WP Protocol v1

1: Overview

The two biggest performance bottlenecks in HTTP are that it needs multiple connections to enable multiple simultaneous downloads, and that it uses a single request/response model and is thus incompatible with streaming data. Using multiple connections increases efficiency with many handshakes, and is more susceptible to the effects of high latency. Furthermore, TCP's slow start mechanism makes these many connections highly inefficient.

WP multiplexes multiple, concurrent data streams over any reliable transport protocol (such as TCP). Using a similar request/response style to HTTP, WP is structured around streaming data responses, which enables on-going, asynchronous data streams. With full support for text headers, the transition from HTTP to WP requires minimal effort from the web application writer.

1.1: Definitions

- client: The endpoint initialising the WP session.
- connection: A transport-layer connection between two endpoints.
- endpoint: Either the client or server of a connection.
- packet: A header-prefixed sequence of bytes sent over a WP session.
- server: The endpoint which did not initiate the WP session.
- session: A synonym for a connection.
- stream: A bi-directional flow of bytes across a virtual channel within a WP session.

2: WP sessions

2.1: Sessions

The WP session runs atop a reliable transport layer such as TCP. The client is the TCP connection initiator. WP connections are persistent connections.

For best performance, it is expected that clients will not close open connections until the user navigates away from all web pages referencing a connection, or until the server closes the connection. Servers are encouraged to leave connections open for as long as possible, but can terminate idle connections if necessary.

2.2: Frames

Once the connection is established, clients and servers exchange framed messages. Frames always have a set of common fields which is one byte long.

The first three bits enumerate the frame type, which determines the rest of the frame's structure. This is followed by a further five bits which are used to store either frame flags (such as the 'Finish' flag used to end streams), or a ping id (used in ping frames). The common fields are designed to make parsing and writing of frames easy, and to give quick understanding of a frame's size.

All WP integer values, including stream id and cache timestamps, are in network byte order (big endian).

2.3: Streams

Streams are independent sequences of bi-directional data divided into frames with several properties:

- Streams may be created by either the client or server.
- Streams can concurrently send data interleaved with other streams.
- Streams may be cancelled.

2.3.1: Stream Frames

WP defines 3 control frames to manage the lifecycle of a stream:

- Data Request - Open a new client stream.
- Data Push - Open a new server stream.
- Stream Error - Close a stream.

2.3.2: Stream Creation

A stream is created with one of two methods. Client streams are opened with a Data Request, while server streams are created with a Data Push. If the server is initiating the stream, the Stream ID must be even. If the client is initiating the stream, the Stream ID must be odd. 0 is never a valid Stream ID. Stream IDs from each side of the connection must increase monotonically as new streams are created. E.g. Stream 2 may be created after stream 3, but stream 7 must not be created after stream 9. Stream IDs do not wrap: when a client or server cannot create a new stream id without exceeding a 16 bit value, it MUST NOT create a new stream.

The Stream ID MUST increase with each new stream. If an endpoint receives a Data Request or Push with a Stream ID which is less than any previously received Data Request or Push (as appropriate), it MUST issue a Stream Error (Section 2.4) with the status Protocol_Error.

It is a protocol error to send two stream-initiating frames with the same Stream ID. If a recipient receives a second stream-initiating frame for the same stream, it MUST issue a Stream Error (Section 2.4.2) with the status code Stream_In_Use, though the existing stream continues.

Upon receipt of a stream-initiating frame, the recipient can reject the stream by sending a Stream Error with the error code Refused_Stream. Note, however, that the creating endpoint may have already sent additional frames for that stream which cannot immediately be stopped.

Once the stream is created, the creator may immediately send Data Content frames for that stream, without needing to wait for the recipient to acknowledge.

2.3.3: Stream Data Exchange

Once a stream is created, it can be used to send arbitrary amounts of data. Generally this means that a series of Data Content frames will be sent on the stream until a frame containing the 'Finish' flag is set. The 'Finish' flag can be set on a Data Request (Section 2.6.3), Data Response (Section 2.6.4), or Data Content (Section 2.6.6) frame. The same effect is caused by a Data Push frame with the 'Suggest' flag set. Alternatively, a Stream Error frame may be sent with the Finish_Stream status.

