Extension Mechanism for the Babel Routing Protocol

Abstract

   This document defines the encoding of extensions to the Babel routing
   protocol, as specified in RFC 6126.

Status of This Memo

   This document is not an Internet Standards Track specification; it is
   published for examination, experimental implementation, and
   evaluation.

   This document defines an Experimental Protocol for the Internet
   community. This is a contribution to the RFC Series, independently
   of any other RFC stream. The RFC Editor has chosen to publish this
   document at its discretion and makes no statement about its value for
   implementation or deployment. Documents approved for publication by
   the RFC Editor are not a candidate for any level of Internet
   Standard; see Section 2 of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc7557.

Table of Contents

1. Introduction

   A Babel packet [RFC6126] contains a header followed by a sequence of
   TLVs, each of which is a sequence of octets having an explicit type
   and length. The original Babel protocol has the following provisions
   for including extension data:

   o A Babel packet with a version number different from 2 MUST be
      silently ignored ([RFC6126], Section 4.2).

   o An unknown TLV MUST be silently ignored ([RFC6126], Section 4.3).

   o Except for Pad1 and PadN, all TLVs are self-terminating, and any
      extra data included in a TLV MUST be silently ignored ([RFC6126],
      Section 4.2).

   o The Flags field of the Update TLV contains 6 undefined bits that
      MUST be silently ignored ([RFC6126], Section 4.4.9).

   o Any data following the last TLV of a Babel packet MUST be silently
      ignored ([RFC6126], Section 4.2).

   Each of these provisions provides a place to store data needed by
   extensions of the Babel protocol. However, in the absence of any
   further conventions, independently developed extensions to the Babel
   protocol might make conflicting uses of the available space, and
   therefore lead to implementations that would fail to interoperate.

This document formalises a set of rules for extending the Babel
protocol that are designed to ensure that no such incompatibilities
arise, and that are currently respected by a number of deployed
extensions.

In the rest of this document, we use the term "original protocol" for
the protocol defined in [RFC6126], and "extended protocol" for any
extension of the Babel protocol that follows the rules set out in
this document.

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 [RFC2119].

2. Mechanisms for Extending the Babel Protocol

   This section describes each of the mechanisms available for extending
   the Babel protocol.

2.1. New Versions of the Babel Protocol

   The header of a Babel packet contains an eight-bit protocol version.
   The current version of the Babel protocol is version 2; any packets
   containing a version number different from 2 MUST be silently
   ignored.

   Versions 0 and 1 were earlier experimental versions of the Babel
   protocol that have seen some modest deployment; these version numbers
   SHOULD NOT be reused by future versions of the Babel protocol.
   Version numbers larger than 2 might be used by a future incompatible
   protocol.

2.2. New TLVs

   An extension may carry its data in a new TLV type. Such new TLVs
   will be silently ignored by implementations of the original Babel
   protocol, as well as by other extended implementations of the Babel
   protocol, as long as the TLV types do not collide.

   All new TLVs MUST have the format defined in [RFC6126], Section 4.3.
   New TLVs SHOULD be self-terminating, in the sense defined in the next
   section, and any data found after the main data section of the TLV
   SHOULD be treated as a series of sub-TLVs.

   TLV types 224 through 254 are reserved for Experimental Use
   [RFC3692]. TLV type 255 is reserved for expansion of the TLV type
   space, in the unlikely event that eight bits turn out not to be
   enough.

2.3. Sub-TLVs

   With the exception of the Pad1 TLV, all Babel TLVs carry an explicit
   length. With the exception of Pad1 and PadN, all TLVs defined by the
   original protocol are self-terminating, in the sense that the length
   of the meaningful data that they contain (the "natural length") can
   be determined without reference to the explicitly encoded length. In
   some cases, the natural length is trivial to determine: for example,
   a HELLO TLV always has a natural length of 2 (4 including the Type
   and Length fields). In other cases, determining the natural length
   is not that easy, but this needs to be done anyway by an
   implementation that interprets the given TLV. For example, the
   natural length of an Update TLV depends on both the prefix length and
   the amount of prefix compression being performed.

   If the explicit length of a TLV defined by the original protocol is
   larger than its natural length, the extra space present in the TLV is
   silently ignored by an implementation of the original protocol;
   extended implementations MAY use it to store arbitrary data and
   SHOULD structure the additional data as a sequence of sub-TLVs.
   Unlike TLVs, the sub-TLVs themselves need not be self-terminating.

   An extension MAY be assigned one or more sub-TLV types. Sub-TLV
   types are assigned independently from TLV types: the same numeric
   type can be assigned to a TLV and a sub-TLV. Sub-TLV types are
   assigned globally: once an extension is assigned a given sub-TLV
   number, it MAY use this number within any TLV. However, the
   interpretation of a given sub-TLV type can depend on which particular
   TLV it is embedded within.

