HTTP/2 Implementation Debug State
draft-benfield-http2-debug-state-01

Abstract

   This document defines a standard format and well-known URI for HTTP/2
   server implementations to expose their internal state for the
   purposes of debugging and interoperability work.

Table of Contents

1. Introduction

   The HTTP/2 [RFC7540] specification provides an alternative framing
   layer for the semantics of HTTP/1.1 [RFC7231]. This alternative
   framing layer includes substantially greater quantities of state to
   be stored by all implementations. Disagreements on the state of the
   connection are the cause of the vast majority of interoperability
   errors in HTTP/2 implementations.

   In general it is not possible for implementations to query the
   internal state of their peer, and those implementations that do
   expose their internal state do it using a number of different
   interfaces, in different places, and in different formats. This
   makes it hard to debug interoperability problems, particularly when
   those problems arise on the open web with implementations that have
   unknown configuration and that may not identify themselves clearly.

This document defines a standard format and well-known URI for HTTP/2
server implementations to make their internal state available for
introspection. This allows both new and established implementers to
do more effective testing of their implementations, as well as to
enable them to more effectively diagnose and report subtle bugs in
both their own and other implementations.

1.1. Notational Conventions

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 [RFC2119].

2. Debug Output

An implementation that wishes to support the HTTP/2 debug state
information does so by publishing a JSON document at a well-known URI
([RFC5785]): specifically, at .well-known/h2/state. This JSON
document reveals aspects of the state of the specific HTTP/2
connection as seen by the implementation in question at the time of
response generation.

This JSON document is represented as a single JSON object with
multiple keys. The object has several mandatory keys, as well as
several optional ones. The fields are outlined below.

2.1. Version

The "version" key in the state object is associated with a string
carrying the version of the debug output specification the debug
output conforms to. For an implementation that supports this draft
specification, the output must be "draft-01".

Sample output:

"version": "draft-01"

                   Figure 1: Example output for version key

2.2. Settings

The "settings" key in the state object is associated with a JSON
object that contains the remote implementation's active settings.
These are the settings that are actually in force for the connection
at this time. This means that if the implementation has emitted a
SETTINGS frame but has not yet received an ACK, the changes in that
SETTINGS frame MUST NOT be reflected in the object.

Each setting is published along with its value. The name of each
setting MUST be the same as its name in [RFC7540] Section 6.5.2: for
example, "SETTINGS_ENABLE_PUSH". The values MUST be sent as JSON
integers.

An implementation MAY omit a setting from this object if it has never
been emitted by the implementation. In this situation it should be
assumed that the default value is in force.

A conforming implementation MUST emit this field.

Sample output:

```
"settings": {
  "SETTINGS_MAX_CONCURRENT_STREAMS": 250,
  "SETTINGS_MAX_FRAME_SIZE": 1048576,
  "SETTINGS_MAX_HEADER_LIST_SIZE": 1048896
}
```

Figure 2: Example output for settings key

2.3. Peer Settings

The "peerSettings" key in the state object is associated with a JSON
object that contains the remote implementation's view of the local
implementation's settings. These are the settings that are actually
in force for the connection at this time.

The value of this key is exactly symmetrical with the value of the
"settings" key: see Section 2.2 for more.

A conforming implementation MUST emit this field.

Sample output:

```
"peerSettings": {
    "SETTINGS_HEADER_TABLE_SIZE": 4096,
    "SETTINGS_ENABLE_PUSH": 1,
    "SETTINGS_INITIAL_WINDOW_SIZE": 6291456,
    "SETTINGS_MAX_FRAME_SIZE": 16384,
    "SETTINGS_MAX_CONCURRENT_STREAMS": 1000
}
```

Figure 3: Example output for peerSettings key

2.4. Outbound Flow Control Window

   The "connFlowOut" key in the state object is associated with a JSON
   integer that reflects the remote peer's outbound connection window
   size. This represents the number of flow controlled bytes the remote
   implementation believes it can emit before the entire connection is
   blocked behind flow control.

   A conforming implementation MUST emit this field.

   Sample output:

   "connFlowOut": 15724175,

                 Figure 4: Example output for connFlowOut key

2.5. Inbound Flow Control Window.

   The "connFlowIn" key in the state object is associated with a JSON
   integer that reflects the remote peer's inbound connection window
   size. This represents the number of flow controlled bytes the remote
   implementation believes it can receive before the entire connection
   is blocked behind flow control.

   A conforming implementation MUST emit this field.