2.3.4: Stream Close

There are 3 ways that streams can be terminated:

- Normal termination: Normal stream termination occurs when either endpoint sends a frame with the 'Finish', or 'Suggest' flag set, or a Stream Error frame with the Finish_Stream status, as described above.
- Abrupt termination: Either the client or server can send a Stream Error frame at any time. A Stream Error contains a status code to indicate the reason for failure. When a Stream Error is sent from the stream originator, it indicates a failure to complete the stream and that no further data will be sent on the stream. When a Stream Error is sent from the stream recipient, the sender, upon receipt, should stop sending any data on the stream. Depending on the status code accompanying the Stream Error, further action may be required, such as ending the entire session. The stream recipient should be aware that there is a race between data already in transit from the sender and the time the Stream Error is received. See Stream Error Handling (Section 2.4)
- TCP connection teardown: If the TCP connection is torn down while un-closed streams exist, then the endpoint must assume that the stream was abnormally interrupted and may be incomplete.

If an endpoint receives a data frame after the stream is closed, it must send a Stream Error to the sender with the status Stream_Closed.

2.4: Error Handling

2.4.1: Session Error Handling

A session error is any error which prevents further processing of the connection. When a session error occurs, the endpoint encountering the error MUST first send a Stream Error (Section 2.6.2) frame with the status code for why the session is terminating. After sending the Stream Error frame, the endpoint MUST close the TCP connection.

Note that because this Stream Error is sent during a session error case, it is possible that the frame will not be reliably received by the receiving endpoint. It is a best-effort attempt to communicate with the remote about why the session is going down.

2.4.2: Stream Error Handling

A stream error is an error related to a specific Stream ID which does not affect processing of other streams in the session. Upon a stream error, the endpoint MUST send a Stream Error frame which contains the Stream ID identifying where the error occurred and the error status code indicating which error took place. After sending the Stream Error, the stream is closed. After sending the Stream Error, if the sender receives any frames for that stream id, it will result in sending additional Stream Error frames. An endpoint MUST NOT send a stream error in response to a stream error, as doing so would lead to Stream Error loops. In some cases, however, a session error may follow a stream error.

If an endpoint has multiple Stream Error frames to send in succession for the same Stream ID and the same status code, it MAY coalesce them into a single Stream Error frame.

2.5: Data Flow

Because TCP provides a single stream of data on which SPDY multiplexes multiple logical streams, clients and servers must intelligently interleave data messages for concurrent sessions.

2.6: Frame Types

2.6.1: Hello

Hello frames are used to send data about the connection that would otherwise be repeated. HTTP clients send details like the User-Agent string with every request, although it is unlikely to change during a single

WP connection. The same applies to the protocol version, choice of language, and can often apply to authentication details like some cookies.

Every WP connection MUST begin with the client sending a Hello frame, to which the server MAY reply with its own, but this is optional. Other frames may follow the client's Hello immediately. Further Hello frames may be sent by either endpoint at any time, although there are some constraints. Subsequent Hellos MUST NOT set the user-agent, protocol version, or hostname, and they MUST be discarded by the recipient. The WP version MUST not change during a session, since it would become unclear which parts of the connection are governed by which version. Although allowing multiple host names would mean that a server could use one connection for multiple hosts, it would add substantial complexity to the protocol state. Stream-specific host names may be set with the text headers sent in frames to that stream.

The protocol version is an unsigned integer larger than zero. Use of version zero causes an end to the connection immediately. The Hostname and Referrer work identically to their HTTP equivalents. It is acceptable for many Hello frames to be sent with different Referrers, to communicate that the user may have returned to the same host from a different origin. The User-Agent string is used as in HTTP, but has a strict structure, as described in Requests (Section 3.2.1).

The language preference differs from HTTP, in that it takes a binary structure. The Language size is a single byte, giving the number of language preferences. If this value is zero, then no preference is given. Each byte of the Language indicates a specific language, in order of preference. For example, a size of 0x2 and Languages of 0x1a, 0x19 indicates en-gb, en; where English is accepted, but British English is preferred. The full table of language codes for WPv1 follows:

0 - af - Afrikaans
1 - ach - Luo
2 - ak - Akan
3 - am - Amharic
4 - ar - Arabic
5 - az - Azerbaijani
6 - be - Belarusian
7 - bem - Bemba
8 - bg - Bulgarian
9 - bh - Bihari
10 - bn - Bengali
11 - br - Breton
12 - bs - Bosnian
13 - ca - Catalan
14 - chr - Cherokee
15 - ckb - Kurdish (Soranî)
16 - co - Corsican
17 - crs - Seychellois Creole
18 - cs - Czech
19 - cy - Welsh
20 - da - Danish
21 - de - German
22 - ee - Ewe
23 - el - Greek
24 - en - English
25 - en-gb - English (United Kingdom)

