Decomposing the Hypertext Transfer Protocol

draft-bishop-httpbis-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 September 15, 2016.

Copyright Notice

Copyright (c) 2016 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

Bishop Expires September 15, 2016 [Page 1]
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. 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 request. 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 which 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 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. "Trailing" headers are split, with the key declared in advance, but the value coming only after the body has been transferred.

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 and the headers provided.

3. Transport Services Required

The HTTP Semantic Layer depends on the availability of several services from its lower layer:

- Reliable delivery
- In-order delivery
- Partial delivery
- Separate request/response, metadata, and payload
- Flow control and throttling

In this section, each of these properties will be discussed at a high level with a focus on why HTTP requires these properties to be
3.1. Reliable delivery

HTTP does not provide the concept to higher layers that fragments of data were received while others were not. If a request is sent, it is assumed that either a response will arrive or the transport will report an error. HTTP itself is not concerned with any intermediate states.

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.

It is worth noting that some consumers of HTTP have relaxed requirements in this space - while HTTP itself has no notion of lossy delivery, some mappings do have weakened guarantees and are only appropriate for scenarios where those weakened guarantees are acceptable.

3.2. In-order delivery

The headers of each message must arrive before any body, since they dictate how the body will be processed. The body is typically exposed as a bytestream which can be read from sequentially, though there are some consumers who are able to use incomplete fragments of certain resource types.

Regardless of the ability to surface and use fragmentary pieces of an HTTP message, the HTTP layer requires the transport be able to ultimately provide a correct ordering and full reconstruction of each message.

3.3. Partial delivery

While only some users of HTTP (client or server) are able to deal with unordered fragments of an HTTP message, it is almost universally necessary to deal with HTTP messages in pieces. There are multiple reasons why that may be necessary:

- The message may be too large to maintain in memory at once (the download of a large file)
- The beginning of a request may be sufficient to generate a response (error due to lack of authorization)

- The message may be constructed incrementally, sending each segment as it becomes available

Regardless, HTTP needs the transport to begin sending the message before the end of the message is available.

3.4. Separate request/response, metadata, and payload

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 or response contains metadata about the message (headers) and an optional body.

These are separate constructs in HTTP, and mechanisms to carry them and keep them appropriately associated must be provided. Note that it’s not actually expected that any _generic_ transport layer would or should have this property, but is nonetheless involved in transporting HTTP messages.

3.5. Flow control and throttling

Flow control is a necessary property of any transport. Because no network can handle an uncontrolled burst of data at infinite speeds, the transport must determine an appropriate sustained data rate for the intervening network. Even in the presence of a nearly-infinite network capacity, the remote server will also have limits on its ability to consume data.

In order to avoid overwhelming either the network or the server, HTTP requires a mechanism to limit sending data rates as well as to limit the rate of new requests going to a server. Although it is optimal for a server to know about all outstanding client requests (even if it chooses not to work on them immediately), the server may wish to protect itself by limiting the memory commitment to outstanding data or requests. The transport should facilitate such protection on the part of a server (or client, in certain scenarios).

3.6. Other desirable properties

There are several properties not properly required for the implementation of HTTP, but which users of HTTP have come to assume are present.
3.6.1. Parallelism

Because a client will often desire a single server to perform multiple actions at once, all HTTP mappings provide the ability to deliver requests in parallel and allow the server to respond to each request as the actions complete. Head-of-line blocking is a particular problem here that transports must attempt to avoid - client requests should ideally reach the server as quickly as possible, allowing the server to choose the correct order in which to handle the requests (with input from the client). Any situation in which a request remains unknown to the server until another request completes is suboptimal.

3.6.2. Security

Integrity and confidentiality are valuable services for communication over the Internet, and HTTP is no exception. While authentication, message integrity, and secrecy are not inherently _required_ for the implementation of HTTP, they are advantageous properties for any mapping to have, so that each party can be sure that what they received is what the other party sent.

Privacy, the control of what data is leaked to the peer and/or third parties, is also a desirable attribute. However, this extends well beyond the scope of any particular mapping and into the use of HTTP.

TLS [RFC5246] is commonly used in mappings to provide this service, and itself requires reliable, in-order delivery. When those services are not provided by the underlying transport, the mapping must either provide those services to TLS as well as HTTP (as in QUIC) or a variant of TLS which provides those services for itself must be substituted (DTLS [RFC6347], as used in CoAP).

3.6.3. Efficiency

While it would be technically possible to define HTTP over a highly inefficient transport or mapping (e.g. format messages in Baudot code, transporting them to the server using avian carriers as in [RFC1149]), there is little reason for applications to use such inefficient mappings when efficient transport mappings exist.