   Sample output:

   "connFlowIn": 65535,

                 Figure 5: Example output for connFlowIn key

2.6. Streams

   The "streams" key in the state object is associated with a JSON
   object containing state about all the active streams on the
   connection. A stream MUST be represnted in this JSON object if it is
   in any state other than IDLE or CLOSED.

   This JSON object has keys that are the stream IDs for the active
   streams. Each key has an object associated with it, with the
   following keys:

   o "state": This key maps to a string value representing the stream
      state. The stream states are represented as all-caps ASCII text
      with all parentheses stripped and spaces replaced with
      underscores. For example, "OPEN" or "HALF_CLOSED_LOCAL". This
      field MUST be present.

o "flowIn": The remote peer's inbound stream window size as a JSON
   integer. This represents the number of flow controlled bytes the
   remote implementation believes it can receive on this stream
   before this stream is blocked behind flow control. This field
   MUST be present.

o "flowOut": The remote peer's outbound stream window size as a JSON
   integer. This represents the number of flow controlled bytes the
   remote implementation believes it can send on this stream before
   this stream is blocked behind flow control. This field MUST be
   present.

o "dataIn": The number of bytes of data the remote implementation
   has received on this stream. This excludes padding bytes. This
   field MAY be present, but is optional.

o "dataOut": The number of bytes of data the remote implementation
   has sent on this stream. This excludes padding bytes. This field
   MAY be present, but is optional.

o "paddingIn": The number of padding bytes the remote implementation
   has received on this stream. This excludes data bytes. This
   field MAY be present, but is optional.

o "paddingOut": The number of padding bytes the remote
   implementation has sent on this stream. This excludes data bytes.
   This field MAY be present, but is optional.

o "queuedData": The number of bytes of data the remote
   implementation has available to send, but has not yet sent. These
   bytes may be blocked behind flow control or priority information:
   the value of the "flowOut" field can be used to distinguish
   between these two cases. This field MAY be present, but is
   optional.

o "created": A timestamp indicating when the peer believes the
   stream first transitioned out of the idle state (see [RFC7540]
   Section 5.1). This time stamp must be in the form of a Unix time
   stamp: that is, a number representing the number of seconds since
   00:00:00 Thursday 1 January 1970 UTC. This number may have any
   number of decimal digits. This field MAY be present, but is
   optional.

A conforming implementation MUST emit this field, but MAY omit any of
the optional sub-fields.

Sample output:

```
      "streams": {
        "5": {
          "state": "HALF_CLOSED_REMOTE",
          "flowIn": 65535,
          "flowOut": 6291456,
          "dataIn": 0,
          "dataOut": 0,
          "paddingIn": 0,
          "paddingOut": 0,
          "created": 1470835059.619137
        },
        "7": {
          "state": "OPEN",
          "flowIn": 65535,
          "flowOut": 6291456,
          "queuedData": 59093,
        }
      },
```

                   Figure 6: Example output for streams key

2.7. HPACK

   The "hpack" key contains information about the HPACK compression
   state for the connection. It maps to a JSON object that represents
   this compression state.

   This JSON object contains the following fields:

   o "inboundTableSize": The current size of the HPACK dynamic header
      table for the headers emitted by the local implementation, as an
      integer. This field MUST be present.

   o "outboundTableSize": The current size of the HPACK dynamic header
      table for the headers emitted by the remote implementation, as an
      integer. Note that this value MUST include the headers added to
      the compression context as part of serving this response. This
      field MUST be present.

   o "inboundDynamicHeaderTable": The entries added to the HPACK
      dynamic header table by the local implementation. This is
      formatted as a JSON array of two-element JSON arrays, the first
      element of which contains the header name and the second element
      of which contains the header value. This field MAY be omitted.

   o "outboundDynamicHeaderTable": The entries added to the HPACK
      dynamic header table by the remote implementation. This is

      formatted in the same manner as "outboundDynamicHeaderTable".
      This field MAY be omitted.

   A conforming implementation MAY omit this field. If it does include
   this field, it MAY omit any optional sub-fields.

