually require the
guarantees that HTTP provides. HTTP over UDP Multicast is targeted at such applications, and therefore does not provide reliable delivery to applications above it.

4.6. In-order delivery

The sequence numbers used to detect the partial loss of data also permit TCP and SCTP to reassemble data in the order it was originally sent.

HTTP/2 does not actually require a full ordering, but TCP does not offer a way to relax its ordering guarantees. HTTP/2 has two ordering requirements:

- All frames on a stream must be delivered to the application in order
- All frames bearing header fragments must be delivered to HPACK in order

UDP mappings of HTTP must define mechanisms to restore the original order of message fragments. HTTPU(M) and the base form of CoAP both do this by restricting messages to the size of a single datagram, while [I-D.ietf-core-block] extends CoAP to define an in-order delivery mechanism in the adaptation layer.

5. Moving Forward

The networks over which we run TCP/IP today look nothing like the networks for which TCP/IP was originally designed. It is the clean separation between TCP, IP, and the lower-layer protocols which has enabled the continued usefulness of the higher-layer protocols as the substrate has changed.

The goal is not merely architectural purity, but modularity. HTTP has enjoyed a long life as a higher-layer protocol and is useful to many varied applications. As transports continue to evolve, we will almost certainly find ourselves in the position of defining a mapping of HTTP onto a new transport once again. With a clear understanding of the HTTP semantic layer and the services it requires, we can better scope the requirements of a new adaptation layer while reusing the components of previous adaptation layers that provide the necessary service well in existing implementations.
6. Informative References

[goland-http-udp]

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-17 (work in progress), March 2015.

[I-D.natarajan-http-over-sctp]

[I-D.tsvwg-quic-protocol]


Author’s Address

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com