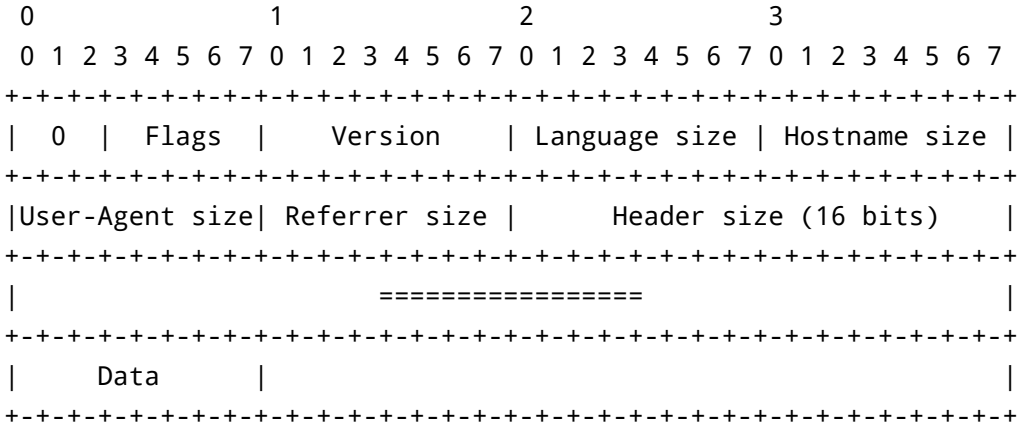
26 - en-us - English (United States)
27 - eo - Esperanto
28 - es - Spanish
29 - es-419 - Spanish (Latin American)
30 - et - Estonian
31 - eu - Basque
32 - fa - Persian
33 - fi - Finnish
34 - fo - Faroese
35 - fr - French
36 - fy - Frisian
37 - ga - Irish
38 - gaa - Ga
39 - gd - Scots Gaelic
40 - gl - Galician
41 - gn - Guarani
42 - gu - Gujarati
43 - ha - Hausa
44 - haw - Hawaiian
45 - hi - Hindi
46 - hr - Croatian
47 - ht - Haitian Creole
48 - hu - Hungarian
49 - hy - Armenian
50 - ia - Interlingua
51 - id - Indonesian
52 - ig - Igbo
53 - is - Icelandic
54 - it - Italian
55 - iw - Hebrew
56 - ja - Japanese
57 - jw - Javanese
58 - ka - Georgian
59 - kg - Kongo
60 - kk - Kazakh
61 - km - Cambodian
62 - kn - Kannada
63 - ko - Korean
64 - kri - Krio (Sierra Leone)
65 - ku - Kurdish
66 - ky - Kyrgyz
67 - la - Latin
68 - lg - Luganda
69 - ln - Lingala
70 - lo - Laothian
71 - loz - Lozi
72 - lt - Lithuanian
73 - lua - Tshiluba
74 - lv - Latvian
75 - mfe - Mauritian Creole

76 - mg - Malagasy
77 - mi - Maori
78 - mk - Macedonian
79 - ml - Malayalam
80 - mn - Mongolian
81 - mo - Moldavian
82 - mr - Marathi
83 - ms - Malay
84 - mt - Maltese
85 - ne - Nepali
86 - nl - Dutch
87 - nn - Norwegian (Nynorsk)
88 - no - Norwegian
89 - nso - Northern Sotho
90 - ny - Chichewa
91 - nyn - Runyakitara
92 - oc - Occitan
93 - om - Oromo
94 - or - Oriya
95 - pa - Punjabi
96 - pcm - Nigerian Pidgin
97 - pl - Polish
98 - ps - Pashto
99 - pt-BR - Portuguese (Brazil)
100 - pt-PT - Portuguese (Portugal)
101 - qu - Quechua
102 - rm - Romansh
103 - rn - Kirundi
104 - ro - Romanian
105 - ru - Russian
106 - rw - Kinyarwanda
107 - sd - Sindhi
108 - sh - Serbo-Croatian
109 - si - Sinhalese
110 - sk - Slovak
111 - sl - Slovenian
112 - sn - Shona
113 - so - Somali
114 - sq - Albanian
115 - sr - Serbian
116 - sr-ME - Montenegrin
117 - st - Sesotho
118 - su - Sundanese
119 - sv - Swedish
120 - sw - Swahili
121 - ta - Tamil
122 - te - Telugu
123 - tg - Tajik
124 - th - Thai
125 - ti - Tigrinya