   Sample output:

   "hpack": {
       "inboundTableSize": 340,
       "inboundDynamicHeaderTable": [
           [
               "accept-encoding",
               "gzip, deflate, sdch, br"
           ],
           [
               "upgrade-insecure-requests",
               "1"
           ],
           [
               "cache-control",
               "max-age=0"
           ],
           [
               ":authority",
               "shootout.lukasa.co.uk"
           ]
       ],
       "outboundTableSize": 137,
       "outboundDynamicHeaderTable": [
           [
               "content-type",
               "application/json"
           ],
           [
               "server",
               "TwistedWeb/16.3.0"
           ]
       ]
   }

                   Figure 7: Example output for hpack key

2.8. Sent GoAway

   The "sentGoAway" field tracks whether or not a GOAWAY frame
   ([RFC7540] Section 6.8) has been sent on the connection by the remote
   implementation. The value of this field is boolean.

A conforming implementation MAY omit this field.

Sample output:

"sentGoAway": false,

                    Figure 8: Example output for sentGoAway key

2.9. Additional Fields

In addition to these fields, implementations MAY add their own
debugging information, as appropriate, to the JSON object. These
MUST be keyed off keys other than the ones defined in this document.
For example, some implementations are known to expose the number of
threads they currently have active in the "threads" field.

3. Debug Headers

One of the most common issues when implementing HTTP/2 is to have
problems with flow control windows. This is why the "connFlowOut"
(Section 2.4) and "connFlowIn" (Section 2.5) fields are defined in
the JSON document.

However, it's possible that the two implementations disagree on the
size of this window, and that the server believes that it cannot send
the response body because it's blocked behind flow control. For this
reason, a small amount of debugging data MUST be inserted into the
response headers for this JSON document. This ensures that it is
possible for implementations to discover that they have inadvertently
blocked the debug response behind flow control, and to take action to
widen the flow control window so that the response can be delivered.

The following header fields MUST be emitted by implementations.

3.1. Flow In

The "conn-flow-in" header field contains the size of the remote
implementation's inbound flow control window. The field value
contains only the size of that window in octets. This MUST be
calculated the same way that the implementation calculates
"connFlowIn" (Section 2.5).

3.2. Flow Out

The "conn-flow-out" header field contains the size of the remote
implementation's outbound flow control window. The field value
contains only the size of that window in octets. This MUST be

calculated the same way that the implementation calculates
"connFlowOut" (Section 2.4).

4. Security Considerations

4.1. HPACK State

For a single-hop HTTP/2 connection there is no risk in exposing the
HPACK state to the client, as the only entity that can possibly have
affected the HPACK state is the client itself.

However, once intermediaries are considered this stops being true.
If any intermediary is performing connection coalescing, the HPACK
state will almost certainly include entries inserted into the dynamic
table by or for multiple clients. Exposing this state will put the
security and privacy of those other clients at risk.

For this reason, if it is at all possible that a server implementing
this specification may have an intermediary on a connection between
itself and a client, the server MUST NOT emit the "hpack" key or any
of its sub-fields. It is only safe to emit this key in controlled
environments.

5. IANA Considerations

5.1. Well-known URI

This document establishes a single well-known URI, with the suffix
"h2/state".

6. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
           Requirement Levels", BCP 14, RFC 2119,
           DOI 10.17487/RFC2119, March 1997,
           <http://www.rfc-editor.org/info/rfc2119>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known
           Uniform Resource Identifiers (URIs)", RFC 5785,
           DOI 10.17487/RFC5785, April 2010,
           <http://www.rfc-editor.org/info/rfc5785>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
           Protocol (HTTP/1.1): Semantics and Content", RFC 7231,
           DOI 10.17487/RFC7231, June 2014,
           <http://www.rfc-editor.org/info/rfc7231>.

   [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext
              Transfer Protocol Version 2 (HTTP/2)", RFC 7540,
              DOI 10.17487/RFC7540, May 2015,
              <http://www.rfc-editor.org/info/rfc7540>.

Appendix A. Defined Fields

   This appendix contains tables of all defined fields, along with their
   field names, field value type, optionality, the versions in which
   they appear, and what section defines them.

   For fields whose values are objects, there are additional tables
   defining the fields in those sub-objects, with the same information.

   This can be used as a quick reference point.