   Sub-TLV types 224 through 254 are reserved for Experimental Use
   [RFC3692]. TLV type 255 is reserved for expansion of the sub-TLV
   type space, in the unlikely event that eight bits turn out not to be
   enough. The format of sub-TLVs is defined in Section 3 below.

2.4. The Flags Field

   The Flags field is an eight-bit field in the Update TLV. Bits 0 and
   1 (the bits with values 80 and 40 hexadecimal) are defined by the
   original protocol and MUST be recognised and used by every
   implementation. The remaining six bits are not currently used and
   are silently ignored by implementations of the original protocol.

   Due to the small size of the Flags field, it is NOT RECOMMENDED that
   one or more bits be assigned to an extension; a sub-TLV SHOULD be
   assigned instead. An implementation MUST ignore any bits in the
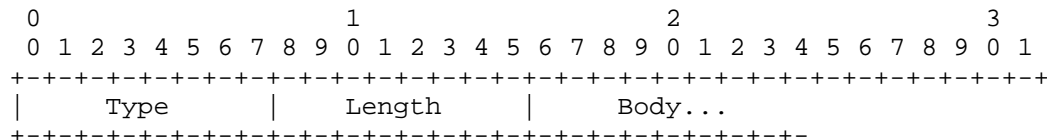   Flags field that it does not know about and MUST send them as zero.

2.5. Packet Trailer

   A Babel packet carries an explicit length in its header. A Babel
   packet is carried by a UDP datagram, which in turn contains an
   explicit length in its header. It is possible for a UDP datagram
   carrying a Babel packet to be larger than the size of the Babel
   packet. In that case, the extra space after the Babel packet, known
   as the packet trailer, is silently ignored by an implementation of
   the original protocol.

   The packet trailer was originally intended to be used as a
   cryptographic trailer. However, the authentication extension to
   Babel [RFC7298] ended up using a pair of new TLVs, and no currently
   deployed extension of Babel uses the packet trailer. The format and
   purpose of the packet trailer is therefore currently left undefined.

3. Format of Sub-TLVs

   A sub-TLV has exactly the same structure as a TLV. Except for Pad1
   (Section 3.1.1), all sub-TLVs have the following structure:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     |    Body...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

   Fields:

   Type   The type of the sub-TLV.

   Length   The length of the body, in octets, exclusive of the Type
             and Length fields.

   Body   The sub-TLV body, the interpretation of which depends on
             both the type of the sub-TLV and the type of the TLV within
             which it is embedded.

3.1. Sub-TLVs Specified in This Document

   This document defines two types of sub-TLVs, Pad1 and PadN. These
   two sub-TLVs MUST be correctly parsed and ignored by any extended
   implementation of the Babel protocol that uses sub-TLVs.

3.1.1. Pad1

```
   0
   0 1 2 3 4 5 6 7
  +-+-+-+-+-+-+-+-+
  |    Type = 0   |
  +-+-+-+-+-+-+-+-+
```
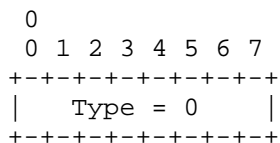
   Fields:

   Type   Set to 0 to indicate a Pad1 sub-TLV.

   This sub-TLV is silently ignored on reception.