126 - tk - Turkmen
127 - tl - Filipino
128 - tn - Setswana
129 - to - Tonga
130 - tr - Turkish
131 - tt - Tatar
132 - tum - Tumbuka
133 - tw - Twi
134 - ug - Uighur
135 - uk - Ukrainian
136 - ur - Urdu
137 - uz - Uzbek
138 - vi - Vietnamese
139 - wo - Wolof
140 - xh - Xhosa
141 - yi - Yiddish
142 - yo - Yoruba
143 - zh-CN - Chinese (Simplified)
144 - zh-TW - Chinese (Traditional)
145 - zu - Zulu
251 - xx-bork - Bork bork bork!
252 - xx-elmer - Elmer Fudd
253 - xx-hacker - Hacker
254 - xx-klingon - Klingon
255 - xx-pirate - Pirate

There has been space left for new entries to be added in later versions of the protocol. Receipt of a Language not in this table constitutes a protocol error.

Further rarely-changing information may be attached in the Header section, such as static cookies, or authentication data. All such data must be data applying to the connection as a whole, not to a subset of the connection's streams. This should be sent in MIME encoding, as with HTTP headers. All data sent in Hello frames must be made available to every stream at either endpoint.



Flags: None defined in WPv1. Must be 0.
Version: WP version number. (8 bits)

```
                - Must not be 0.
Language size:  See languages above (8 bits).
                - 0 means accept all.
Hostname size:  Number of bytes in hostname (8 bits).
User-Agent size: Number of bytes in the user agent.
Referrer size:  Number of bytes in the referrer URI.
Header size:    Number of bytes in any remaining headers.
Data:           Length-specified fields and headers named above.
```

2.6.2: Stream Error

```
 0               1               2
 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  1  |  Flags  |       Stream ID (16 bits)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Status    |      Status Data (16 bits)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
Flags:      No flags defined in WPv1. Must be 0.
Stream ID:  See streams (Section 2.3.2) (16 bits).
Status:     8 bit code, 16 bit data:
            List of codes:
            0: [INVALID].
            1: Protocol error (ends session).
            2: Unsupported version (send version, end session).
            3: Invalid stream (send expected stream ID).
            4: Refused stream (send expected stream ID).
            5: Stream closed.
            6: Stream in use (send expected stream ID).
            7: Stream ID missing.
            8: Internal error (ends session).
            9: Finish stream.
```

2.6.3: Data Request

```
 0               1               2               3
 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  2  |  Flags  |       Stream ID (16 bits)     |               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Resource size (16 bits)    |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|                      Header size (32 bits)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Cache Timestamp                        |
|                           (64 bits)                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       =================                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Res (0-65535) | Headers (cookies etc)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Flags:
      Upload:    Indicates an upload request.
Stream ID:       See streams (Section 2.3.2) (16 bits).
Resource size:   Number of bytes in the resource name.
                 - 0 is invalid and triggers Protocol_Error.
Header size:     Number of bytes in the request header.
Cache Timestamp: Timestamp of cached resource.
                 - 0 means not cached.
Res:             Resource URI.
Headers:         Header content.
```

2.6.4: Data Response

Response Codes:

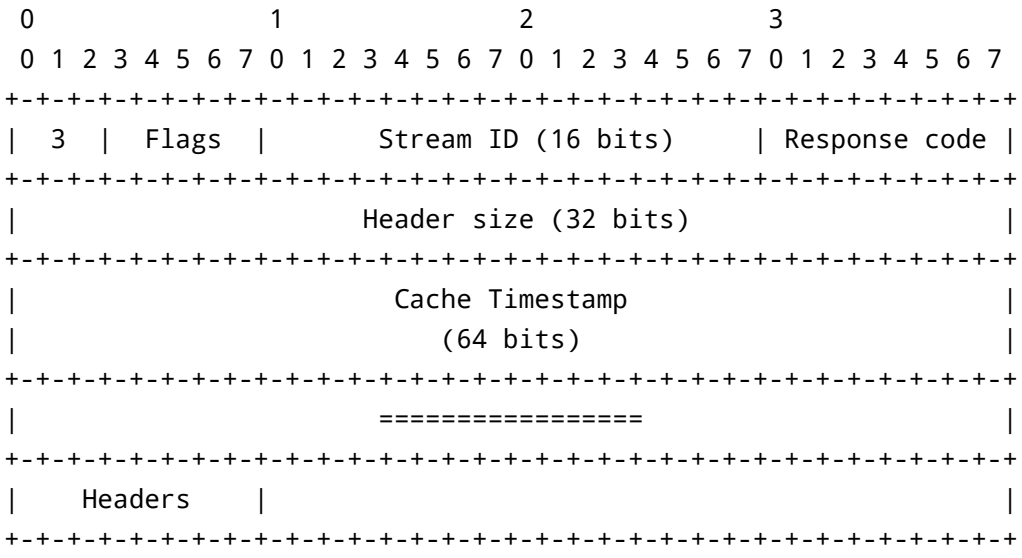
Response types:
- 0 - Success
- 1 - Redirection
- 2 - Client error
- 3 - Server error

Response subtypes:
- 0/0 - Success
- 0/1 - Cached
- 0/2 - Partial

- 1/0 - Moved temporarily
- 1/1 - Moved permanently

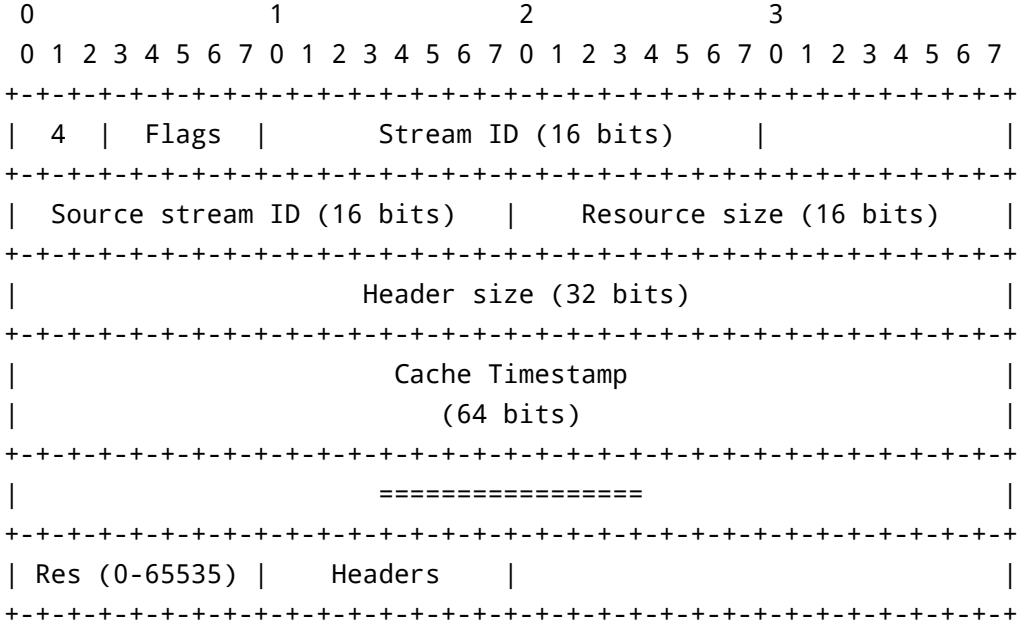
- 2/0 - Bad request
- 2/1 - Forbidden
- 2/2 - Not found
- 2/3 - Removed

- 3/0 - Server error
- 3/1 - Service unavailable
- 3/2 - Service timeout




```
Flags:
      Finish:  See stream closing (Section 2.3.4).
Stream ID:     See streams (Section 2.3.2) (16 bits).
Response code: 2-bit response type, 6-bit subtype.
Header size:   Number of bytes in the response header.
Cache:         Time of when resource becomes invalid.
             - 0 means do not cache.
```

