HTTPbis Working Group
Internet-Draft
Expires: May 12, 2014

M. Bishop
Microsoft

November 08, 2013

Extension Frames in HTTP/2.0
draft-bishop-http2-extension-frames-00

Abstract

This document describes a proposed modification to the HTTP/2.0 specification to better support the creation of extensions without the need to version the core protocol or invoke additional protocol identifiers.

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 May 12, 2014.

Table of Contents

1. Introduction

HTTP/2.0 currently offers an inconsistent story about the use of extensions. As the draft currently stands, extensions have the following traits:

- They may define SETTINGS values, provided they do not modify identifiers within the base HTTP/2.0 spec

- They may define new frame types, provided they do not:

  Conflict with the base HTTP/2.0 frame types

  Modify session state

  Require understanding in order to communicate over the base protocol

This poses a number of problems for such extension frames. To begin with, there is no way to know whether the peer supports a given extension before sending extension-specific information. This is addressed in the current spec by saying that implementations MUST ignore frame types and settings values they don't understand, but sending information that your peer cannot parse wastes bandwidth. Further, it shackles extensions since they are prohibited from modifying session state.

As an additional concern, with only 256 frame types it is conceivable that the frame type space may be exhausted if many extensions are defined. It more than conceivable that different extensions will

collide with each other in the choice of frame type identifiers, since the space is limited. Requiring IANA to register frame type identifiers is onerous, since the number and types of the frames has changed often during HTTP/2.0 development and there is no reason to expect that a complex extension would do otherwise. This should be balanced against the goal of defining a simple, single-frame extension and being able to quickly allocate this single frame type.

Future versions of the HTTP/2.0 specification will face exactly the same problem as extension authors, since they currently share a frame type and setting value space with any extensions. Thus, a new frame introduced with HTTP/2.1+ must avoid collision with HTTP/2.0 extensions and must deal with space exhaustion. Any means of resolving such adoption after the fact complicates forward-porting of existing extensions.

This document proposes an alternative method of supporting extension frames and settings, with the following goals:

- Reduce the probability of collision among extensions and between extensions and future versions of HTTP
- Enable peers to completely disable all extensions
- Enable peers to selectively disable an extension without requiring knowledge of the extension they wish to disable
- Enable peers to quickly discover support for a particular extension on the far side

1.1. Conventions and Terminology

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].

All numeric values are in network byte order. Values are unsigned unless otherwise indicated. Literal values are provided in decimal or hexadecimal as appropriate. Hexadecimal literals are prefixed with "0x" to distinguish them from decimal literals.

2. Extension Functionality

2.1. Extension Identification

An extension to HTTP/2.0 is identified by an Extension ID. An Extension ID is a 24-bit identifier with two parts: A sixteen-bit private enterprise number and an eight-bit enterprise-local extension identifier. Private enterprise numbers are issued by IANA, with an existing registry [IANA-PEN].

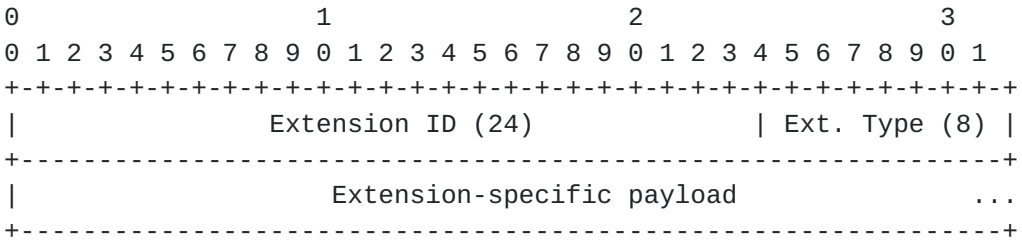
An organization is free to utilize any method to allocate and track the assignment of its organization-local extensions, though expert review of proposed extensions is strongly recommended. If all 256 values have been allocated, the organization may request an additional organizational identifier. Extensions allocated by the IETF will be tracked by IANA.

It is suggested that each organization reserve an extension ID (for instance, 0x00) for low-volume single-frame extensions, and allocate these frame types with a local registry. This avoids consuming an entire extension ID for a single new frame type, at the expense of being unable to separately disable these single-frame extensions. Such extensions MUST NOT modify the state of the base protocol in any circumstance, since support for the specific frame cannot be detected. Extensions which require several frames or the ability to have support separately detected should be allocated a separate ID.

The private enterprise number 0x000000, allocated to IANA, is reserved in this context for IETF RFCs. Extension ID 0x00000000 is reserved for values defined in base HTTP specifications and single-frame extensions defined by the HTTP working group.

2.2. Extension Frames

The EXTENSION frame (0xFF) carries content which is specific to an extension. The EXTENSION frame contains a sub-header of the following format:



Extension Payload Format

The payload of an EXTENSION frame includes the Extension ID of the extension which should be used to process the frame, as well as an extension-local frame type, permitting any extension up to 256 distinct frame types which it has complete freedom to define.

EXTENSION frames may be sent on any stream which is half-open in the sender's direction (that is, a stream on which the sender could send a DATA frame) or on stream zero.