Field Name	Field Type	Optional?	Versions	Section
version	String	No	draft-01 ..	Section 2.1
settings	Object	No	draft-01 ..	Section 2.2
peerSettings	Object	No	draft-01 ..	Section 2.3
connFlowOut	Number	No	draft-01 ..	Section 2.4
connFlowIn	Number	No	draft-01 ..	Section 2.5
streams	Object	No	draft-01 ..	Section 2.6
hpack	Object	Yes	draft-01 ..	Section 2.7
sentGoAway	Boolean	Yes	draft-01 ..	Section 2.8

                       Table 1: Top-level Fields

A.1. Settings Sub-Fields

   This table lists the sub-fields of the "settings" and "peerSettings"
   values, each of which is a single JSON object containing the
   following fields.

```
+--------------------------------+---------+-----------+-----------+
| Field Name                     | Field   | Optional? | Versions  |
|                                | Type    |           |           |
+--------------------------------+---------+-----------+-----------+
| SETTINGS_HEADER_TABLE_SIZE     | Number  | Yes       | draft-01  |
|                                |         |           | ..        |
|                                |         |           |           |
| SETTINGS_ENABLE_PUSH           | Number  | Yes       | draft-01  |
|                                |         |           | ..        |
|                                |         |           |           |
| SETTINGS_MAX_CONCURRENT_STREAMS | Number | Yes       | draft-01  |
|                                |         |           | ..        |
|                                |         |           |           |
| SETTINGS_INITIAL_WINDOW_SIZE   | Number  | Yes       | draft-01  |
|                                |         |           | ..        |
|                                |         |           |           |
| SETTINGS_MAX_FRAME_SIZE        | Number  | Yes       | draft-01  |
|                                |         |           | ..        |
|                                |         |           |           |
| SETTINGS_MAX_HEADER_LIST_SIZE  | Number  | Yes       | draft-01  |
|                                |         |           | ..        |
+--------------------------------+---------+-----------+-----------+
```

                  Table 2: settings and peerSettings Sub-Fields

A.2. Streams Sub-Fields

   This table lists the sub-fields of the "streams" value. The
   "streams" object is defined more thoroughly in section Section 2.6.
   All of the fields defined here appear in the objects that are the
   values of the "streams" sub-keys.

```
+------------+-----------+----------+-------------+
| Field Name | Field Type | Optional? | Versions   |
+------------+-----------+----------+-------------+
| state      | String    | No       | draft-01 .. |
|            |           |          |             |
| flowIn     | Number    | No       | draft-01 .. |
|            |           |          |             |
| flowOut    | Number    | No       | draft-01 .. |
|            |           |          |             |
| dataIn     | Number    | Yes      | draft-01 .. |
|            |           |          |             |
| dataOut    | Number    | Yes      | draft-01 .. |
|            |           |          |             |
| paddingIn  | Number    | Yes      | draft-01 .. |
|            |           |          |             |
| paddingOut | Number    | Yes      | draft-01 .. |
|            |           |          |             |
| queuedData | Number    | Yes      | draft-01 .. |
|            |           |          |             |
| created    | Number    | Yes      | draft-01 .. |
+------------+-----------+----------+-------------+
```

                  Table 3: Stream Sub-Fields

A.3. HPACK Sub-Fields

   This table lists the sub-fields of the "hpack" value, each of which
   is a single JSON object containing the following fields.

```
+----------------------------+--------------+----------+----------+
| Field Name                 | Field Type   | Optional? | Versions |
+----------------------------+--------------+----------+----------+
| inboundTableSize           | Number       | No       | draft-01 |
|                            |              |          | ..       |
|                            |              |          |          |
| outboundTableSize          | Number       | No       | draft-01 |
|                            |              |          | ..       |
|                            |              |          |          |
| inboundDynamicHeaderTable  | List of list | Yes      | draft-01 |
|                            | of String    |          | ..       |
|                            |              |          |          |
| outboundDynamicHeaderTable | List of list | Yes      | draft-01 |
|                            | of String    |          | ..       |
+----------------------------+--------------+----------+----------+
```

                   Table 4: HPACK Sub-Fields

[Appendix B](#). **Acknowledgements**

   We would like to thank the attendees of the 2016 HTTP Workshop in
   Stockholm for their feedback on early prototype implementations of
   this debugging feature.

[Appendix C](#). **Changelog**

   This appendix to be deleted by the RFC editor.)

   Since -00:

   o Changed URI from "/.well-known/h2interop/state" to "/.well-
      known/h2/state".

   o Changed keys of "hpack" entry to all be camel-case, rather than
      snake-case.

   o Added the "version" top-level key.

   o Added the "created" sub-key to the "stream" objects.

   o Added the "queuedData" sub-key to the "stream" objects.

   o Added the "paddingIn" and "paddingOut" sub-keys to the "stream"
      objects.

   o Added appendix documenting all field values.

Authors' Addresses

   Cory Benfield
   Hewlett Packard Enterprise

   Email: cory@lukasa.co.uk

   Brad Fitzpatrick
   Google, Inc.

   Email: brad@danga.com