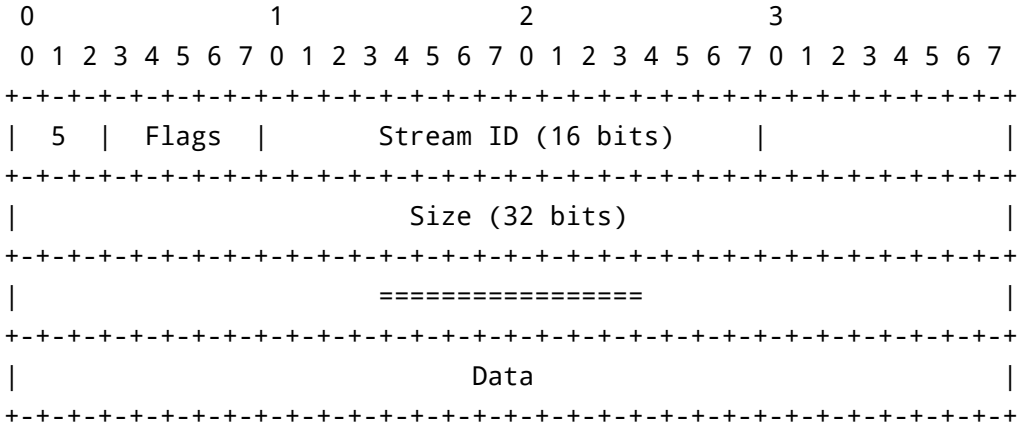
2.6.5: Data Push




```
Flags:
```

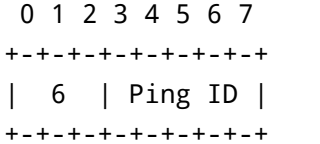
```
      Suggest:    This is just a suggestion.
Stream ID:        See streams (Section 2.3.2) (16 bits).
Source stream:    The request which triggered this push.
Resource size:    Number of bytes in the URI.
Header size:      Number of bytes in the push header.
Cache timestamp:  Timestamp of when file was last changed.
                  - 0 means do not cache (non-suggest only).
Res:              Resource URI.
```

2.6.6: Data Content



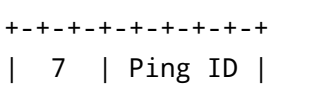

```
Flags:
      Finish: See stream closing (Section 2.3.4).
      Ready:  Data is ready to be processed.
Stream ID:    See streams (WP specification).
Size:         Number of bytes in the whole packet.
```

2.6.7: Ping Request




```
Ping ID:          Number identifying this request (5 bits).
```

2.6.8: Ping Response



```
+-+-+-+-+-+-+-+-+

Ping ID:          Number identifying the request (5 bits).
```

3: WP and HTTP

WP provides fairly similar request/response semantics to HTTP as far as basic usage is concerned. The important differences are discussed below.

3.1: Connection Management

Clients SHOULD NOT open more than one WP session to a given origin concurrently.

Note that it is possible for one SPDY session to be finishing (e.g. all valid Stream IDs for one endpoint have been used, but not all streams have finished), while another WP session is starting.

The standard network port for WP is 99, so user-agents should treat a request for “wp://example.com/” as a request for “wp://example.com:99/”.

3.2: HTTP Request/Response

3.2.1: Request

The client initiates a request by sending a Data Request frame. For requests which do not contain a body, the Data Request frame MUST set the ‘Finish’ flag, indicating that the client intends to send no further data on this stream. For requests which do contain a body, the Data Request will not contain the ‘Finish’ flag, and the body will follow the Data Request in a series of Data Content frames. The last Data Content frame will set the ‘Finish’ flag to indicate the end of the body.

The Data Request’s Headers field will contain all of the HTTP-like headers which are associated with the request. The Headers field in WP is mostly unchanged from today's HTTP header block, with the following differences:

- User-Agent values must follow the form: render engine and version, browser and version, OS and version, architecture (using Plan 9 designation).
    - Rather than the following:
        - “Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5) AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.95 Safari/537.11”
    - Use this:
        - “webkit/537.1,chrome/21.0.1180.89,macosx/10.7.4,6”
- Header names are all lowercase.
- User-agents MUST support deflate compression. Regardless of headers like Accept-Encoding sent by the user-agent, the server may always send content encoded with gzip or deflate encoding.

3.2.2: Response

The server responds to a client request with a Data Response frame. Symmetric to the client's upload stream, server will send data after the Data Response frame via a series of Data Content frames, and the last data frame will contain the ‘Finish’ flag to indicate successful end-of-stream. If a response contains no body, the Data Response frame should contain the ‘Finish’ flag to indicate no further data will be sent on the stream.