The semantics of EXTENSION frames are defined by their respective extensions, with the following restrictions:

- An extension frame on stream zero with semantics which modify the state of the base protocol MUST NOT be sent until the remote peer has indicated support for the extension.

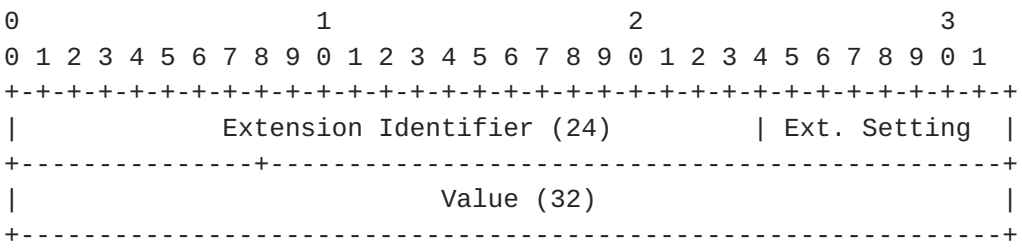
- An extension frame on a non-zero stream MUST NOT modify the state of the base protocol.

2.3. Extension Settings

An implementation which does not intend to interpret or relay any extension frames or setting values SHOULD send the setting value BASE_SPECIFICATION_ONLY (0xFF) set to a non-zero value. Upon receipt of this setting value, an implementation MUST NOT send any setting value or frame not defined in the specification of the negotiated protocol.

In order to minimize round-trips, it is advisable to exchange any necessary initial state as early as possible. This means that critical information should be included in the initial SETTINGS frame, in order to benefit from the transmission of this frame in advance of any other frames. However, to avoid bloating this initial frame, extensions are encouraged to send only the minimum amount of information necessary for the extension to be useful in the initial SETTINGS frame. Information which is not critical for bootstrapping SHOULD be sent in a subsequent SETTINGS frame, or in an extension-specific frame type.

This draft updates the definition of a setting value as follows:



Modified Setting Value Format

The Extension Setting field with a value equal to zero is reserved for extension identification. An implementation which wishes to inform a peer that it will be employing a certain extension MAY set this to any non-zero value. Interpretation of the non-zero value is extension-dependent, and may communicate some initial state or preferences. An implementation which wishes to explicitly inform a peer that it does not understand, does not wish to use, or will not relay a certain extension MAY set this value to zero.

The semantics of all other values of Extension Setting are defined by the extension. These semantics MUST NOT modify the state the base protocol. (If this is needed, define an extension-specific frame to carry settings.) These setting values MUST NOT be sent for an extension the sender does not implement, and MUST be ignored upon receipt if not understood.

2.4. Intermediary Behavior

EXTENSION frames sent on stream zero are explicitly hop-by-hop and MUST be ignored by intermediaries which do not understand the specified Extension ID. In order to ensure consistent state, an intermediary which is not terminating the HTTP request MUST provide one of the following behaviors to each stream other than zero:

- Forward all extension frames on this stream
- Discard all extension frames on this stream

Because distinct streams on a given connection may be related to different backend servers or different clients, these behaviors MAY vary from stream to stream without notification to the peer on either side. An intermediary which intends to discard all extension frames on all streams SHOULD set the BASE_SPECIFICATION_ONLY (Section 2.3, Paragraph 1) setting to avoid wasted effort on a peer's part.

2.5. Base Specification Behavior

Since extensions now have a segregated frame and settings space, the receipt of any non-EXTENSION frame type not defined in the negotiated protocol is a session error of (new) type INVALID_FRAME_TYPE.

2.6. IANA Considerations

This draft employs the Private Enterprise Number registry, already maintained by IANA, to avoid creating a nearly duplicate registry specific to HTTP/2.0. However, due to the number of bytes available, it restricts this registry from 32 bits to 16 bits. This still allows enough space for the current number of registrations to double, but it does sharply increase the possibility of eventual exhaustion.

One possible mitigation would be to expand the Extension ID from 24 to 32 bits, expanding the space for the PEN to 24 bits. This would require removing the extension frame sub-type and setting value space for extensions, constraining extensions to a single 32-bit setting value; extensions which require more than 32 bits for settings would need to define an equivalent to the SETTINGS frame. In the same way, extensions which require multiple frame types would need to define a frame type field inside their payload.

The proper resolution here remains an open issue, and discussion is one goal of this draft.

3. References

[HPACK] Ruellan, H. and R. Peon, "HPACK - Header Compression for HTTP/2.0", draft-ietf-httpbis-header-compression-04 (work in progress), October 2013.

[IANA-PEN] , "Private Enterprise Numbers", .

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels ", BCP 14, RFC 2119, March 1997.

[SnellExtensions] Snell, J., "HTTP Extensions", draft-snell-httpbis-ext-frames-00 (work in progress), May 2013.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax ", STD 66, RFC 3986, January 2005.

Appendix A. Example Extensions

This section describes how certain extensions might leverage the EXTENSION frame. It is explicitly not an attempt to define the specific extension at this time, though the ideas have been discussed and should be explored in the future.