3.1.2. PadN

```
   0                   1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |    Type = 1   |    Length     |      MBZ...
  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```
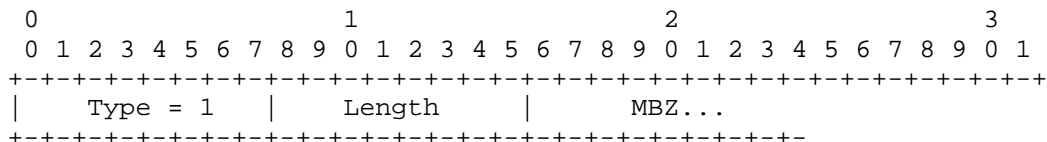
   Fields:

   Type   Set to 1 to indicate a PadN sub-TLV.

   Length   The length of the body, in octets, exclusive of the Type
               and Length fields.

   MBZ   Set to 0 on transmission.

   This sub-TLV is silently ignored on reception.

3.2. Unknown Sub-TLVs

   Any unknown sub-TLV MUST be silently ignored by an extended
   implementation that uses sub-TLVs.

4. Choosing between Extension Mechanisms

   New versions of the Babel protocol should only be defined if the new
   version is not backwards compatible with the original protocol.

   In many cases, an extension could be implemented either by defining a
   new TLV or by adding a new sub-TLV to an existing TLV. For example,
   an extension whose purpose is to attach additional data to route
   updates can be implemented either by creating a new "enriched" Update
   TLV or by adding a sub-TLV to the Update TLV.

The two encodings are treated differently by implementations that do
not understand the extension. In the case of a new TLV, the whole
unknown TLV is ignored by an implementation of the original protocol,
while in the case of a new sub-TLV, the TLV is parsed and acted upon,
and the unknown sub-TLV is silently ignored. Therefore, a sub-TLV
should be used by extensions that extend the Update in a compatible
manner (the extension data may be silently ignored), while a new TLV
must be used by extensions that make incompatible extensions to the
meaning of the TLV (the whole TLV must be thrown away if the
extension data is not understood).

Using a new bit in the Flags field is equivalent to defining a new
sub-TLV while using less space in the Babel packet. Due to the
limited Flags space, and the doubtful space savings, we do not
recommend the use of bits in the Flags field -- a new sub-TLV should
be used instead.

We refrain from making any recommendations about the usage of the
packet trailer due to the lack of implementation experience.

5. IANA Considerations

   IANA has created three new registries, called "Babel TLV Types",
   "Babel Sub-TLV Types", and "Babel Flags Values". The allocation
   policy for each of these registries is Specification Required
   [RFC5226].

The initial values in the "Babel TLV Types" registry are as follows:

Type	Name	Reference
0	Pad1	[RFC6126]
1	PadN	[RFC6126]
2	Acknowledgment Request	[RFC6126]
3	Acknowledgment	[RFC6126]
4	Hello	[RFC6126]
5	IHU	[RFC6126]
6	Router-Id	[RFC6126]
7	Next Hop	[RFC6126]
8	Update	[RFC6126]
9	Route Request	[RFC6126]
10	Seqno Request	[RFC6126]
11	TS/PC	[RFC7298]
12	HMAC	[RFC7298]
13	Source-specific Update	[BABEL-SS]
14	Source-specific Request	[BABEL-SS]
15	Source-specific Seqno Request	[BABEL-SS]
224-254	Reserved for Experimental Use	this document
255	Reserved for expansion of the type space	this document

The initial values in the "Babel Sub-TLV Types" registry are as
follows:

Type	Name	Reference
0	Pad1	this document
1	PadN	this document
2	Diversity	[BABEL-DIV]
3	Timestamp	[BABEL-RTT]
224-254	Reserved for Experimental Use	this document
255	Reserved for expansion of the type space	this document

The initial values in the "Babel Flags Values" registry are as
follows:

Bit	Name	Reference
0	Default prefix	[RFC6126]
1	Default router-id	[RFC6126]
2-7	Unassigned	

6. Security Considerations

   This document specifies the structure of fields that are already
   present in the original Babel protocol and does not, by itself, raise
   any new security considerations. Specific extensions may change the
   security properties of the protocol, for example, by adding security
   mechanisms [RFC7298] or by enabling new kinds of attack.

7. References

7.1. 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>.

   [RFC3692]   Narten, T., "Assigning Experimental and Testing Numbers
                Considered Useful", BCP 82, RFC 3692,
                DOI 10.17487/RFC3692, January 2004,
                <http://www.rfc-editor.org/info/rfc3692>.

   [RFC5226]   Narten, T. and H. Alvestrand, "Guidelines for Writing an
                IANA Considerations Section in RFCs", BCP 26, RFC 5226,
                DOI 10.17487/RFC5226, May 2008,
                <http://www.rfc-editor.org/info/rfc5226>.

   [RFC6126]   Chroboczek, J., "The Babel Routing Protocol", RFC 6126,
                DOI 10.17487/RFC6126, April 2011,
                <http://www.rfc-editor.org/info/rfc6126>.

7.2. Informative References

   [BABEL-DIV] Chroboczek, J., "Diversity Routing for the Babel Routing
                Protocol", Work in Progress, draft-chroboczek-babel-
                diversity-routing-00, July 2014.

   [BABEL-RTT] Jonglez, B. and J. Chroboczek, "Delay-based Metric
                Extension for the Babel Routing Protocol", Work in
                Progress, draft-jonglez-babel-rtt-extension-01, May 2015.

   [BABEL-SS]   Boutier, M. and J. Chroboczek, "Source-Specific Routing
                in Babel", Work in Progress, draft-boutier-babel-
                source-specific-01, May 2015.

   [RFC7298]   Ovsienko, D., "Babel Hashed Message Authentication Code
                (HMAC) Cryptographic Authentication", RFC 7298,
                DOI 10.17487/RFC7298, July 2014,
                <http://www.rfc-editor.org/info/rfc7298>.

Acknowledgments

Author's Address

   Juliusz Chroboczek
   PPS, University of Paris-Diderot
   Case 7014
   75205 Paris Cedex 13
   France

   EMail: jch@pps.univ-paris-diderot.fr