3.3: Server Push Transactions

WP enables a server to send multiple replies to a client for a single request. The rationale for this feature is that sometimes a server knows that it will need to send multiple resources in response to a single request. Without server push features, the client must first download the primary resource, then discover the secondary resource(s), and request them. Pushing of resources avoids the round-trip delay, but also creates a potential race where a server can be pushing content which a user-agent is in the process of requesting. The following mechanics attempt to prevent the race condition while enabling the performance benefit.

If the browser accepts a pushed response (e.g. it does not send a Stream Error), the browser MUST attempt to cache the pushed response in same way that it would cache any other response. This means validating the response headers and inserting into the cache.

Because pushed responses have no request, they have no request headers associated with them. WP pushed streams contain an "associated Stream ID" which indicates the requested stream for which the pushed stream is related.

Implementation note: With server push, it is theoretically possible for servers to push unreasonable amounts of content or resources to the user-agent. Browsers MUST implement throttles to protect against unreasonable push attacks.

3.3.1: Server implementation

When the server intends to push a resource to the user-agent, it opens a new stream by sending a Data Push. The Data Push MUST include an Associated Stream ID, and the Resource field, which represents the URL for the resource being pushed. The purpose of the association is so that the user-agent can differentiate which request induced the pushed stream; without it, if the user-agent had two tabs open to the same page, each pushing unique content under a fixed URL, the user-agent would not be able to differentiate the requests.

The Associated Stream ID must be the ID of an existing, open stream. The reason for this restriction is to have a clear endpoint for pushed content. If the user-agent requested a resource on stream 11, the server replies on stream 11. It can push any number of additional streams to the client before sending a ‘Finish’ flag on stream 11. However, once the originating stream is closed no further push streams may be associated with it. The pushed streams do not need to be closed before the originating stream is closed, they only need to be created before the originating stream closes.

It is illegal for a server to push a resource with the Associated Stream ID of 0.

To minimize race conditions with the client, the Data Push for the MUST be sent prior to sending any content which could allow the client to discover the pushed resource and request it.

The server MUST only push resources which would have been returned from a Data Request.

3.3.2: Client implementation

When fetching a resource the client has 3 possibilities:

- the resource is not being pushed
- the resource is being pushed, but the data has not yet arrived

- the resource is being pushed, and the data has started to arrive

When a Data Push frame which contains an Associated Stream ID is received, the client must not issue Data Requests for the resource in the pushed stream, and instead wait for the pushed stream to arrive.

If a client receives a server push stream with Stream ID 0, it MUST issue a session error (Section 2.4.1) with the status code Protocol_Error.

To cancel individual server push streams, the client can issue a stream error (Section 2.4.2) with error code Finish_Stream. Upon receipt, the server MUST stop sending on this stream immediately (this is an Abrupt termination).

To cancel all server push streams related to a request, the client may issue a stream error (Section 2.4.2) with error code Finish_Stream on the Associated Stream ID. By cancelling that stream, the server MUST immediately stop sending frames for any streams with in-association-to for the original stream.

4: Security Considerations

4.1: Cross-Protocol Attacks

By utilizing TLS, I believe that WP introduces no new cross-protocol attacks. TLS encrypts the contents of all transmission (except the handshake itself), making it difficult for attackers to control the data which could be used in a cross-protocol attack.

5: Privacy Considerations

5.1: Long Lived Connections

WP aims to keep connections open longer between clients and servers in order to reduce the latency when a user makes a request. The maintenance of these connections over time could be used to expose private information. For example, a user using a browser hours after the previous user stopped using that browser may be able to learn about what the previous user was doing. This is a problem with HTTP in its current form as well, however the short lived connections make it less of a risk.

6: Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

7: Acknowledgements

WP was partly inspired by the SPDY protocol, and works on similar principles. As a result, the content of WP's specification is very similar to SPDY's. To reduce unnecessary rewriting, much of this document is a modified form of the SPDY protocol's 3rd draft specification. Credit goes to Mike Belshe and Roberto Peon for SPDY/3, which is available at http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3 .

Further thanks go to my supervisor, Prof. Omer Rana, whose guidance kept me on track and pointed me in useful directions.

Author's Address

Jamie Hall e-mail: the.sly.marbo@googlemail.com