Other sample extensions can be found in another Internet Draft [SnellExtensions] submitted to this working group.

A.1. Server Hint

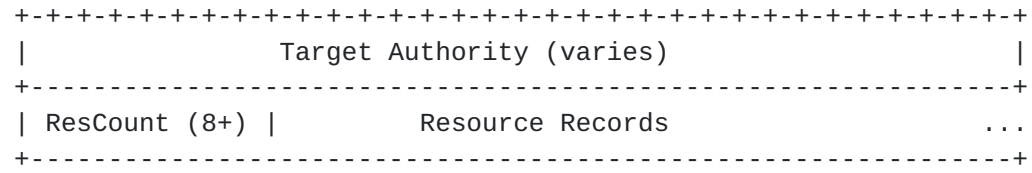
Server Push has many attractive features, because it can eliminate both the RTT needed to send a request and the DOM processing time to realize a resource will be needed. However, it suffers from several drawbacks as well:

- The server cannot know what the client has in its cache, leading to wasted data transmission
- The server cannot push resources for other origins, specifically resources which may be hosted on CDNs of which the server is aware
- The server must fully specify the client request, which it will do inaccurately

Server Hint defines a mechanism which sacrifices the RTT savings, but retains the ability to initiate requests before DOM parsing (or even document transfer) has completed and overcomes the stated drawbacks of server push. While not intended as a replacement for Server Push, it may be a useful complement or replacement when a PUSH_PROMISE is not appropriate.

A.1.1. Format

The format of the Server Hint frame is as follows:



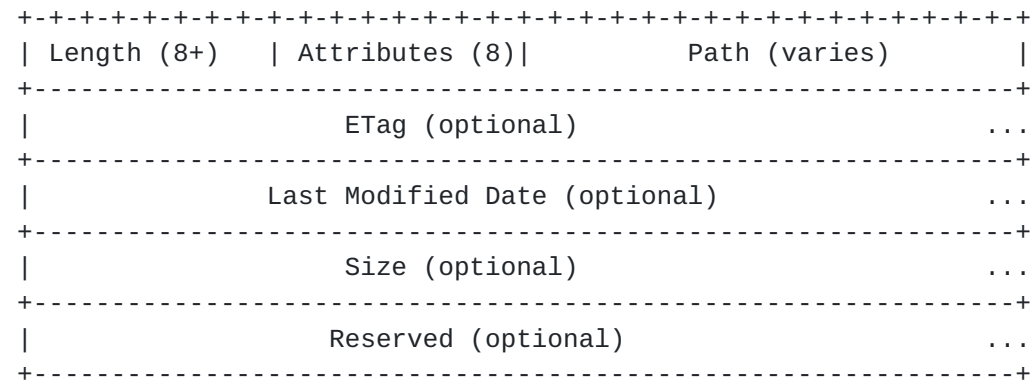
Server Hint Extension Frame

Target Authority specifies the scheme and authority components of the URI, which apply all resources in this frame. (Resources which should be retrieved from a different authority or using a different

scheme MUST be sent in a separate frame.) This URI fragment is formatted as a length-prefixed Huffman-encoded string, as defined in HPACK [HPACK].

ResCount is an 8-bit-prefix variable-length integer, also as defined in HPACK [HPACK]. It defines how many Resource Records for the specified authority follow.

The format of each Resource Record is as follows:



Server Hint Extension Frame

Length: An 8-bit prefix integer giving the number of bytes in the resource record

Attributes: A bitmask specifying which optional pieces of information the server has opted to include:

* 0x80: Set if ETag is present

* 0x40: Set if Last-Modified Date is present

* 0x20: Set if Size is present

The remaining bits are reserved.

Path: A length-prefixed Huffman-encoded string as specified in HPACK [HPACK] containing the path (and optional query) component(s) of the resource URI [URI].

ETag: A length-prefixed Huffman-encoded string as specified in HPACK [HPACK], containing the ETag as specified in RFC 2616.

Last-Modified Date: Last-Modified Date is the HTTP Date converted into the number of seconds since 1970-01-01 0:00 UTC formatted as an 8-bit prefix variable-length integer.

Size: The content length, formatted as an 8-bit prefix variable-length integer.

Reserved: MUST be zero-length when sent, and unknown fields in the Reserved space MUST be ignored.

A.1.2. Behavior

A server MAY send a Server Hint frame multiple times on a stream. It may be sent only in a circumstance where a PUSH_PROMISE frame would have been permissible, except for the value of PUSH_ENABLED. A single frame contains a list of resources accessible under a single authority, but multiple instances of the frame MAY be sent to refer clients to resources available from multiple authorities.

Upon receipt of a Server Hint frame, a client MUST check its cache for a corresponding resource. If the resource is not available in the cache, it SHOULD open connections to the specified authority and request the resource.

When processing a Server Hint frame containing no resources or in which all resources are already available from cache, a client MAY prepare to make other requests in various ways, such as beginning DNS resolution, connection establishment, etc.

Appendix B. Acknowledgements

This document includes input from Martin Thompson and Gabriel Montenegro.

Author's Address

Mike Bishop
Microsoft

EMail: michbish@microsoft.com