Efficiency can be characterized on many levels:

- Reducing the number of bytes required to transport a message, either through lower overhead or better compression
- Reducing the time from request generation to response receipt
- Reducing the amount of computation or memory required to process or route a request
- Reducing the power consumption required to generate or process a request

4. The Transport Adaptation Layer

No present transport over which HTTP has been mapped actually provides all of the services on which the HTTP Semantic Layer depends. 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, as well as any additional features the mapping finds to be desirable.

In the following table, we can see multiple transports over which HTTP has been deployed and the services which the underlying transports 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>Reliable delivery</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>In-order delivery</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Partial delivery</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Separate metadata and payload</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Flow control &amp; throttling</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Some mappings contain entirely new protocol machinery constructed specifically to serve as an adaptation layer and carried within the transport (HTTP/2 framing over TCP). Others rely on implementation-level meta-protocol behavior (simultaneous TCP connections handled in parallel) not visible to the transport. Because the existence of these adaptation layers 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 arbitrary
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. (In the table above, QUIC is referenced as a generic transport; the HTTP-over-QUIC mapping is discussed below.)

4.1. HTTP/1.x over TCP

Since HTTP/1.x is defined over TCP, many of the necessary services are provided by the transport, enabling a relatively simple mapping. However, there were a number of conventions introduced to fill lacks in the underlying transport.

4.1.1. Metadata and framing

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 header values even after being unframed.

4.1.2. Parallelism and request limiting

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. However, this blurring made it difficult for some implementations to correctly identify the presence and length of bodies, making request-smuggling attacks possible as in [watchfire-request-smuggling].

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.

### 4.1.3. Security

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.

### 4.1.4. Attempts to improve the TCP mapping

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.

### 4.2. HTTP/1.x 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. HTTP/2 over TCP

HTTP/2, also a TCP mapping, attempted to improve the mapping of HTTP to TCP without introducing changes at the semantic level.

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.

4.3.1. Framing and Parallelism

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.

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 on each stream. 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.

4.3.2. Congestion and flow control

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. This led HTTP/2 to create a flow-control scheme within the adaptation layer in addition to TCP’s flow control algorithms.

In HTTP/1.1, this was not needed - the application simply reads from TCP as space is available, and allow’s TCP’s own flow control to govern. In HTTP/2, this would cause severe head-of-line blocking due to the increased parallelism, and so the control must be exerted at a higher level.

Another drawback to the application-layer multiplexing approach is the fact that TCP’s congestion-avoidance mechanisms cannot identify the flows separately, magnifying the impact of packet losses. This manifests both by reducing the congestion window for the entire connection (versus one-sixth of the "connection" in HTTP/1.1) on packet loss, and delayed delivery of packets on unaffected streams due to head-of-line blocking behind lost packets.

4.3.3. Security

HTTP/2 directly defines how TLS may be used to provide security services as part of its adaptation layer.

4.4. HTTPU(M) and CoAP

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.

Adaptation layers of HTTP mappings over UDP have also needed to introduce mechanisms for reliable delivery. 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.

4.5. QUIC over UDP, or HTTP/2 over QUIC, or…?

QUIC is an overloaded term. QUIC is a rich HTTP mapping to UDP
[I-D.tsvwg-quic-protocol] which implements many TCP- and SCTP-like
behaviors in its adaptation layer. It describes itself this way:

QUIC (Quick UDP Internet Connection) is a new multiplexed and
secure transport atop UDP, designed from the ground up and
optimized for HTTP/2 semantics. While built with HTTP/2 as the
primary application protocol, QUIC builds on decades of transport
and security experience, and implements mechanisms that make it
attractive as a modern general-purpose transport. QUIC provides
multiplexing and flow control equivalent to HTTP/2, security
equivalent to TLS, and connection semantics, reliability, and
congestion control equivalent to TCP.

Consequently, QUIC is _also_ a "general-purpose transport" over which
an HTTP mapping can be defined and implemented.

This division makes it unclear which parts belong to the transport
versus an HTTP mapping on top of this new transport. For example,
[I-D.tsvwg-quic-protocol] does define how to separately transport the
headers and body of an HTTP message. However, this capability is
likely not relevant in a general-purpose transport and might better
be removed from QUIC-the-transport and incorporated into HTTP-over-
QUIC.

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. Likewise, the actions and content carried
over HTTP look very different, reflecting well on the abstraction
achieved by the HTTP layer.

It is the layer between HTTP and the transport where abstraction has
not always been successfully achieved. New capabilites in transports
have required new expressions at the HTTP layer to take advantage of
them, and mappings have defined concepts which are tightly bound to
the underlying transport without clearly separating them from the semantics of HTTP.

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]

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

[I-D.tsvwg-quic-protocol]


Author’s Address

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com